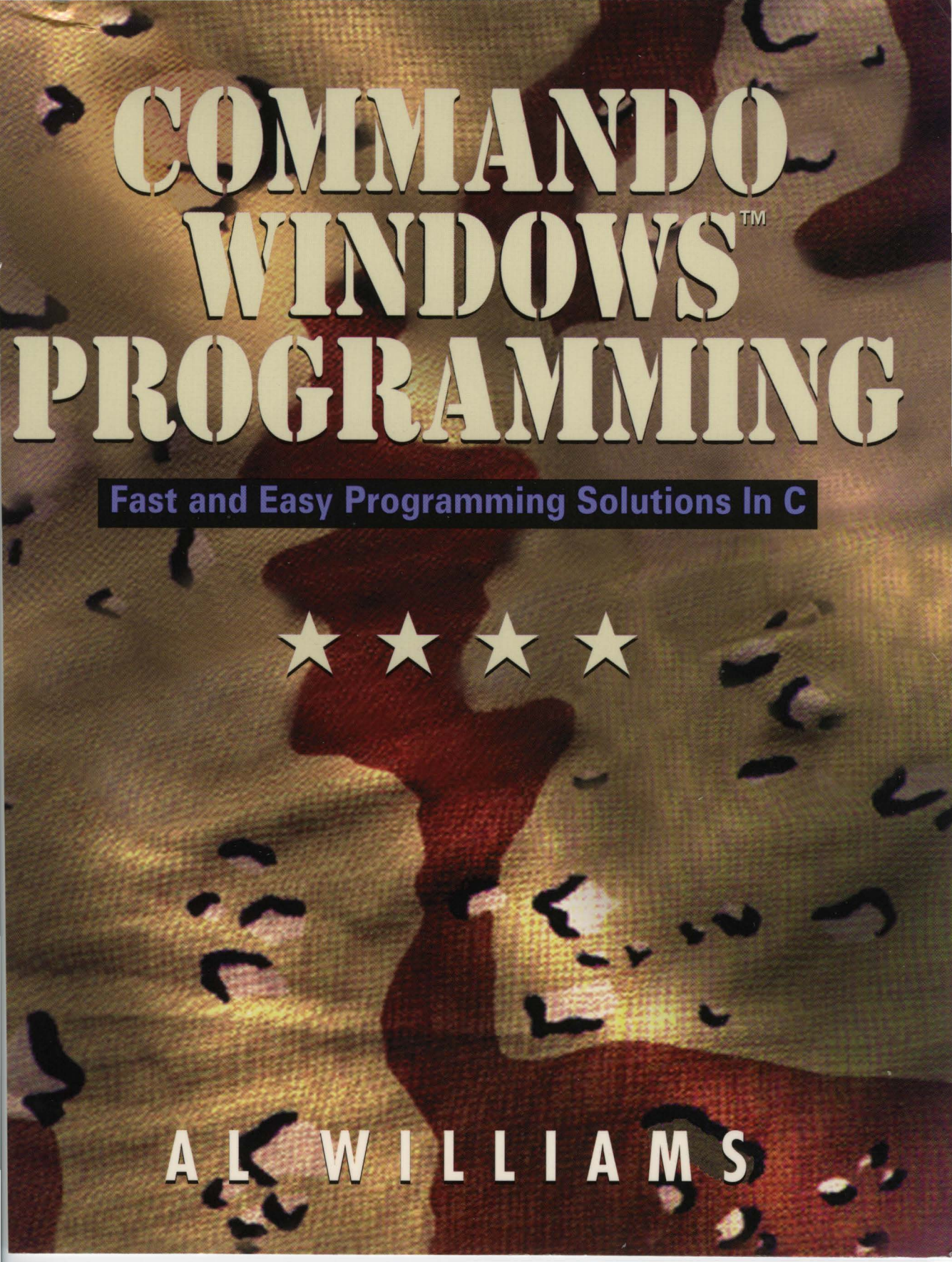
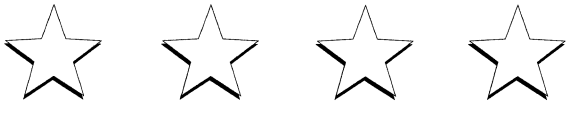

COMMANDO WINDOWS™ PROGRAMMING
Fast and Easy Programming Solutions In C
AL WILLIAMS

COMMANDO WINDOWS™ PROGRAMMING

COMMANDO WINDOWS™ PROGRAMMING

Fast and Easy Programming Solutions in C



AL WILLIAMS

Addison-Wesley Publishing Company
Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan
Paris Seoul Milan Mexico City Taipei

Library of Congress Cataloging-in-Publication Data

Williams, Al, 1963-

Commando Windows programming : fast and easy programming solutions in C / Al Williams.

p. cm.

Includes index.

ISBN 0-201-62484-2

1. Windows (Computer programs) I. Title.

QA76.6.W56W49 1993

005.4'3--dc20 93-24862

CIP

Managing Editor: Amorette Pedersen

Production Editor: Jennifer Noble

Set in 11-point Palatino by Benchmark Productions, Inc.

1 2 3 4 5 6 7 8 9-MA-97 96 95 94 93

First Printing, July 1993

For my mother, Jerid, Amy, and always for Pat.

Contents

Listings

1

2

5

6

7

Acknowledgments

My appreciation to Andrea Mulligan, Amy Pedersen, Andrew Williams, and Chris Williams along with everyone else at Benchmark and Addison-Wesley for their usual great job. Thanks to John Hamilton for showing me the need for this book and Larry Coates for his excellent editorial comments. And last, but never least, thanks to my family for helping me make the leap to full-time writing and consulting.

A version of the VWINL library appeared in *Dr. Dobb's Journal* and is used here with permission (thanks to Jon Erickson).

Introduction

Open most Windows C/C++ programming books and you'll find information about event loops, GDI, device contexts, and update routines—a daunting barrier to the beginning Windows programmer.

The arcane approach is fine if you are a student with plenty of time to learn Windows' intricacies. But you live in the real world. You don't write drawing or word processing programs; you write real applications to access databases, fill in forms, and print reports. You need to write these programs fast, and you don't have much time to learn every detail about Windows. You are a commando programmer.

Luckily there are many techniques and tools to help commando programmers code for Windows and Windows NT. *Commando Windows Programming* covers:

- *Writing dialog-only and menu-only programs*

- *Emulating text-based programs with edit controls*
- *Using libraries to simplify application creation*
- *Using C++ class libraries such as Borland's OWL and Microsoft's Visual C++*

Along with coverage of commercial tools, *Commando Windows Programming* includes two original libraries—TWIN and VWIN—that can simplify many common programming tasks. TWIN simplifies the writing of text-based programs, and VWIN works for any type of program. These libraries can simplify your programs or can provide a starting point for developing your own tools.

Using the techniques and tools in *Commando Windows Programming*, you can write practical Windows programs quickly and easily—often within a few hours of picking up the book. The techniques are shortcuts, but they frequently are the best way to write a practical program. As a bonus, if you later decide to tackle a more conventional Windows programming book, you'll already understand many of the concepts.

Commando Windows Programming offers a quick return on your reading investment. Unlike most other Windows books, you'll start writing practical programs almost right away. Depending on your interests, there are several paths through the book. You can select chapters according to your needs.

Special Features

Because of the broad scope of information in this book, there is a special section at the beginning of each chapter to help guide you. This section describes what is in the chapter and what you need to know to get the most out of it. Also, some chapter sections have the commando paratrooper symbol at the top of the paragraph. The paratrooper identifies sections that will only interest advanced Window commandos. You might want to skip these sections the first time you read through the book. You'll need to read the paratrooper sections only if you want to know why a tool works.

What You Will Need

Most of the programs in this book will work with Borland C/C++ or Microsoft C/C++. Some programs that require proprietary libraries will work only with Borland or only with Microsoft. Chapter 6 covers programs that work exclusively with Microsoft Visual C++. You should also have the Windows SDK documentation either in book form or online.

The Road Map

You probably won't read the chapters of this book in sequence. Instead, you'll probably skip around to satisfy your interests. But everyone should read Chapter 1 first.

If you are an experienced Windows programmer, you might want to skip Chapter 2; otherwise, read it after Chapter 1. After that, you are on your own. The table below will help you find the chapters that interest you most.

IF YOU WANT TO...	GO TO CHAPTER...
Display simple text	3 or 4
Write form-based programs	3
Port existing text-based DOS programs	4
Use C++ to simplify your programs	5
Write programs that incorporate text editors	5 or 6
Use visual programming techniques with C++	6
Use Visual Basic controls in C++ programs	6
Write conventional Windows programs while automatically managing updates, scroll bars, resizing, etc.	6 or 7

1

Windows Myths

WHAT'S IN THIS CHAPTER

You'll find an overview of C programming for Windows and a discussion of why it is so difficult compared to ordinary programming.

PREREQUISITES

None

C programming for Windows has the reputation of being difficult. In addition to your normal C programming skills, you need to know about hundreds of Windows API calls that allocate memory, create windows, and perform a variety of other functions.

Worse still, Windows programs don't look like traditional C programs. Windows programs are event driven (which is largely a good thing). They also require you to cooperate with the system to conserve memory, multitask, and provide user interface operations (which is largely a bad thing).

Commando programmers want a simpler way to write Windows programs. However, they also want to retain the power inherent in Windows.

The key question is: Why is Windows programming difficult? The extra Windows API calls are not that difficult. You don't need to know them all, and the ones you will use are comparable to third-party library calls that do DOS user interfaces and graphics.

Event-driven programming (see Chapter 2) is a little different from normal C programming, but not much. A conventional C program that takes keyboard input and maintains a timer might contain this code fragment:

```
while (1)
  {
  if (kbhit())
    process_key(getch());
  if (timer_flag)
```

```
    time_passed();
  }
```

The same program fragment in a Windows program might look like this:

```
event(HWND w,unsigned *m,WORD wParam,LONG lParam)
  {
  switch (m)
    {
    case WM_CHAR:
      process_key(wParam);
      break;
    case WM_TIMER:
      time_passed();
      break;
     }
   }
```

Not much of a difference. The Windows kernel takes the place of the original while loop and passes many events (not just keyboard and timer events) to a function of your choice.

For most Windows programs, this structure really is an advantage. If you have written a Windows program before, you may disagree. Actually, the bad part about event-driven programming under Windows is not the basic idea but some of the specific messages. Be careful not to confuse the two.

In reality, the culprits that give event-driven programming a bad name are some of the specific events. For example, suppose you want to write text to the screen. You can't just write it to the window and expect it to stay there. At any time, Windows may decide to ask you to redraw a part of it. For simple text, this isn't much of a problem, but for complex graphics or text documents, it may be difficult to do. Other things that Windows should take care of for you (scroll bars, for example) will barrage your program with difficult events.

A traditional Windows program has four parts: the initialization section, an application model, an update routine, and program logic. The initialization section starts up the program, of course. The model is a representation of program data (for example, a word processing document). The update routine decides how to display a portion of the model on the screen. Finally, the program logic takes input (usually from the user) and uses it to modify the model.

Howdy World

A good example of Windows programming complexity is the famous "Howdy World" program (if you aren't from Texas, you may know this as the "Hello World" program). In case you haven't seen it, Listing 1-1 has the famous program written for Windows. Wow! It's almost 200 lines of code—and that doesn't include its resource and DEF file (see Chapter 2). Even with great formatting and comments, a DOS HOWDY.C is only ten lines of code.

Here the init() function handles initialization, the update() routine displays the model, and the model is the character array named model. The HOWDY program doesn't allow its model to change, so it has almost no program logic. However, for the sake of illustration, the menu() function causes the program to terminate—the only program logic step HOWDY uses.

Some Windows programmers will tell you that this isn't a fair comparison. The Windows program has more to do. However, commandos know that this isn't really a far-fetched comparison. For example, how many times do you need to write a program that displays some data from a database? Or print some status message to the screen. Can you really afford 200 lines of code to do that?

Commando Programmers Have Shortcuts

Luckily, commando programmers have shortcuts that make writing Windows programs easier. Listing 1-2 shows a short Windows HOWDY.C that uses the techniques described in Chapter 3. That's better, isn't it? Listing 1-3 shows a program written with Borland's EZWIN product, which will also work with Microsoft's QuickWin. It looks just like a DOS program (of course, it also acts just like a DOS program). You'll learn more about these techniques in later chapters. For now, just realize that traditional Windows programming techniques are not always the best way to write a Windows program.

Why Is Windows This Way?

When Windows first appeared, it operated (barely) on a conventional 8088 PC with less than 640K of memory. Since even a moderate-sized screen image could take 64K of memory (or more), it wasn't very practical for Windows to store screen images for later recall. Without this ability, Windows cannot manage screen redisplay, scroll bars, and other important functions. You must deal with these yourself. Most of the commando techniques in this book hide this complexity from you.

When Using Windows Is Better

If there are easier ways to write Windows programs, why does the traditional method still persist? Although the commando techniques are useful for many programs in many situations, they are not right for every program. Some commando techniques, for instance, consume large amounts of memory, which may not be suitable for your application.

Other programs naturally fit the Windows model. For example, a word processor builds a document; this document corresponds exactly to the model portion of a Windows program. Directly placing text and formatting on the screen is of little value when you must update the model anyway.

Still, many programs can benefit from simplified Windows techniques. Even a word processor will have portions of code that don't fit well with the traditional programming model. Programs that work with databases

are often good candidates for commando techniques. Simple utilities that create files, set up printers or network connections, or perform similar tasks can often benefit from the commando approach.

Where to Next?

Although the commando techniques simplify Windows programming, most of them still require some Windows knowledge. Unless you are already an experienced Windows programmer, you should read Chapter 2 next. It will quickly teach you some basic Windows ideas and terms. Once you are familiar with these concepts, you can find chapters of interest in the roadmap at the end of the Introduction.

Listing 1-1. HOWDY.C

```
/*********************************************************
 *                                                       *
 * File: HOWDY.C                                         *
 *                                                       *
 * Typical (noncommando) Windows program                 *
 *                                                       *
 * Required to Compile:                                  *
 * HOWDY.C HOWDY.H HOWDY.RC HOWDY.DEF                    *
 *                                                       *
 *********************************************************/
#include <windows.h>
#include <string.h>
#include "howdy.h"


/* current instance */
HANDLE hInst;
/* main window */
HWND topwindow;

/* String to display -- the "model" */
char *model = "Howdy World!";


/* Main window function */
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                   LPSTR cmdline, int show)
  {
  MSG msg;
```

```
  if (!init(hInst, prev, show))
    return FALSE;

  /* Vanilla event loop */
  while (GetMessage(&msg, NULL, NULL, NULL))
      {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
      }
  /* Exit program */
  return (msg.wParam);
  }

/* Start up stuff */
int init(HANDLE hInst, HANDLE prev, int show)
  {
  if (!prev)
      {
      if (!init_app(hInst))
   /* Exit if unable to initialize */
        return FALSE;
      }
  /* Perform instance init */
  if (!init_inst(hInst, show))
    return FALSE;
  return TRUE;
  }

/* Create window class here */
```

```
BOOL init_app(HANDLE hInstance)
  {
  WNDCLASS wc;
  wc.style = NULL;
  wc.lpfnWndProc = (void FAR *) win_proc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hInstance = hInstance;
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName = "HOWDYMENU";
  wc.lpszClassName = "HOWDY_Class";
  return (RegisterClass(&wc));
  }


/* Create window here */
BOOL init_inst(HANDLE hInstance, int nCmdShow)
  {
  HWND hWnd;
  /* Save the instance handle in global variable */
  hInst = hInstance;

  /* Create a main window */
  topwindow = hWnd = CreateWindow(
               "HOWDY_Class",
               "Howdy, Howdy, Howdy!",
               WS_OVERLAPPEDWINDOW,
               CW_USEDEFAULT,
```

```
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                NULL,
                NULL,
                hInstance,
                NULL
      );

  if (!hWnd)
    return FALSE;

  /* Make the window visible, update its client area, and
   * return "success" */
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  return (TRUE);
  }


/* Window procedure */
long WINAPI _export win_proc(HWND hWnd, UINT message,
                  UINT wParam, LONG lParam)
  {
  switch (message)
      {
    case WM_COMMAND:
      menu(hWnd, wParam);
      break;
```

```
    case WM_DESTROY:
      PostQuitMessage(0);
      break;

    case WM_PAINT:
      update(hWnd);
      break;

    default:
      return (DefWindowProc(hWnd, message,
              wParam, lParam));
      }
  return NULL;
  }

/* Update screen in response to WM_PAINT messages */
void update(HWND w)
  {
  HDC dc;
  PAINTSTRUCT paint;
  dc = BeginPaint(w, &paint);
  TextOut(dc, GetDeviceCaps(dc, LOGPIXELSX) / 2,
     GetDeviceCaps(dc, LOGPIXELSY) / 2,
     model, strlen(model));
  EndPaint(w, &paint);
  }


void menu(HWND hWnd, UINT wParam)
  {
```

```
  /* pointer for "About" function */
  FARPROC aboutproc;
  if (wParam == IDM_ABOUT)
      {
      aboutproc = MakeProcInstance(about, hInst);
      DialogBox(hInst, "AboutBox", hWnd, aboutproc);
      FreeProcInstance(aboutproc);
      return;
      }
  else if (wParam == IDM_STOP)
      {
      DestroyWindow(hWnd);
      return;
      }
  }

/* Ordinary about box */
BOOL WINAPI _export about(HWND hDlg, UINT message,
                    UINT wParam, LONG lParam)
  {
  switch (message)
      {
    case WM_INITDIALOG:
      return (TRUE);

    case WM_COMMAND:
      /* Use LOWORD for Win32 compatibility */
      if (LOWORD(wParam) == IDOK
         || LOWORD(wParam) == IDCANCEL)
         {
```

```
            EndDialog(hDlg, TRUE);
            return (TRUE);
            }
        break;
        }
    return FALSE;
    }
```

Listing 1-2. CHOWDY.C

```
/*********************************************************
 *                                                       *
 * File: CHOWDY.C                                        *
 *                                                       *
 * Commando version of HOWDY                             *
 *                                                       *
 * Required to Compile:                                  *
 * CHOWDY.C CHOWDY.DEF                                   *
 *                                                       *
 *********************************************************/
#include <windows.h>

/* Main window function */
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                   LPSTR cmdline, int show)
    {
    MessageBox(NULL, "Howdy World!", "Howdy!",
               MB_OK);
    return FALSE;
    }
```

Listing 1-3. EZHOWDY.C

```
/*********************************************************
 *                                                       *
 * File: EZHOWDY.C                                       *
 *                                                       *
 * Simple Howdy world program for EZWIN or QuickWin.     *
 *                                                       *
 * Required to Compile:                                  *
 * ezhowdy.c                                             *
 *                                                       *
 *********************************************************/

#include <stdio.h>

main()
  {
  printf("Howdy World\n");
  }
```

2

Windows Basics

WHAT'S IN THIS CHAPTER

Chapter 2 covers basic Windows programming concepts (for example, event loops, windows, menus, and resources). You will need a working understanding of these concepts to apply most of the commando techniques that appear in the remainder of the book. If you already know how to write conventional Windows programs, you may wish to skip this chapter.

PREREQUISITES

To get the most from this chapter, you'll need a knowledge of C programming.

Windows programs look different than ordinary C programs. Even the simplest Windows program usually has several files and many functions. This chapter will help you get oriented in the Windows world and introduce some important new terms.

Windows programs are different from conventional programs in four major ways:

- *The structure of the program is different.*
- *Windows programs are event driven.*
- *The compile-and-link cycle differs.*
- *Windows provides an enormous number of new API calls and messages.*

This chapter will focus mainly on the first three differences. You can pick up the new API calls as you go along. Be sure that you have an API reference (either the book or online help from the Microsoft or Borland compilers) to answer your API questions.

The HOWDY program in Chapter 1 (see Listing 1-1) is a simple Windows program that we will dissect in this chapter. Windows programs are usually more complex, but the principles are the same as in HOWDY. Listing 1-1 contains the C source code for HOWDY, but that isn't all it takes to build a Windows program. Listings 2-1, 2-2, and 2-3 show HOWDY.H, HOWDY.RC, and HOWDY.DEF. You'll need these three files to actually compile HOWDY.

Note that HOWDY is not a commando program—it is a typical Windows application. If it takes a program this large to write one string to a window, how big will your program be? Luckily, Chapter 3 will show you better ways to write this type of program. In fact, the examples in Chapter 3 will do more with less code. Still, you should wade through HOWDY to see how the other half lives. It will greatly enhance your appreciation of the programs that follow.

Anatomy of a Windows Program

Conventional C programs begin execution at their main() function. Windows programs begin at WinMain(). Like main(), WinMain() is responsible for starting up your program, and when it exits, so does your application.

Here is the prototype for WinMain():

```
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                   LPSTR cmdline, int show);
```

The PASCAL keyword signals the compiler to use a Pascal-style calling convention for efficiency reasons. Most functions that Windows supplies or calls use this calling convention.

The arguments to WinMain() are straightforward. Windows assigns each running program an *instance handle*. This handle uniquely identifies the program in the same way the PSP address identifies a DOS program. The hInst parameter is your program's instance handle. Windows

will often require you to pass this back to it as a parameter to other API calls.

The second HANDLE argument, prev, is NULL if your program is running for the first time or is a Windows NT program. If there is another copy of your program running under Windows 3.x, Windows places its instance handle in the prev parameter. By examining it, you could prevent multiple copies of your program from executing. You also can skip some initialization steps when you know that another copy of your program has already done them (see below). If you want to, you can even write a program that simply issues requests to the older copy of itself and terminate.

The cmdline parameter is a far pointer to your command line. The command line is unparsed and null terminated. This is not as handy as the argc and argv parameters you usually get, so the Borland compilers provide global variables _argc and _argv to take their place. While Microsoft doesn't supply these variables, Listings 2-4 and 2-5 (ARGCARGV.C and ARGCARGV.H) allow you to use them with either compiler.

ARGCARGV.C and ARGCARGV.H (Listings 2-4 and 2-5) supply a simple command line parser for Microsoft programs. Although it isn't as sophisticated as the standard parser, it is more than adequate for most Windows programs. You can include ARGCARGV.H in your Borland or Microsoft programs. You also need to call set_args() before using the _argc or _argv variables. For Borland programs, this call does nothing, and the header just includes the proper header that defines _argc and _argv.

The MOTD program in Chapter 3 (Listing 3-1) shows how to use ARGCARGV.

If you use Borland, you can either compile and link with ARGCARGV.C or not—it expands to nothing under Borland. For Microsoft programs, you must compile and link ARGCARGV.C. The version in Listing 2-4 limits you to a maximum of 20 arguments. It also doesn't understand argument quoting. For example, the arguments:

```
"Hex Mode" On
```

would normally result in the following assignments:

```
argv[1]="Hex Mode";
argv[2]="On";
```

With ARGCARGV.C, the results are:

```
argv[1]="\"Hex";
argv[2]="Mode\"";
argv[3]="On";
```

The final parameter to WinMain() is show. This variable indicates whether the program should start as a normal window, an icon, or a full-screen window. Usually, you just pass this parameter back to Windows in the WinShow() call and forget about it.

WinMain() usually (but not always) has three main functions:

- *Do global initialization (if this is the first copy of your program).*
- *Do instance initialization.*
- *Start the event loop.*

If the prev parameter in WinMain() is NULL, you have to do some global initialization that you would skip if it was not NULL. You'll see more about that soon. Since Windows NT programs are isolated from each other, this parameter is always null under NT.

Nearly all programs will need to perform some private initialization in WinMain(). Most often, this is the creation of the application's main window.

HOWDY.C uses the init() function to perform both types of initialization. The init_app() function does the global setup (if required), and the init_inst() routine handles the window creation.

Finally, WinMain() enters an event loop. This captures messages sent by Windows and routes them to the correct functions in your program. We will look at messages and events shortly.

The event loop is written so that it ends when you terminate your program (say, by clicking close from the system menu). Since the loop is the last thing in WinMain(), ending it causes WinMain() and your program to end.

Classy Windows

A program's windows serve as focal points for its activity. Windows receive events, own menus, and, of course, display data. Each window you create must belong to a window *class*. A window's class defines its default behavior. Windows provides some built-in classes (for example, button, scroll bar, and so on), but usually you create your own classes.

The init_app() routine in HOWDY.C creates a window class named HOWDY_Class. All windows of this class will have the same menu (HOWDYMENU), the same icon (IDI_APPLICATION), and the same cursor (IDC_ARROW). The WNDCLASS structure holds the information about the class, and the RegisterClass() call creates it.

Creating a Window

Creating a class does not create a window; that is the job of init_inst(). There, the CreateWindow() call makes an actual window of class HOWDY_Class.

The CreateWindow() call takes a number of parameters. In HOWDY.C, the CW_USEDEFAULT constant requests the default position and size for the window. Other parameters specify the window's title, the window's menu (if different from the default class menu), and the window's style (for example, whether it has scroll bars or a system menu).

Since the window that HOWDY creates is its main window, it has no parent. Often windows will be children of another window and therefore have a non-null value for the parent parameter in CreateWindow(). Child windows

fit inside their parent window and are only visible when the parent is visible.

When you create a window, you may specify that it have a certain style. The WS_SYSMENU style, for example, causes the window to have a system menu box in the top-left corner. For convenience, Windows provides some common styles that consist of several styles merged together. For instance, the WS_OVERLAPPEDWINDOW style is equivalent to:

```
WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME |
WS_MINIZEBOX | WS_MAXIMIZEBOX
```

Events and Messages

Each window class specifies its *window procedure* when it is created. HOWDY_Class windows, for example, use win_proc() as their window procedure. Windows sends all events that pertain to HOWDY to win_proc().

If you look at win_proc() in Listing 1-1, you will see that it is just a big switch statement. The message parameter contains an integer that defines which event happened. For example, the WM_COMMAND message occurs when the user clicks on a menu item (or uses a keyboard accelerator). The WM_DESTROY message occurs when the window is closing. The WM_PAINT message indicates that Windows wants you to redraw all or part of the window. You can find a complete list of messages and their parameters in your API documentation.

You can intercept and process any windows messages you want. If you don't handle a message, you should

pass it on to DefWindowProc(), which is part of Windows. This default window procedure accounts for much of the standard window behavior.

Some messages originate from Windows. These messages occur when interesting events occur (for example, when the mouse moves, or the user selects a menu choice). Your program can also send messages to windows to cause certain actions.

Some Windows operations take the form of conventional function calls. For example, if you want a particular window to move to the front and respond to the keyboard, you make the call:

```
SetFocus(w);  /* w=window's handle */
```

Other operations take the form of messages. You can send a message to a window and optionally wait for it to respond to you. There are two primary methods of sending a message: SendMessage() and PostMessage(). SendMessage() waits until the message completes and returns a value. PostMessage() puts a message in the window's queue but doesn't wait for it.

There are many Windows messages, and some of them mean slightly different things to different windows. For example, the WM_SETTEXT message sets the title of a normal window. However, for text-edit windows (controls), the WM_SETTEXT message determines the text inside the window.

All messages take two parameters (by convention, wParam and lParam). These arguments specify data for the message. For example, to use WM_SETTEXT, you must cast a string pointer to a long and pass it as lParam like this:

```
SendMessage(w,WM_SETTEXT,0,(long)"Title!");
```

WM_SETTEXT ignores the wParam argument.

Under Windows NT, wParam is 32-bits (the same as lParam). NT uses the extra 16-bits for additional information. Programs that can compile for Windows 3.1 or Windows NT often use lines like this:

```
switch (LOWORD(wParam))
  {
  }
```

This accomplishes nothing under Windows 3.1 and is harmless. If the Windows NT version of a message doesn't change the meaning of the lower 16-bits of wParam, the code will work on either platform.

License to Export

Functions that Windows will call (except for WinMain()) must be far and exportable. You will notice that the MainWndProc() function, for example, uses the FAR and _export keywords. These keywords allow Windows to call the functions properly, even if more than one copy of

your program is running. Functions that you only call yourself—for example, init() or menu()—don't require the _export keyword and usually are not far functions.

When dealing with function addresses, you have to bind the address to your instance for Windows to call it correctly. (The exception is for calls to RegisterClass(); that function binds the window procedure for you.) For example, the menu() function uses this code:

```
aboutproc=MakeProcInstance(about,hInst);
```

This code allows Windows to correctly call the about() function via the pointer aboutproc. If you simply pass Windows the address of about(), your function will not be able to properly access its variables. When you are done with a bound function address, you should free it. HOWDY uses:

```
FreeProcInstance(aboutproc);
```

Drawing to the Screen

Most commando techniques don't require you to draw directly to a window. However, you should know how it's done. All Windows drawing functions (both graphics and text) use a *device context*. This is a magic number that references a particular window and set of drawing tools.

You should usually draw to a window only during WM_PAINT message processing. For example, the update() routine in HOWDY uses the BeginPaint() func-

tion to get a device context, and the TextOut() function to write text to it.

You can make calls to alter how a device context works (for example, to change the color of drawings). You can also query the context for information. HOWDY wants to draw text one-half inch away from the top-left corner. Therefore, it uses the call:

```
GetDeviceCaps(dc,LOGPIXELSX)
```

to find the number of pixels per inch.

If you draw to a window outside of the WM_PAINT context, your drawing will be transient. When Windows asks you to redraw that portion of your window, your WM_PAINT routine won't be able to re-create it. On rare occasions (for example, drawing a selection), this may be what you want to do. Most often, however, you will draw only during WM_PAINT messages.

Memory Allocation

Although HOWDY doesn't allocate any memory, you should still have a general idea about Windows memory management. Windows manages two separate heaps, or areas, for memory allocation.

The *local heap* is in your program's default data segment. Because a segment must be 64K or less, the local heap can never be larger than 64K and is almost always less (except under Windows NT).

The *global heap* can access the large pool of memory outside your program's data segment. You can allocate items nearly 16M in size using Window's GlobalAlloc() call.

Under Windows 3.0 real mode, you had to be careful with global memory. GlobalAlloc() returns a handle that you have to convert to a far address using GlobalLock(). In real mode, locking memory hinders Windows, so you had to keep locking and unlocking memory. In protected mode, Windows doesn't care if you keep memory locked or not. So if you need a one million byte buffer, you could say:

```
char far *p;
HANDLE p_handle;
p_handle=GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT,1000000);
if (!p_handle) error();
p=GlobalLock(p_handle);
if (!p) error();
/* use p */
GlobalUnlock(p_handle);
GlobalFree(p_handle);
```

Note that you can call GlobalAlloc() from a small model program—just be sure to use a far pointer (or the Windows LPSTR type). You'll also need to use far-pointer versions of any library functions you want to use. For example, instead of strcpy(), you would use _fstrcpy(), since it will accommodate a far pointer.

Avoid making large allocations using malloc(), calloc(), and related calls in Windows 3.1 programs. By default,

these allocate from your local heap and are not as useful as GlobalAlloc(). They run out of space much quicker. Of course, WIN32S and Windows NT programs don't have this limitation.

By the same token, in Windows 3.1 programs, avoid using GlobalAlloc() for small allocations. There is a systemwide limit on the number of memory regions that GlobalAlloc() can return (around 8,000). If you use GlobalAlloc() for many small memory allocations, you can quickly cause the entire system to run out of memory.

Resources

Listing 2-2 is a resource file. Resources are data that you can store inside your EXE file. These data can specify menus, bitmaps, cursors, icons, or user-defined data. Resources can also specify special-purpose windows called dialogs. Dialogs are very important to the commando programmer (see Chapter 3) because they will do most of the dirty work for many Windows programs.

You can create resources in an ASCII text file (like Listing 2-2), or you can use a specialized resource editor (like Borland's Resource Workshop or Microsoft's Application Studio) to draw menus, bitmaps, and dialogs interactively.

Resources, especially dialog boxes, are key to many commando strategies. Unlike regular windows, dialog boxes don't require much work to use. They paint themselves and only bother you when something interesting happens (like when you press a button).

When you need text-only input and output, you should automatically think about dialog boxes. You'll see how to write some powerful dialog-only programs in Chapter 3.

Dialogs come in two flavors: modal and modeless. When a modal dialog box is visible, you can't access other windows in your program. A modeless dialog box (which requires a special event loop) is more like an ordinary window. While it is present, you can still switch to another window in your program. Later, you can switch back to the dialog box.

Menus are also simple to use. You actually can write some useful Windows programs that contain nothing but a menu (if you don't believe it, look at Listing 3-3).

Special Libraries

Don't overlook the many special-purpose libraries that ship with Windows. For example, how difficult would it be to write an audio CD player? Without the Windows Multimedia Control Interface (MCI) libraries, it would be very difficult. With MCI, it becomes almost trivial. (We will write this program soon.)

Windows comes with APIs to manage multimedia devices, display online help, launch other programs, and do a host of otherwise difficult tasks. Be sure to start each project with a search for potentially useful Windows calls.

Compiling and Linking

Windows programs require a special compile-and-link process since they have a more complicated structure than the ordinary EXE format. Figure 2-1 shows the process. As usual, the compiler converts C and H files to OBJ files. However, the linker must make a different type of EXE file. Since there is more information in the EXE file, the linker now reads a DEF file (like HOWDY.DEF in Listing 2-3) to gain additional information about the program.

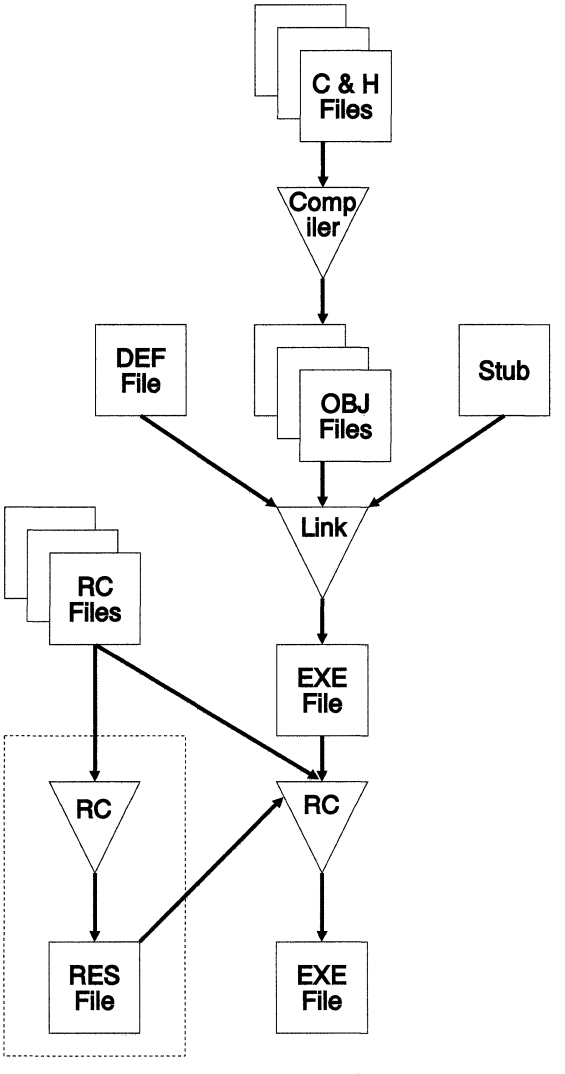
Although the linker creates a Windows EXE file, the file has no resources in it. If you run it, it will work, but it will have no menus, icons, and so on. To get your resources in the EXE file, you need a resource compiler like Microsoft's RC program. You can use this program in two ways. The easiest way is to directly bind the RC file into the EXE file. For example:

```
rc HOWDY.RC HOWDY.EXE
```

However, if your resources don't change often, this procedure wastes time because it compiles the resources each time. You may want to compile the RC file into a binary RES file. You do this with:

```
rc -r HOWDY.RC HOWDY.RES
```

Figure 2-1. Windows Compile/Link Cycle



Then, you can quickly bind the RES file to an EXE file like this:

```
rc HOWDY.RES HOWDY.EXE
```

If you forget to bind your resources, you'll figure it out soon enough. When you run your program, you'll see a window with no menu, and you won't be able to see any dialogs. Just double-click the system box (in the top-left corner) to end the program and then run RC.

Listings 2-6, 2-7, and 2-8 are three batch files that will compile and link HOWDY using Borland C/C++, Microsoft C/C++, or the Microsoft Win32 tools, repectively. The basic steps are the same as the ones used to compile and link any Windows program. The NT batch file also requires the NTLINK.RSP file (Listing 2-9) to supply commands to the linker.

You should almost always use small model for Windows programs. Small-model programs can still access as much data as they need by using far pointers and GlobalAlloc(). If you have an extraordinary amount of code, you could use medium model. However, Windows has difficulty loading large and compact model programs—using these models may prevent you from running more than one copy of your application.

Learning More

This chapter has only scratched the surface of Windows programming. If you want to learn more about Windows

programming, you might check out some of the books in the bibliography. However, for most of the commando techniques in this book, you now know all you need to know about Windows. See the sidebar *The Commando Commandments* on the following page for some general advice about commando techniques. You'll hear more about the techniques in later chapters.

At this point, you may or may not want to proceed to Chapter 3. Look at the roadmap in the Introduction to this book to determine which chapter best suits your interest.

The Commando Commandments

I. Use dialogs instead of windows if possible.

II. Use the Windows API effectively.

III. Avoid creating resources by hand; instead, use a graphical tool.

IV. For text-based programs, use dialogs or text-emulation tools.

V. Learn C++ and use class libraries to simplify your programs.

VI. Avoid writing WM_PAINT routines.

VII. Program visually when possible.

VIII. Select the right commando technique for the job.

IX. Resort to traditional techniques only when necessary.

X. Even when using traditional techniques, try to find parts of your program that could benefit from commando programming methods.

Listing 2-1. HOWDY.H

```
/*********************************************************
 *                                                       *
 * File: HOWDY.H                                         *
 *                                                       *
 * Header for HOWDY.C                                    *
 *                                                       *
 * Required to Compile:                                  *
 * HOWDY.C HOWDY.H HOWDY.RC HOWDY.DEF                    *
 *                                                       *
 *********************************************************/
/* Menu defines */
#define IDM_ABOUT 100
#define IDM_STOP 101

/* prototypes */
BOOL init_app(HANDLE);
BOOL init_inst(HANDLE, int);
long WINAPI _export win_proc(HWND, UINT, UINT, LONG);
BOOL WINAPI _export about(HWND, UINT, UINT, LONG);
void menu(HWND, UINT);
void update(HWND);
int init(HANDLE, HANDLE, int);
```

Listing 2-2. HOWDY.RC

```
/*********************************************************
 *                                                       *
 * File: HOWDY.RC                                        *
 *                                                       *
```

```
 * Resources (i.e., menus and dialogs) for HOWDY.C    *
 *                                                    *
 * Required to Compile:                               *
 * HOWDY.C HOWDY.H HOWDY.RC HOWDY.DEF                 *
 *                                                    *
 ******************************************************/

#include "windows.h"
#include "howdy.h"

/* Main menu */
HOWDYMENU MENU
BEGIN
    POPUP         "&File"
    BEGIN
        MENUITEM "&About Howdy...",IDM_ABOUT
        MENUITEM "&Exit",IDM_STOP
    END
END

/* About dialog box */
AboutBox DIALOG 22,17,144,75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About Howdy"
BEGIN
    CTEXT "Howdy"      -1,0,5,144,8
    CTEXT "By Al Williams" -1,0,14,144,8
    CTEXT "Version 1.0" 1,0,34,144,8
```

```
    DEFPUSHBUTTON "OK"  IDOK,53,59,32,14,WS_GROUP
END
```

Listing 2-3. HOWDY.DEF

```
Name HOWDY
Description 'Hello World, Texas Style'
Exetype WINDOWS
Code PRELOAD MOVEABLE DISCARDABLE
Data PRELOAD MOVEABLE SINGLE
Heapsize 4096
Stacksize 5120
Stub 'WINSTUB.EXE'
```

Listing 2-4. ARGCARGV.C

```
/*****************************************************
 *                                                   *
 * File: ARGCARGV.C                                  *
 *                                                   *
 * Module to allow Microsoft users to use _argc      *
 * and _argv.                                        *
 * Limitations:                                      *
 *  1) Only 20 arguments are supported.              *
 *  2) No quotes are processed.                      *
 *                                                   *
 * Required to Compile:                              *
 * Many programs use ARGCARGV.C and ARGCARGV.H       *
 * They do not compile separately.                   *
 *                                                   *
 *****************************************************/
```

```
/* Only required for Microsoft */
#ifndef __BORLANDC__
#include <windows.h>
#include <string.h>
#include "argcargv.h"

/* local buffer for command line */
static char local_cmd[129];
static char my_file_name[129];

int _argc;
char *_argv[MAXARG];

void set_args(LPSTR cmd, HANDLE inst)
  {
  char *lbuf = local_cmd;
  /* copy to local buffer */
  while (*lbuf++ = *cmd++);
  /* set up _argv[0] */
  GetModuleFileName(inst, my_file_name,
          sizeof(my_file_name));
  _argv[0] = my_file_name;
  _argc = 1;
  lbuf = local_cmd;
  while (_argv[_argc] = strtok(lbuf, " \t\n\r"))
      {
      lbuf = NULL;       /* reset for next token */
      _argc++;        /* note valid argument */
      if (_argc=MAXARG) break;
```

```
      }
  }



#endif
```

Listing 2-5. ARGCARGV.H

```
/*******************************************************
 *                                                     *
 * File: ARGCARGV.H                                    *
 *                                                     *
 * Header to allow Windows programs to use _argc       *
 * and _argv. Microsoft users also need ARGCARGV.C     *
 *                                                     *
 * Required to Compile:                                *
 * Many programs use ARGCARGV.C and ARGCARGV.H         *
 * They do not compile separately.                     *
 *                                                     *
 *******************************************************/

#ifndef _ARGCARGV_H
#define _ARGCARGV_H

#ifdef __BORLANDC__
#include <dos.h>
#define set_args(a,b)
#else
```

```
/* Limit to 20 arguments max */
#define MAXARG 20

extern int _argc;
extern char *_argv[MAXARG];
void set_args(LPSTR cmd, HANDLE inst);
#endif

#endif
```

Listing 2-6. BORCOMP.BAT

```
REM Batch file to compile HOWDY with Borland C
bcc -v -W howdy.c
rc howdy.rc howdy.exe
```

Listing 2-7. MICCOMP.BAT

```
REM Batch file to compile HOWDY with Microsoft C
cl -Zi -GA howdy.c howdy.def
rc howdy.rc howdy.exe
```

Listing 2-8. NTCOMP.BAT

```
REM Batch file to compile HOWDY under Windows NT
REM (also needs NTLINK.RSP)
cl386 -c -G3 -W3 -Di386=1 -DWIN32 -Zi -Od -DNT -DWIN howdy.c
rc -r howdy.rc
cvtres -i386 howdy.res -o howdy.rbj
link -out:howdy.exe howdy.obj  howdy.rbj @NTLINK.RSP
```

Listing 2-9. NTLINK.RSP

```
-debug:full
-debugtype:both
-subsystem:windows
-entry:WinMainCRTStartup
libc.lib
ntdll.lib
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
```

3

Unlimited Resources

WHAT'S IN THIS CHAPTER

In Chapter 3, you'll learn how to write simple text-oriented programs using menu and dialog resources. You'll see how to write a message utility, an audio CD player, a simple conversion calculator, and a powerful phonebook program without drawing to a single window.

PREREQUISITES

You'll need to know C programming and basic Windows concepts as well as have a knowledge of resources.

Resources form an integral part of nearly all Windows applications. Resources allows a program to store data inside its EXE file. These data specify items such as menus, icons, and dialog boxes. The key to many commando programs is to avoid using regular Windows—only use resources (particularly dialogs and menus).

While windows are complicated to create and maintain, dialogs and menus require almost no maintenance. Your program can concentrate on the task it's trying to perform and avoid user interface issues.

How to Write Howdy

The HOWDY program (Listing 2-1) is terrible. It is not a commando program. Commando Commandment II ("Use the Windows API effectively") dictates that you should use the MessageBox() function to display the "Howdy World" string (see Listing 1-2).

Listings 3-1 and 3-2 contain a program inspired by Unix's MOTD file. It reads a line from the file of your choice and displays it using MessageBox(). This program is useful to run automatically when Windows starts on a network workstation. You can display short messages to users (for example, "Network will be down at noon," "New compiler version in \newgizmo\compile"). Figure 3-1 shows the calling details for MessageBox().

This is a true commando program—short and sweet. If you want to display only a fixed string, the program could be even simpler; the bulk of the code reads the message from the file.

Figure 3-1. Calling MessageBox()

```
int MessageBox(HWND parent, LPSTR text, LPSTR title, WORD
flags);
```

parent - Handle to parent window. May be NULL.

text - String to display in box.

title - Caption for box. If NULL, Windows uses "Error".

flags - Any combination of the following values joined by the or operator (|):

MB_ABORTRETRYIGNORE - Display three buttons: abort, retry, and ignore.

MB_OK - Display OK button.

MB_OKCANCEL - Display OK and cancel button.

MB_RETRYCANCEL - Display two buttons: retry and cancel.

MB_YESNO - Display yes and no buttons.

MB_YESNOCANCEL - Display three buttons: yes, no, and cancel.

MB_APPLMODAL - Make box modal (default).

MB_SYSTEMMODAL - Suspend all applications until box is dismissed.

MB_TASKMODAL - Suspend current task (useful for acting like MB_APPLMODAL when parent is NULL).

MB_DEFBUTTON1 - First button is the default button (default).

MB_DEFBUTTON2 - Second button is the default.

MB_DEFBUTTON3 - Third button is the default.

MB_ICONASTERISK - Place info icon in box.

MB_ICONINFORMATION - Place info icon in box.

MB_ICONEXCLAMATION - Place exclamation icon in box.

MB_ICONHAND - Place stop icon in box.

(Cont.)

Figure 3-1. Calling MessageBox() (Cont.)

MB_ICONSTOP - Place stop icon in box.

MB_ICONQUESTION - Place question mark icon in box.

Return value:

Returns zero if there was an error, otherwise the value indicates which button the user picked: IDABORT, IDCANCEL, IDIGNORE, IDNO, IDOK, IDRETRY, or IDYES.

A Simple CD Player

Although menus are easy to create, they can form the basis for some useful Windows programs. Listings 3-3 to 3-6 (CDPLAYER) show a simple audio CD player that only uses a menu. You'll need a CDROM drive and the Windows MCICDA driver to use this program. Of course, your platform must support the multimedia extensions (WIN32s currently does not). Figure 3-2 shows CDPLAYER's window.

As you can see, CDPLAYER is just a slight variation on the HOWDY program in Chapter 2. It has no model string, no update routine, and a bigger menu() function. In addition, its window is very small, making it more attractive. CDPLAYER also replaces the complex about box dialog code with a call to MessageBox(). (Remember commando Commandment II.)

Figure 3-2. The CDPLAYER Application

As only one CD player should run at once, CDPLAYER checks the prev field in WinMain(). It must be NULL or CDPLAYER will refuse to run.

CDPLAYER uses Window's MCI API. This API reduces device control to the incredibly obvious. For example, to play the CD, you use the following MCI call:

```
mciSendString("play cdaudio",NULL,0,NULL);
```

The second parameter is a string buffer for a return value (if any), and the third argument is the length of the buffer. The final parameter is only useful if you are trying to do other things while the MCI commands execute. Look in your API reference for more about the MCI API.

Unlike most Windows programs, many of CDPLAYER's menu items don't have a submenu. Although this is unconventional, it closely models how a real CD player works. Part of the power of Windows is this type of flexibility. Just because most programs do something one way, don't be afraid to experiment with other techniques.

You probably can find many uses for menu-only programs. A program that sends commands over a network, for example, might need only a menu. Coupling a menu with WinExec() lets you quickly write custom launch menus that start up other programs. Other possibilities are programs that print forms or data to a printer, backup or erase files, or make fixed entries into databases.

Back to Basics

During Windows programming, you'll often find yourself wishing for normal C I/O like printf() and gets(). Maybe you'd like to put some printf's in your code for debugging. Perhaps you need a single string entered for some reason. Either way, you'll miss these functions eventually.

MessageBox() can replace printf() in many cases, but it is awkward if you need to print variables. For example:

```
void error(int errno,char *errmsg)
  {
  char tbuf[1025];
  sprintf(tbuf,"Error %d: %s",errno,errmsg);
  MessageBox(NULL,tbuf,NULL,MB_OK|MB_ICONSTOP);
  return;
  }
```

Notice that if you pass a NULL title to MessageBox(), it uses the default title, "Error". Also, you can pass a NULL as the window handle if you wish.

Unfortunately, there isn't a simple function to get a line of input. Luckily, however, it is simple to overcome both this oversight and the awkwardness inherent in MessageBox(). Listings 3-7, 3-8, and 3-9 contain the WPRINT library. WPRINT provides you with two simple functions: win_printf() and win_input().

Both of these functions take a variable number of arguments. The first argument is a title for the input or output

window that these functions create. The second argument is a printf-style format string that will display inside the window. Of course, other arguments depend on the contents of the format string. For example:

```
win_printf("Presto","The magic number is: %d",123);
s=win_input("Press Enter","Enter your id number(%d)",1);
```

The win_input() function returns a pointer to a static buffer that contains the user's input string. The pointer will never be NULL, but, if the user did not enter anything, the string may be (that is, the first byte may be zero). Both functions use static buffers for input and output strings, so they can't exceed 512 bytes (unless you change the print_buf and in_buf definitions, of course).

The win_input() function uses a custom dialog that you must include in your program's RC file. Just place the line:

```
#include "wprint.rc"
```

in your RC file to get the required definition. WPRINT also expects you to define a global variable, hInst, that contains your instance handle.

Listings 3-10, 3-11, and 3-12 show a simple example of the power of WPRINT. CTOF is a basic centigrade to Fahrenheit temperature converter. This program contains no windows and no resources (except for WPRINT's dialog box). A similar DOS text program could hardly be

much simpler and probably would be more complex if it allowed text editing, and windows. Windows gives us all of that for free.

WPRINT allows you to provide more sophisticated functionality in your programs. You should be careful when using WPRINT inside programs that do not have a normal window (like CTOF) under Windows 3.1. CTOF (and other nonstandard programs) subverts the normal Windows cooperative multitasking mechanism. Therefore, other programs will not execute unless there is a call to win_printf() or win_input(). These functions notify Windows that it may switch to another task. CTOF does very little processing between WPRINT calls, but if it did, you would want to call yield() frequently to give other programs a chance to run. Windows NT will run programs like CTOF with no problems since it preemptively multitasks.

Forms for Free

Data entry forms are pervasive, especially in GUI programming. Windows recognizes this and provides dialogs to simplify using forms from inside a Windows application.

However, many programs don't just *need* forms—they *are* forms. Many database applications fit this description. You fill in a form and update the database. Perhaps you fill in a form, query the database, and display one or more forms to show the results. Even if you don't ordinarily work with a full-scale database, you probably write programs like this on occasion.

Consider a typical online telephone directory. You want to store names, phone numbers, and perhaps some notes

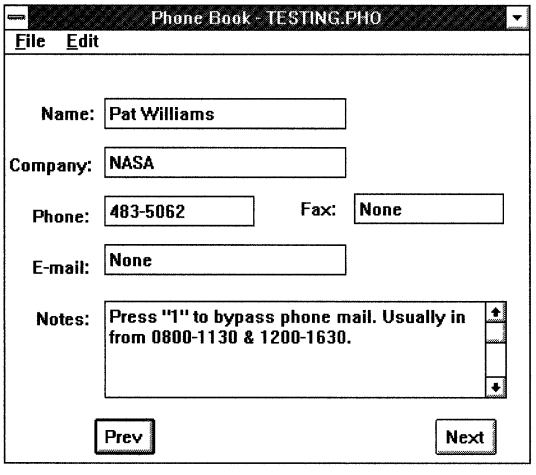
in a disk file. You'll need to allow for database insertions, deletions, and queries. Users also will want to browse randomly through the database.

This typical program clearly consists of two parts: a GUI interface and a database. With this in mind, our phonebook example consists of two main files: PHONE.C (Listing 3-13) and PHONEDB.C (Listing 3-14). PHONE.C contains the Windows GUI code—the event loop, the menu code, and other interface-related functions. PHONEDB.C is mainly database code. The only Windows-specific code in PHONEDB.C relates to reading and writing fields from the screen, using the built-in file to open dialogs, and manipulating the cursor. You'll also need the supporting file in Listings 3-15, 13-16, and 3-17 to compile PHONE.

Figure 3-3 on the next page shows the complete PHONE application's window. Unlike most dialog-based programs, PHONE has a menu bar and uses accelerators (keyboard shortcuts). This is possible because PHONE uses a regular window that the user can't see. The sole purpose of the window is to support the menu bar and accelerators.

Since the menu window must be active at the same time as the data entry screen, you must use a modeless dialog. The size of the window is adjusted so that the dialog just fits inside. When the window receives a WM_SETFOCUS message, it immediately transfers control to the dialog. In this way, the window is never in control.

Figure 3-3. The Phone Application



You specify the dimensions of a dialog box in device-independent units. Windows transforms these units at run time into reasonable sizes depending on the current display. You can obtain the dialog base unit by calling GetDialogBaseUnits(). To convert the height of a dialog box to device units, use:

```
deviceHi=dialogHi*HIWORD(GetDialogBaseUnits())/8;
```

The width is similar:

```
deviceWide=dialogWide*LOWORD(GetDialogBaseUnits())/4;
```

Of course, when adjusting the window's size, you have to add the height of the menu (GetSystemMetrics(SM_CY-

MENU)) and the caption bar (GetSystemMetrics(SM_ CY-CAPTION)).

Events

PHONE's event loop looks a little different from the one you have typically used in the past:

```
while (GetMessage(&msg,NULL,NULL,NULL))
    {
    if (!TranslateAccelerator(topwindow,acctable,&msg))
      {
      if (!maindlg||!IsDialogMessage(maindlg,&msg))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }
      }
    }
```

PHONE allows accelerator shortcut keys (like F1 for find). The TranslateAccelerator() call intercepts these keys and transforms them into conventional menu (WM_COMMAND) messages. The RC file defines the accelerator keys, and PHONE loads them into the acc-table variable.

Since modeless dialogs coexist with other windows in your program, the dialog events come to your event loop. If the IsDialogMessage() call returns TRUE, you are processing an event for the specified dialog. The IsDialog-

Message() call automatically routes the message to the dialog for you.

The GUI

PHONE uses three tools to simplify its GUI. It uses MessageBox() everywhere it can to avoid custom dialogs. The WPRINT module from earlier in the chapter provides the win_input() function that PHONE uses to prompt for queries. Finally, PHONE's file open-and-save routines use the Windows common-file dialogs. For just a little effort, these dialogs give your programs a polished look.

The form portion of PHONE makes heavy use of Window's built-in dialog processing. The disp_record() routine uses the WM_SETTEXT message to set the contents of each field, and the SetFocus() call to place the cursor on the first field.

The commit_record() routine is slightly more complex. It scans the modify flag in each field using the EM_GETMODIFY message. If the modify flag is set, the user has changed the field. Then, commit_record() uses the WM_GETTEXT message to retrieve the new text. Finally, calling EM_SETMODIFY with a zero argument resets the modify flag.

Using Common Dialogs

Common dialogs are a powerful Windows programming tool. Why write your own dialogs when Windows makes these powerful dialogs available to you?

Before using a common file dialog, you must initialize an OPENFILENAME structure (see Tables 3-1 and 3-2). You

must make sure that the entire structure is set to zero. Next, you place the size of the structure in its lStructSize field to allow future versions of Windows to accommodate older programs. Below is some typical code:

```
OPENFILENAME ofile;
memset(&ofile,0,sizeof(OPENFILENAME));
ofile.lStructSize=sizeof(OPENFILENAME);
```

Table 3-1. The OPENFILENAME Structure

Element	Type	Description
lStructSize	DWORD	Size of this structure in bytes.
hwndOwner	HWND	Owner of dialog (could be NULL).
hInstance	HINSTANCE	Program's instance handle (only used when templates are used).
lpstrFilter	LPCSTR	Pointer to file filter strings.
lpstrCustomFilter	LPSTR	Buffer to hold custom file filters.
nMaxCustFilter	DWORD	Size of above buffer.
nFilterIndex	DWORD	Initial filter index (starts at 1).
lpstrFile	LPSTR	Buffer to hold filename.
nMaxFile	DWORD	Size of above buffer.
lpstrFileTitle	LPSTR	Buffer to hold file title (see text). May be NULL if you don't need the file title.
nMaxFileTitle	DWORD	Size of above buffer.
lpstrInitialDir	LPCSTR	Initial directory. If NULL, use current directory.

(Cont.)

Table 3-1. The OPENFILENAME Structure (Cont.)

Element	Type	Description
lpstrTitle	LPCSTR	Title of dialog. If NULL, Windows provides a default.
Flags	DWORD	Controls dialog operation (see Table 3-2).
nFileOffset	UINT	Length of directory information in filename.
nFileExtension	UINT	Offset in filename of extension.
lpstrDefExt	LPCSTR	Default extension.
lCustData	LPARAM	Available for your use.
lpfnHook	Function Pointer	Custom message handler, if required.
lpTemplateName	LPCSTR	Name of custom dialog template if required.

Table 3-2. OPENFILENAME Flags.

Flag	Meaning
OFN_ALLOWMULTISELECT	Allows multiple selections.
OFN_CREATEPROMPT	Prompt before creating file that doesn't exist.
OFN_ENABLEHOOK	Use hook function (see lpfnHook).
OFN_ENABLETEMPLATE	Use custom template (see lpTemplateName).
OFN_ENABLETEMPLATE-HANDLE	Use custom template already loaded. hInstance actually contains a handle to the data block.
OFN_FILEMUSTEXIST	User can't select a nonexistent file.
OFN_HIDEREADONLY	Hides the read-only checkbox.

(Cont.)

Table 3-2. OPENFILENAME Flags. (Cont.)

Flag	Meaning
OFN_NOCHANGEDIR	Forces the dialog to reset the current directory before returning.
OFN_NOREADONLY-RETURN	Disallow files that are read-only.
OFN_NOTESTFILECREATE	Don't try to create file.
OFN_NOVALIDATE	Allows illegal characters in filenames.
OFN_OVERWRITEPROMPT	Prompt if selected file already exists.
OFN_PATHMUSTEXIST	Selected directory must exist.
OFN_SHAREAWARE	Ignore sharing errors.
OFN_READONLY	Initially check the read-only checkbox. Also reflects the state of the read-only checkbox upon return.
OFN_EXTENSION-DIFFERENT	Set to indicate that the returned fil name does not match the default extension.

The remaining fields allow you to set up options for the box. Some options are very simple; others can quickly get complicated. At a minimum, you'll want to set the hwnd-Owner, lpstrFilter, nFilterIndex, lpstrFile, nMaxFile, lpstrFileTitle, nMaxFileTitle, and Flags fields.

You can pass the OPENFILENAME structure to GetOpen-FileName() or GetSaveFileName(). These common dialogs return two forms of the filename to your program (via the lpstrFile and lpstrFileTitle fields). The filename is the entire pathname the user selected. The file title is just the base name of the file and is useful for placing in title bars, for example.

The return value from GetOpenFileName() is ordinarily nonzero. If it is zero, the user must have pressed cancel or there was an error. You can call CommDlgExtendedError() to detect an error. If the return value from CommDlgExtendedError() is zero, the user simply cancelled the operation.

Giving Good PHONE

PHONE is a useful program in its own right. But it has certain limitations you might not tolerate in a real program. One obvious addition to the program would be a sort routine. Such a routine is not difficult to add since the database is separate from the Windows portion of the program.

PHONE's memory allocation strategy is simple. It uses the standard library call calloc() to allocate zero-filled memory regions. For Windows NT, this isn't a problem. However, Windows 3.1 users will run out of memory when the database approaches 64K. Of course, a real version of PHONE would probably use an external database instead of storing each entry in memory.

PHONE could use GlobalAlloc() to access much more memory. You could rewrite zmalloc() like this:

```
LPSTR zmalloc(unsigned long siz)
  {
  return
    GlobalLock(
```

```
        GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT,siz));
    }
```

You would also have to replace free() with:

```
#define free(p) \
  GlobalFree(GlobalHandle(HIWORD(p)));
```

or for Windows NT:

```
#define free(p) \
  GlobalFree(GlobalHandle(p));
```

However, for Windows 3.1, you then face the LDT segment limit: you can only allocate about 8,000 regions from GlobalAlloc() before you exhaust the LDT. Each phone entry requires seven allocations, so you could run out quickly. Worse, the 8,000 limit applies to the system as a whole, not just your program. A better alternative is to allocate large chunks of memory from GlobalAlloc() and then parcel out smaller pieces to your program as you need them needed.

Printing the database would be a difficult task. Windows printing is somewhat complex and differs from Windows 3.1 to Windows NT. If you need printing, you might consider using an application framework (see Chapters 4, 6, and 7). Microsoft's MFC, for example, gives you printing and print preview with very little effort on your part.

Constructing Resources

Although the RC files in this chapter are simple, you should never create RC files by hand. Instead, use a resource editor, such as Borland's Resource Workshop, Microsoft's Application Studio, or the Whitewater Resource Toolkit. Remember Commandment VII: program visually when possible.

Modern resource editors allow you to create menus, dialogs, bitmaps, cursors, and icons. You can align elements of dialogs, and changing the location or text of a button is painless. Most resource editors allow you to run a simulation of a dialog or menu to check out its operation.

Listing 3-1. MOTD.C

```
/*******************************************************
 *                                                     *
 * File: MOTD.C                                        *
 *                                                     *
 * Display a Message Of The Day (MOTD).                *
 *                                                     *
 * Required to Compile:                                *
 * MOTD.C MOTD.DEF ARGCARGV.C ARGCARGV.H               *
 * (ARGCARGV.C only required for Microsoft)            *
 *                                                     *
 *******************************************************/
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "argcargv.h"

/* String to hold message */
char string[1025];

/* Main window function */
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                   LPSTR cmdline, int show)
  {
  FILE *f;
  /* set up _argc,_argv (not req'd for Borland, but
   * harmless) */
  set_args(cmdline, hInst);
  if (_argc > 1)
```

```
    f = fopen(_argv[1], "r");
  /* If no argument or file won't open... */
  if (_argc == 1 || !f)
    strcpy(string, "No message today");
  else
      {
      /* read string from file if OK */
      fgets(string, sizeof(string), f);
      fclose(f);
      }
  /* Show it... */
  MessageBox(NULL, string, "Message for today",
        MB_OK | MB_ICONINFORMATION);
  return FALSE;
  }
```

Listing 3-2. MOTD.DEF

```
Name MOTD
Description 'Message of the day'
Exetype WINDOWS
Code PRELOAD MOVEABLE DISCARDABLE
Data PRELOAD MOVEABLE SINGLE
Heapsize 4096
Stacksize 5120
Stub 'WINSTUB.EXE'
```

Listing 3-3. CDPLAYER.C

```
/*****************************************************
 *                                                   *
 * File: CDPLAYER.C                                  *
```

```
 *                                                          *
 * MCI Audio CD Player. Uses only a menu.                   *
 * You must link with MMSYSTEM.LIB (a standard              *
 * Windows library).                                        *
 *                                                          *
 * Required to Compile:                                     *
 * CDPLAYER.C CDPLAYER.H CDPLAYER.RC CDPLAYER.DEF           *
 *                                                          *
 ************************************************************/
#include <windows.h>
#include <mmsystem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cdplayer.h"


/* current instance */
HANDLE hInst;
/* main window */
HWND topwindow;
/* Door open? */
int door_open = 0;

/* This routine calls MCI and displays an error box if
 * needed */
void mci_call(char *cmd, char *ret, unsigned siz,
              HWND w, int noerr)
  {
  DWORD rc;
```

```
  char msgbuf[257], *msg = msgbuf;
  rc = mciSendString(cmd, ret, siz, w);
  if (rc && !noerr)
      {
      rc = mciGetErrorString(rc, msgbuf, sizeof(msgbuf));
      if (!rc)
         msg = "Unknown MCI error";
      MessageBox(NULL, msg, NULL, MB_OK | MB_ICONSTOP);
      }
  }

/* Main window function */
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                LPSTR cmdline, int show)
  {
  MSG msg;
  /* Only allow 1 CDPLAYER at a time */
  if (prev)
      {
      MessageBox(NULL, "CDPLAYER already running!",
       "Error", MB_ICONSTOP | MB_OK);
      return FALSE;
      }
  if (!init(hInst, prev, show))
    return FALSE;

  /* Vanilla event loop */
  while (GetMessage(&msg, NULL, NULL, NULL))
      {
      TranslateMessage(&msg);
```

```
        DispatchMessage(&msg);
        }
    /* Exit program */
    return (msg.wParam);
    }

/* Start up stuff */
int init(HANDLE hInst, HANDLE prev, int show)
    {
    if (!prev)
        {
        if (!init_app(hInst))
     /* Exit if unable to initialize */
          return FALSE;
        }
    /* Perform instance init */
    if (!init_inst(hInst, show))
      return FALSE;
    return TRUE;
    }

/* Create window class here */
BOOL init_app(HANDLE hInstance)
    {
    WNDCLASS wc;
    wc.style = NULL;
    wc.lpfnWndProc = (void FAR *) win_proc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
```

```
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName = "CDMENU";
  wc.lpszClassName = "CD_Class";
  return (RegisterClass(&wc));
  }



/* Create window here */
BOOL init_inst(HANDLE hInstance, int nCmdShow)
  {
  HWND hWnd;
  /* Save the instance handle in global variable */
  hInst = hInstance;

  /* Create a main window */
  topwindow = hWnd = CreateWindow(
              "CD_Class",
              "CD Player",
       WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
              WS_MINIMIZEBOX,
              CW_USEDEFAULT,
              CW_USEDEFAULT,
              350,
              GetSystemMetrics(SM_CYMENU)
              + GetSystemMetrics(SM_CYCAPTION) + 2,
              NULL,
              NULL,
```

```
                hInstance,
                NULL
      );

  if (!hWnd)
    return FALSE;

  /* Make the window visible, update its client area, and
   * return "success" */
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  return (TRUE);
  }



/* Window procedure */
long WINAPI _export win_proc(HWND hWnd, UINT message,
                   UINT wParam, LONG lParam)
  {
  switch (message)
      {
      case WM_CREATE:
      /* Set door closed so we know the state. If you
       * manually open/close the door, we get in trouble
       * since door_open won't be correct... */
      mci_call("set cdaudio door closed wait",
                NULL, 0, NULL, 1);
      break;
```

```
    case WM_COMMAND:
      menu(hWnd, wParam);
      break;

    case WM_DESTROY:
      /* Simulate a STOP menu selection */
      menu(hWnd, IDM_STOP);
      PostQuitMessage(0);
      break;

    default:
      return (DefWindowProc(hWnd, message,
              wParam, lParam));
      }
  return NULL;
  }



/* Process menu events */
void menu(HWND hWnd, UINT wParam)
  {
  /* return from MCI */
  char mciout[256];
  int value;
  switch (LOWORD(wParam))
      {
    case IDM_ABOUT:
      MessageBox(topwindow,
```

```
    "CDPLAYER Version 1.0 by Al Williams",
    "About", MB_OK | MB_ICONINFORMATION);
  return;

case IDM_QUIT:
  DestroyWindow(hWnd);
  return;

case IDM_STOP:
  mci_call("stop cdaudio wait", NULL, 0, NULL, 0);
  return;


case IDM_EJECT:
  if (door_open)
    mci_call("set cdaudio door closed wait",
             NULL, 0, NULL, 1);
  else
    mci_call("set cdaudio door open wait",
             NULL, 0, NULL, 1);
  door_open ^= 1;
  return;

case IDM_TRACKUP:
case IDM_TRACKDN:
  mci_call("set cdaudio time format TMSF",
      NULL, 0, NULL, 0);
  mci_call("status cdaudio current track wait",
      mciout, sizeof(mciout), NULL, 0);
  value = atoi(mciout);
```

```
      value += wParam == IDM_TRACKUP ? 1 : -1;

      if (value < 1)
        value = 1;
      sprintf(mciout, "seek cdaudio to %d wait", value,
0);
      mci_call(mciout, NULL, 0, NULL, 0);
      //Fall into play case
    case IDM_PLAY:
        {
   /* This will fail if the device is not ready... */
        mci_call("play cdaudio", NULL, 0, NULL, 0);
        return;
        }
      }
  }
```

Listing 3-4. CDPLAYER.H

```
/*****************************************************
 *                                                   *
 * File: CDPLAYER.H                                  *
 *                                                   *
 * Header for CDPLAYER.C                             *
 *                                                   *
 * Required to Compile:                              *
 * CDPLAYER.C CDPLAYER.H CDPLAYER.RC CDPLAYER.DEF    *
 *                                                   *
 *****************************************************/
/* Menu defines */
#define IDM_ABOUT 100
```

```
#define IDM_QUIT 101
#define IDM_PLAY 102
#define IDM_STOP 103
#define IDM_EJECT 104
#define IDM_TRACKUP 105
#define IDM_TRACKDN 106

/* Prototypes */
BOOL init_app(HANDLE);
BOOL init_inst(HANDLE, int);
long WINAPI _export win_proc(HWND, UINT, UINT, LONG);
void menu(HWND, UINT);
int init(HANDLE, HANDLE, int);
```

Listing 3-5. CDPLAYER.RC

```
/*********************************************************
 *                                                       *
 * File: CDPLAYER.RC                                     *
 *                                                       *
 * Menu for CDPLAYER.C                                   *
 *                                                       *
 * Required to Compile:                                  *
 * CDPLAYER.C CDPLAYER.H CDPLAYER.RC CDPLAYER.DEF        *
 *                                                       *
 *********************************************************/
#include "windows.h"
#include "cdplayer.h"

CDMENU MENU
BEGIN
```

```
    POPUP         "&File"
    BEGIN
        MENUITEM "&About CDPlayer...",IDM_ABOUT
        MENUITEM "&Exit",IDM_QUIT
    END
    MENUITEM "&Play", IDM_PLAY
    MENUITEM "&Stop", IDM_STOP
    MENUITEM "&Eject", IDM_EJECT
    MENUITEM "&+Track", IDM_TRACKUP
    MENUITEM "&-Track", IDM_TRACKDN
END
```

Listing 3-6. CDPLAYER.DEF

```
Name CDPLAYER
     Description 'MENU driven CD Player'
     Exetype WINDOWS
     Code PRELOAD MOVEABLE DISCARDABLE
     Data PRELOAD MOVEABLE SINGLE
     Heapsize 4096
     Stacksize 5120
     Stub 'WINSTUB.EXE'
```

Listing 3-7. WPRINT.C

```
/*****************************************************
 *                                                   *
 * File: WPRINT.C                                    *
 *                                                   *
 * Supply win_printf() and win_input dialogs to      *
 * other programs.                                   *
 *                                                   *
```

```
 * Required to Compile:                                    *
 * Many programs use WPRINT.C, WPRINT.RC and               *
 * WPRINT.H. They do not compile separately.               *
 *                                                         *
 ***********************************************************/
#include <windows.h>
#include <stdio.h>
#include <stdarg.h>

/* Buffers for I/O */
static char print_buf[513];
static char in_buf[513];
/* current app's instance -- must be set by caller */
extern HANDLE hInst;

/* Sort of printf for Windows -- maximum output is 512
 * characters */
int win_printf(char *title, char *fmt,...)
  {
  int rc;
  va_list alist;
  va_start(alist, fmt);
  rc = vsprintf(print_buf, fmt, alist);
  if (rc != -1)
    MessageBox(NULL, print_buf, title ? title : "Output",
          MB_OK);
  return rc;
  }
```

```
/* Dialog callback for input function */
BOOL FAR PASCAL _export inp_dlg(HWND hDlg,
                    unsigned message,
                  WORD wParam, LONG lParam)
  {
  switch (message)
      {
    case WM_INITDIALOG:
      /* Set title */
      SendMessage(hDlg, WM_SETTEXT, 0, lParam);
      /* Set prompt */
      SetDlgItemText(hDlg, 101, print_buf);
      return (TRUE);

    case WM_COMMAND:
      if (wParam == IDOK)
       {
     /* read input */
       GetDlgItemText(hDlg, 102, in_buf, sizeof(in_buf));
       EndDialog(hDlg, TRUE);
       return (TRUE);
       }
     break;
     }
  return FALSE;
  }


/* line input for Windows title is the dialog title -- the
```

```
 * remaining arguments are used as a prompt */
char *win_input(char *title, char *fmt,...)
  {
  FARPROC dlgfunc;
  int rc;
  va_list alist;
  va_start(alist, fmt);
  /* print prompt to print_buf */
  rc = vsprintf(print_buf, fmt, alist);
  if (rc != -1)
      {
      /* Get instance of inp_dlg() function */
      dlgfunc = MakeProcInstance(inp_dlg, hInst);
      if (!dlgfunc)
        win_printf("Error", "MakeProcInstance failed");
      else
       {
     /* call dialog */
       rc = DialogBoxParam(hInst, "InputBox",
               NULL,
               dlgfunc, (long) title);
       FreeProcInstance(dlgfunc);
       }
      }
  return in_buf;
  }
```

Listing 3-8. WPRINT.H

```
/*****************************************************
 *                                                   *
```

```
 * File: WPRINT.H                                          *
 *                                                         *
 * Supply win_printf() and win_input dialogs to            *
 * other programs.                                         *
 *                                                         *
 * Required to Compile:                                    *
 * Many programs use WPRINT.C, WPRINT.RC and               *
 * WPRINT.H. They do not compile separately.               *
 *                                                         *
 ***********************************************************/
#ifndef WPRINT_H
#define WPRINT_H
int win_printf(char *title, char *fmt,...);
char *win_input(char *title, char *fmt,...);
#endif
```

Listing 3-9. WPRINT.RC

```
/***********************************************************
 *                                                         *
 * File: WPRINT.RC                                         *
 *                                                         *
 * Supply win_printf() and win_input dialogs to            *
 * other programs. Programs that use WPRINT must           *
 * include this RC file in their own RC file.              *
 *                                                         *
 * Required to Compile:                                    *
 * Many programs use WPRINT.C, WPRINT.RC, and              *
 * WPRINT.H. They do not compile separately.               *
 *                                                         *
 ***********************************************************/
```

```
InputBox DIALOG 100,100,144,75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "Input"
BEGIN
   LTEXT "Prompt" 101,0,5,144,32
   EDITTEXT 102,10,34,130,12,WS_TABSTOP | ES_AUTOHSCROLL
   DEFPUSHBUTTON "OK" IDOK,53,59,32,14,WS_GROUP
END
```

Listing 3-10. CTOF.C

```
/*********************************************************
 *                                                       *
 * File: CTOF.C                                          *
 *                                                       *
 * Convert temperatures (C to F) using only dialogs.     *
 *                                                       *
 * Required to Compile:                                  *
 * CTOF.C CTOF.DEF CTOF.RC WPRINT.C WPRINT.RC            *
 * WPRINT.H                                              *
 *                                                       *
 *********************************************************/
#include <windows.h>
#include "wprint.h"
#include <stdlib.h>
#include <string.h>

/* WPRINT requires this global to be set */
HANDLE hInst;

/* Main window function */
```

```
int PASCAL WinMain(HANDLE hInstance, HANDLE prev,
                   LPSTR cmdline, int show)
  {
  char *input;
  double deg;
  hInst = hInstance;
  /* Do "forever" */
  while (1)
      {
      /* Get C value via win_input dialog */
      input = win_input("Temperature Conversion C to F",
         "Enter the temperature in C");
      /* If user just hit enter, quit! */
      if (!*input)
         return FALSE;
      /* Do calculations */
      deg = atof(input);
      deg = 9.0 / 5.0 * deg + 32.0;
      /* Show results */
      win_printf("Result", "Degrees F=%.1f", deg);
      }
  }
```

Listing 3-11. CTOF.RC

```
/*****************************************************
 *                                                   *
 * File: CTOF.RC                                     *
 *                                                   *
 * Includes the WPRINT.RC file required by CTOF.C    *
 *                                                   *
```

```
 * Required to Compile:                                        *
 * CTOF.C CTOF.DEF CTOF.RC WPRINT.C WPRINT.RC                  *
 * WPRINT.H                                                    *
 *                                                             *
 *************************************************************/
#include <windows.h>
#include "wprint.rc"
```

Listing 3-12. CTOF.DEF

```
Name CTOF
Description 'Convert temps'
Exetype WINDOWS
Code PRELOAD MOVEABLE DISCARDABLE
Data PRELOAD MOVEABLE SINGLE
Heapsize 4096
Stacksize 5120
Stub 'WINSTUB.EXE'
```

Listing 3-13. PHONE.C

```
/*************************************************************
 *                                                           *
 * File: PHONE.C                                             *
 *                                                           *
 * Windows portion of Phonebook application.                 *
 * You must link with COMMDLG.LIB (a standard                *
 * Windows library).                                         *
 *                                                           *
 * Required to Compile:                                      *
 * PHONE.C PHONE.H PHONEDB.C PHONE.DEF PHONE.RC              *
 * WPRINT.C WPRINT.H WPRINT.RC                               *
```

```
 *                                                              *
 ****************************************************/
#include <windows.h>
#include "wprint.h"
#include "phone.h"

/* current instance */
HANDLE hInst;
/* main window */
HWND topwindow;
/* main dialog */
HWND maindlg;

/* Main window function */
int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                   LPSTR cmdline, int show)
  {
  MSG msg;
  HACCEL acctable;
  if (!init(hInst, prev, show))
    return FALSE;
  /* load accelerators */
  acctable = LoadAccelerators(hInst, "PHONEMENU");
  /* Event loop for modeless dialog with accel */
  while (GetMessage(&msg, NULL, NULL, NULL))
      {
      if (!TranslateAccelerator(topwindow, acctable, &msg))
        {
        if (!maindlg || !IsDialogMessage(maindlg, &msg))
         {
```

```
            TranslateMessage(&msg);
            DispatchMessage(&msg);
            }
          }
        }
    /* Exit program */
    return (msg.wParam);
    }

/* Start up stuff */
int init(HANDLE hInst, HANDLE prev, int show)
    {
    if (!prev)
        {
        if (!init_app(hInst))
     /* Exit if unable to initialize */
          return FALSE;
        }
    /* Perform instance init */
    if (!init_inst(hInst, show))
      return FALSE;
    return TRUE;
    }

/* Create window class here */
BOOL init_app(HANDLE hInstance)
    {
    WNDCLASS wc;
    wc.style = NULL;
    wc.lpfnWndProc = (void FAR *) win_proc;
```

```
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hInstance = hInstance;
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName = "PHONEMENU";
  wc.lpszClassName = "PHONE_Class";
  return (RegisterClass(&wc));
  }



/* Create window here */
BOOL init_inst(HANDLE hInstance, int nCmdShow)
  {
  HWND hWnd;
  /* Save the instance handle in global variable */
  hInst = hInstance;

  /* Create a main window */
  topwindow = hWnd = CreateWindow(
              "Phone_Class",
              "Phone Book",
       DS_MODALFRAME | WS_SYSMENU | WS_VISIBLE |
              WS_MINIMIZEBOX,
              CW_USEDEFAULT,
              CW_USEDEFAULT,
  /* If you look in the header and the RC file, you will see
   * that the dialog is DLG_WID X DLG_HI units in size.
```

```
    *
    * We use GetDialogBaseUnits() to convert these numbers to
    * window units and make the window exactly the same size
    * as the dialog (of course, we have to leave room for
    * the caption bar and the menu, too). */
          DLG_WID * LOWORD(GetDialogBaseUnits()) / 4,
          DLG_HI * HIWORD(GetDialogBaseUnits()) / 8 +
          GetSystemMetrics(SM_CYMENU)
          + GetSystemMetrics(SM_CYCAPTION),
               NULL,
               NULL,
               hInstance,
               NULL
      );

  if (!hWnd)
    return FALSE;
  /* Make the window visible, update its client area, and
   * return "success" */
  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  return (TRUE);
  }

/* Main dialog callback for modeless dialog */
BOOL WINAPI _export dialog_func(HWND hDlg, UINT message,
                  UINT wParam, LONG lParam)
  {
  switch (message)
      {
```

```
    case WM_INITDIALOG:
      return TRUE;

    case WM_COMMAND:
      if (wParam == PREV_BUTTON)
       {
     /* commit current record */
       commit_record(current);
     /* display previous */
       disp_record(
          (current->prev && current->prev != &head) ?
          (current = current->prev) : current);
        }
      if (wParam == NEXT_BUTTON)
        {
     /* commit current record */
        commit_record(current);
     /* display next */
        disp_record(current->next ?
            (current = current->next) : current);
        }
      break;
      }
  return FALSE;
  }


/* Window procedure */
long WINAPI _export win_proc(HWND hWnd, UINT message,
                  UINT wParam, LONG lParam)
```

```
{
switch (message)
    {
    static FARPROC dlgfunc;
  case WM_CREATE:
    /* Init db */
    new_record();      /* Init database */
    current = head.next;
    /* create modeless dialog */
    dlgfunc = MakeProcInstance(dialog_func, hInst);
    maindlg = CreateDialog(hInst, "PHONEDLG",
            hWnd, dlgfunc);
    break;

  case WM_COMMAND:
    menu(hWnd, wParam);
    break;
  case WM_DESTROY:
    /* Reset db -- this will prompt for save if needed */
    reset_db();
    DestroyWindow(maindlg);
    FreeProcInstance(dlgfunc);
    PostQuitMessage(0);
    break;

  case WM_SETFOCUS:
    /* Never take focus -- always shift to maindlg */
    SetFocus(maindlg);
    break;
```

```
    default:
      return (DefWindowProc(hWnd, message,
              wParam, lParam));
      }
  return NULL;
  }

void menu(HWND hWnd, UINT wParam)
  {
  switch (LOWORD(wParam))
      {
    case IDM_ABOUT:
      MessageBox(topwindow,
       "Phone Book Version 1.0 by Al Williams",
       "About", MB_OK | MB_ICONINFORMATION);
      return;

    case IDM_EXIT:
      DestroyWindow(hWnd);
      return;

    case IDM_LOAD:
      /* display first record */
      read_file();
      break;

    case IDM_SAVE:
      /* Commit current record */
      commit_record(current);
      /* write out file */
```

```
      write_file(0);
      break;

    case IDM_SAVEAS:
      /* Commit current record */
      commit_record(current);
      write_file(1);
      break;

    case IDM_FIND:
        {
        char *target;
        struct record *p;
        target = win_input("Find Record", "Name to
find?");
   /* If user just hits enter, forget it */
        if (!*target)
           break;
        p = find_record(target);
        if (!p)
           MessageBox(topwindow, "Record not found", NULL,
                      MB_OK | MB_ICONSTOP);
        else
           disp_record(current = p);
        break;
        }

    case IDM_DEL:
      if (MessageBox(topwindow, "Delete record?",
          "Confirm", MB_YESNO | MB_ICONQUESTION) == IDYES)
```

```
            current = del_record(current);
        disp_record(current);
        break;

    case IDM_NEWDB:
        new_db();
        current = head.next;
        break;

    case IDM_NEWREC:
        /* insert blank record */
        current = new_record();
        /* display it */
        disp_record(current);
        break;
        }

  }
```

Listing 3-14. PHONEDB.C

```
/*****************************************************
 *                                                   *
 * File: PHONEDB.C                                   *
 *                                                   *
 * Database portion of Phonebook application.        *
 *                                                   *
 * Required to Compile:                              *
 * PHONE.C PHONE.H PHONEDB.C PHONE.DEF PHONE.RC      *
 * WPRINT.C WPRINT.H WPRINT.RC                       *
 *                                                   *
```

```
 ****************************************************/
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <commdlg.h>
#include <stdlib.h>
#include "phone.h"
#include "wprint.h"

/* set to 1 when phonebook was modified */
int dirty;
/* file name of current phonebook */
char fn[256];

/* Phone book is stored as link list. head is a dummy
 * record -- only its next field is used.
 * The current variable always points to the current
 * record */
struct record head, *current;

/* Filters for common dialog */
char filefilter[] = "Phone files (*.pho)\0*.pho\0"
                    "All files (*.*)\0*.*\0";

#ifndef NT
#define zmalloc(n) calloc(1,n)
#else
/* Work around memset problem in some early NT versions --
 * also affected calloc() */
void *zmalloc(unsigned int n)
```

```
  {
  void *p = malloc(n);
  char *fill = (char *) p;
  if (p)
      {
      unsigned int i;
      for (i = 0; i < n; i++)
          fill[i] = 0;
      }
  return p;
  }

#endif

/* make new record */
struct record *new_record()
  {
  struct record *db = &head;
  int i;
  while (db->next)
    db = db->next;
  /* zmalloc allocates zero-filled memory */
  db->next = zmalloc(sizeof(struct record));
  db->next->prev = db;
  for (i = 0; i < NRFIELDS; i++)
    db->next->fields[i] = zmalloc(1);
  return db->next;
  }

/* delete current record, returns next record */
```

```
struct record *del_record(struct record * r)
  {
  struct record *n, *p;
  int i;
  n = r->next;
  p = r->prev;
  if (p)
    p->next = n;
  if (n)
    n->prev = p;
  for (i = 0; i < NRFIELDS; i++)
    free(r->fields[i]);
  free(r);
  r = n ? n : p;
  if (r == &head)
    r = new_record();
  dirty = 1;
  return r;
  }


/* Clear out database */
void reset_db()
  {
  struct record *db, *next;
  int i;
  if (head.next)
      {
      commit_record(current);
      if (dirty &&
```

```
          MessageBox(topwindow, "Save current file?",
            "Confirm",MB_YESNO | MB_ICONQUESTION) == IDYES)
        write_file(1);
      db = head.next;
      while (db)
        {
        for (i = 0; i < NRFIELDS; i++)
          free(db->fields[i]);
        next = db->next;
        free(db);
        db = next;
        }
      head.next = NULL;
      }
  dirty = 0;
  *fn = '\0';
  }

/* Make new empty database */
void new_db()
  {
  reset_db();
  new_record();
  }

/* Display record, set cursor in name field and select
 * contents */
void disp_record(struct record * r)
  {
  int i;
```

```
  for (i = 0; i < NRFIELDS; i++)
      {
      SendDlgItemMessage(maindlg, NAME_FIELD + i,
            WM_SETTEXT, 0, (long) r->fields[i]);
      }
  /* Put cursor in NAME_FIELD */
  SetFocus(GetDlgItem(maindlg, NAME_FIELD));
  /* Select contents of NAME_FIELD */
  SendDlgItemMessage(maindlg, NAME_FIELD, EM_SETSEL,
          0, MAKELONG(32767, 0));
  }

/* If any fields in the dialog have been changed, update
 * the indicated record */
void commit_record(struct record * r)
  {
  int i;
  char tbuf[256];
  for (i = 0; i < NRFIELDS; i++)
      {
      /* continue if no change to field */
      if (!SendDlgItemMessage(maindlg, NAME_FIELD + i,
              EM_GETMODIFY, 0, 0))
         continue;
      /* set dirty flag */
      dirty = 1;
      /* free old field */
      free(r->fields[i]);
      /* fetch text */
      SendDlgItemMessage(maindlg, NAME_FIELD + i,
```

```
            WM_GETTEXT, sizeof(tbuf),
            (long) tbuf);
        /* make new field buffer */
        /* zmalloc allocates zero-filled memory */
        r->fields[i] = zmalloc(strlen(tbuf) + 1);
        strcpy(r->fields[i], tbuf);
        /* clear modify flag */
        SendDlgItemMessage(maindlg, NAME_FIELD + i,
            EM_SETMODIFY, 0, 0);
        }
    }

/* find a record by name -- compare is not case sensitive
 * If no match is found, we look for a partial match. */
struct record *find_record(char *target)
    {
    struct record *db = head.next;
    int len;
    /* try whole name search first */
    while (db)
      if (!stricmp(db->fields[0], target))
        return db;
      else
        db = db->next;
    /* not found... try partial search */
    db = head.next;
    len = strlen(target);
    while (db)
      if (!strnicmp(db->fields[0], target, len))
        return db;
```

```
    else
      db = db->next;
  /* None found */
  return NULL;
  }


/* Write file out. If asflag is 1 or the global fn is not
 * set, we bring up a save dialog box. Otherwise, we just
 * save to the file in fn. */
void write_file(int asflag)
  {
  OPENFILENAME ofile;
  FILE *out;
  char ft[256], tbuf[256];
  int err, field, stat;
  HCURSOR cursor;
  struct record *db;
  if (asflag || !*fn)
      {
      asflag = 1;
      /* common file save dialog setup */
      memset(&ofile, 0, sizeof(OPENFILENAME));
      *fn = *ft = '\0';
      ofile.lStructSize = sizeof(OPENFILENAME);
      ofile.hwndOwner = topwindow;
      ofile.lpstrFilter = filefilter;
      ofile.nFilterIndex = 1;
      ofile.lpstrFile = fn;
```

```
    ofile.nMaxFile = sizeof(fn);
    ofile.lpstrFileTitle = ft;
    ofile.nMaxFileTitle = sizeof(ft);
    ofile.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
      OFN_NOREADONLYRETURN | OFN_OVERWRITEPROMPT;
    }
/* if asflag==0 then save file under previous name */
if (!asflag)
    {
    stat = (out = fopen(fn, "w")) == NULL;
    err = 1;
    }
else
  /* bring up dialog and open file */
  stat = !(err = GetSaveFileName(&ofile)) ||
     !(out = fopen(fn, "w"));
if (stat)
    {
    /* If err is FALSE then might just be a cancel
     * CommDlgExtendedError returns 0 if it was a
       cancel */
    if (err || CommDlgExtendedError())
       MessageBox(topwindow, "File open error", NULL,
                  MB_ICONSTOP | MB_OK);
    return;
    }
if (asflag)
    {
    /* Set title if filename changed */
    /* wsprintf is Window's built-in version of sprintf().
```

```
        * Be sure to always cast near strings to far for
        * this function */
       wsprintf(tbuf, "Phone Book - %s", (char FAR *) ft);
       SendMessage(topwindow, WM_SETTEXT, 0, (LONG) tbuf);
       }
   /* Wait cursor */
   cursor = SetCursor(LoadCursor(NULL, IDC_WAIT));
   /* write file */
   for (db = head.next; db; db = db->next)
       {
       for (field = 0; field < NRFIELDS; field++)
          fprintf(out, "%s%c", db->fields[field], 0);
       }
   /* Done with file */
   fclose(out);
   dirty = 0;
   /* Restore cursor */
   SetCursor(cursor);
   }

/* Load file from disk */
void read_file()
   {
   OPENFILENAME ofile;
   FILE *in;
   char tbuf[256], ft[256];
   int err, field, cp;
   HCURSOR cursor;
   struct record *db;
   /* clear database */
```

```
reset_db();
/* set up open file dialog */
memset(&ofile, 0, sizeof(OPENFILENAME));
*fn = *ft = '\0';
ofile.lStructSize = sizeof(OPENFILENAME);
ofile.hwndOwner = topwindow;
ofile.lpstrFilter = filefilter;
ofile.nFilterIndex = 1;
ofile.lpstrFile = fn;
ofile.nMaxFile = sizeof(fn);
ofile.lpstrFileTitle = ft;
ofile.nMaxFileTitle = sizeof(ft);
ofile.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
   OFN_FILEMUSTEXIST;
/* bring up dialog and open file */
if (!(err = GetOpenFileName(&ofile)) ||
    !(in = fopen(fn, "r")))
    {
    /* If err is FALSE then might just be a cancel
     * CommDlgExtendedError returns 0 if it was a
       cancel */
    if (err || CommDlgExtendedError())
       MessageBox(topwindow, "File open error", NULL,
                  MB_ICONSTOP | MB_OK);
    return;
    }
/* set title */
/* wsprintf is Window's built-in version of sprintf(). Be
 * sure to always cast near strings to far for this
 * function */
```

```
wsprintf(tbuf, "Phone Book - %s", (char FAR *) ft);
SendMessage(topwindow, WM_SETTEXT, 0, (LONG) tbuf);
/* Wait (hourglass) cursor */
cursor = SetCursor(LoadCursor(NULL, IDC_WAIT));
/* read file */
db = &head;
while (!feof(in))
    {
    for (field = 0; field < 6; field++)
      {
      int c;
      cp = 0;
   /* read to NULL or EOF */
      while (tbuf[cp++] = c = getc(in))
        if (c == EOF)
           break;
      if (c == EOF)
         break;
      if (field == 0)
         {
       /* zmalloc allocates zero-filled memory */
         db->next = zmalloc(sizeof(struct record));
         db->next->prev = db;
         db = db->next;
         }
      db->fields[field] = zmalloc(strlen(tbuf) + 1);
      strcpy(db->fields[field], tbuf);
      }
    }
/* Done with file */
```

```
  fclose(in);
  disp_record(current = head.next);
  /* Restore cursor */
  SetCursor(cursor);
  }
```

Listing 3-15. PHONE.H

```
/*******************************************************
 *                                                     *
 * File: PHONE.H                                       *
 *                                                     *
 * Header for Phonebook application.                   *
 *                                                     *
 * Required to Compile:                                *
 * PHONE.C PHONE.H PHONEDB.C PHONE.DEF PHONE.RC        *
 * WPRINT.C WPRINT.H WPRINT.RC                         *
 *                                                     *
 *******************************************************/
#define IDM_LOAD 1
#define IDM_SAVE 2
#define IDM_EXIT 3
#define IDM_ABOUT 4
#define IDM_FIND 5
#define IDM_DEL 6
#define IDM_NEWREC 7
#define IDM_NEWDB 8
#define IDM_SAVEAS 9

#define NAME_FIELD 101
#define CO_FIELD 102
```

```
#define NR_FIELD 103
#define FAX_FIELD 104
#define EMAIL_FIELD 105
#define NOTES 106
#define PREV_BUTTON 107
#define NEXT_BUTTON 108


/* 6 fields in dialog box ... we assume they are
 * consecutive numbers (e.g., NAME_FIELD+1 is the field
 * after NAME_FIELD) */
#define NRFIELDS (NOTES-NAME_FIELD+1)

extern HANDLE hInst;
extern HWND topwindow;
extern HWND maindlg;
extern int dirty;
extern char fn[];

extern struct record
  {
  char *fields[NRFIELDS];
  struct record *prev;
  struct record *next;
  }      head, *current;


struct record *new_record(void);
struct record *del_record(struct record * r);
void reset_db(void);
```

```
void new_db(void);
void disp_record(struct record * r);
void commit_record(struct record * r);
struct record *find_record(char *target);
void write_file(int asflag);
void read_file(void);

int PASCAL WinMain(HANDLE hInst, HANDLE prev,
                LPSTR cmdline, int show);
     int init(HANDLE hInst, HANDLE prev, int show);
     BOOL init_app(HANDLE hInstance);
     BOOL init_inst(HANDLE hInstance, int nCmdShow);
     BOOL WINAPI _export dialog_func(HWND hDlg,
             UINT message, UINT wParam, LONG lParam);
     long WINAPI _export win_proc(HWND hWnd, UINT message,
              UINT wParam, LONG lParam);
     void menu(HWND hWnd, UINT wParam);

/* Size of dialog box (and, therefore, of PHONE's window) */
#define DLG_WID 207
#define DLG_HI 160
```

Listing 3-16. PHONE.RC

```
/*****************************************************
 *                                                   *
 * File: PHONE.RC                                    *
 *                                                   *
 * Resources for Phonebook application.              *
 *                                                   *
```

```
 * Required to Compile:                                        *
 * PHONE.C PHONE.H PHONEDB.C PHONE.DEF PHONE.RC                *
 * WPRINT.C WPRINT.H WPRINT.RC                                 *
 *                                                             *
 ***************************************************************/
#include <windows.h>
#include "phone.h"

#include "wprint.rc"

PHONEMENU MENU
BEGIN
     POPUP     "&File"
     BEGIN
          MENUITEM "&New", IDM_NEWDB
          MENUITEM "&Open...",IDM_LOAD
          MENUITEM "&Save",IDM_SAVE
          MENUITEM "S&ave as...",IDM_SAVEAS
          MENUITEM SEPARATOR
          MENUITEM "&Exit",IDM_EXIT
          MENUITEM "&About Phone Book...",IDM_ABOUT
     END
     POPUP     "&Edit"
     BEGIN
          MENUITEM "&Find entry...\tF1",IDM_FIND
          MENUITEM "&New entry\tShift+Ins",IDM_NEWREC
          MENUITEM "&Delete entry\tShift+Del",IDM_DEL
     END
END
```

```
PHONEMENU ACCELERATORS
BEGIN
      VK_F1, IDM_FIND, VIRTKEY
      VK_INSERT, IDM_NEWREC, VIRTKEY,SHIFT
      VK_DELETE, IDM_DEL, VIRTKEY,SHIFT
END


PHONEDLG DIALOG 0, 0, DLG_WID, DLG_HI
STYLE WS_CHILD | WS_VISIBLE
BEGIN
  EDITTEXT NAME_FIELD, 39, 17, 95, 12, ES_AUTOHSCROLL
  EDITTEXT CO_FIELD, 39, 36, 95, 12, ES_AUTOHSCROLL
  EDITTEXT NR_FIELD, 39, 55, 59, 12, ES_AUTOHSCROLL
  EDITTEXT FAX_FIELD, 137, 54, 59, 12, ES_AUTOHSCROLL
  EDITTEXT EMAIL_FIELD, 39, 74, 95, 12, ES_AUTOHSCROLL
  EDITTEXT NOTES, 39, 96, 158, 38,
    WS_TABSTOP |WS_VSCROLL
    |ES_MULTILINE |ES_AUTOVSCROLL
  RTEXT "Name:", -1, 14, 19, 21, 8,
    SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  RTEXT "Company:", -1, 1, 39, 33, 8,
    SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  RTEXT "Phone:", -1, 7, 59, 26, 8,
    SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  LTEXT "Fax:", -1, 116, 56, 16, 8,
    WS_CHILD | WS_VISIBLE | WS_GROUP
  RTEXT "E-mail:", -1, 8, 79, 25, 8,
    SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  RTEXT "Notes:", -1, 10, 99, 23, 8,
```

```
    SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  PUSHBUTTON "Prev", PREV_BUTTON, 35, 142, 24, 14,
    WS_CHILD | WS_VISIBLE | WS_TABSTOP
  PUSHBUTTON "Next", NEXT_BUTTON, 169, 142, 24, 14,
    WS_CHILD | WS_VISIBLE | WS_TABSTOP
END
```

Listing 3-17. PHONE.DEF

```
Name Phone
Description 'Phone Book by Al Williams'
Exetype WINDOWS
Code PRELOAD MOVEABLE DISCARDABLE
Data PRELOAD MOVEABLE SINGLE
Heapsize 4096
Stacksize 5120
Stub 'WINSTUB.EXE'
```

4

Porting Without Pain

WHAT'S IN THIS CHAPTER

Borland and Microsoft both supply tools that allow you to directly port many text-based DOS programs directly to Windows. (Microsoft's tools will even support some graphics programs.) This chapter will show you how to get the most from these tools. It also includes a similar library, TWIN.

PREREQUISITES

You should understand C programming. Also, an understanding of some basic Windows concepts will help you get the most from this chapter.

Many Windows programs started life as DOS programs. As the Windows market has grown, more DOS programs have been ported to Windows. Recognizing this fact, several vendors supply tools that allow you to compile and run text-based DOS programs as Windows applications. Many programs will not require any changes to their source code.

Of course, your program will still look like an ordinary DOS program. These toolkits create a window that acts like a terminal attached to the stdin and stdout streams of your program. Microsoft's toolkit can also create windows that support Microsoft graphics calls.

Why Not a DOS Box?

You may wonder why you would want such a tool. Why not just run DOS programs in a DOS box? If you are in 386 enhanced mode, that approach might be viable. However, in standard or real mode, Windows can't multitask DOS programs. Also, DOS programs might not run in a window at all.

Of course, once you port your program using a DOS emulation toolkit, you can begin to enhance the program for Windows. For example, Microsoft's toolkit (QuickWin) allows you to add multiple windows to your programs.

These toolkits are mainly for porting DOS programs. However, you may sometimes write original programs with them. If the program you are writing needs (or could use) a DOS "look and feel," you might as well use

the toolkit and save the trouble of writing a real Windows program.

Available Tools

Both Borland and Microsoft supply DOS-emulation toolkits with their C compilers. In addition, this chapter contains the complete source code for an emulation toolkit named TWIN. Unlike its commercial counterparts, TWIN supports both compilers and has some capabilities the others lack (printing, for example).

Currently, none of these tools work with Windows NT, although there is no reason why they could not. It seems likely that Borland and Microsoft will soon support EZWIN and QuickWin under Windows NT. TWIN would be reasonably simple to port. The code that uses Catch() and Throw() would have to change. Of course, you could port DOS programs to NT by using the console I/O API, which allows text-based programs to run under NT.

Borland

Borland's EZWIN is the most barebones toolkit of the three. It allows DOS programs to compile and run, but that is all. You can't easily add any window functionality such as menus to an EZWIN program, although you can directly call the Windows API.

When you instruct Borland's linker to create a Windows executable, the linker checks for a WinMain() function in

your code. If WinMain() is not present, the linker assumes the program is an EZWIN program.

Listing 1-3 (in Chapter 1) shows EZHOWDY.C. You could compile this with the Borland compiler for DOS using the following command:

```
bcc ezhowdy.c
```

For EZWIN, just use:

```
bcc -W ezhowdy.c
```

That's all you need to know about EZWIN. Like the other products in this chapter, it won't port arcane DOS programs that hook interrupts or directly write to the screen. EZWIN doesn't support graphics.

Microsoft

Microsoft's QuickWin library is similar to EZWIN in its basic form. You supply the /Mq switch when you compile a QuickWin program. If all you want is basic DOS emulation, that's all you need to do.

Unlike the other toolkits in this chapter, QuickWin does support graphic output. You can draw to a graphic window with the same calls Microsoft supplies for DOS programs in GRAPH.H.

QuickWin becomes more interesting when you want to extend the functionality of your program. QuickWin allows you to use special calls to create multiple win-

dows of arbitrary size. You will see a QuickWin example later in this chapter. Microsoft does not allow you to directly call the Windows API from a QuickWin program. QuickWin programs have a default menu that you can't change. If your program runs other programs (via exec() or system(), directly accesses the screen, or does BIOS I/O, you must change these practices before your program will work with QuickWin.

TWIN

TWIN is not quite as transparent as the commercial products. You need to include TWIN.H in your program, and you may have to set a few constants (using #define) to configure your program.

In return for this small amount of additional work, TWIN allows you to create screens of any size and use custom menus and icons. The default window size is 80 columns by 25 rows, but you can easily create one window that is 40x10 and another that is 40x100. You also can use a default menu or create one of your own, and you can easily extend TWIN programs by using the Windows API.

Advanced QuickWin Features

QuickWin programs can create additional windows, set the text for the about box, and control window sizes and behavior. Table 4-1 shows the available QuickWin functions (see the Microsoft documentation for specifics).

Table 4-1. QuickWin Functions

Function	Purpose
_wopen()	Opens a text window—returns a file handle
_fwopen()	Opens a text window—returns a FILE pointer
_wgopen()*	Opens a graphic window
_wclose()	Closes a text window
_wgclose()*	Closes a graphic window
_wsetexit()	Sets exit behavior
_wgetexit()	Gets exit behavior
_wsetsize()	Sets text window's position and size
_wgetsize()	Gets window's position and size
_wsetscreenbuf()	Sets text window's screen buffer size
_wgetscreenbuf()	Gets text window's screen buffer size
_wsetfocus()	Makes window active
_wgetfocus()	Gets current active window
_wgsetactive()*	Sets the active graphics window
_wggetactive()*	Gets the active graphics window
_wmenuclick()	Simulates menu action
_wyield()	Yields processing time to other applications
_wabout()	Sets custom message for about box
_inchar()*	Reads a character in a graphic window

**Graphics-only functions*

Each QuickWin program starts with a default window that represents the program's stdout and stderr streams. Additional windows appear to be separate files. You can

use the _fwopen() function to create a FILE stream (for use with fprintf(), etc.). If you prefer, you can create a window with an unbuffered file handle with _wopen(). The file handle is for use with functions like write().

Opening Windows

Both _fwopen() and _wopen() take _wopeninfo and _wsizeinfo structures as arguments. You can set either of these fields to NULL to get default values for these structures. Both structures have a _version field that you must set to _QWINVER. The _wopeninfo structure also contains the window's title, and the size of the window's buffer (in bytes). This size determines how much text the window can hold. If you like, you can use _WINBUFDEF for the default size or _WINBUFINF if you don't want a limit. The _wsizeinfo structure contains information about the window's size.

Closing Windows

You can close a QuickWin window by passing its file handle to the _wclose() function. You also pass a constant to _wclose() that determines how to close the window. If you use _WINPERSIST, the window will remain visible. Although your program can't operate on the window, the user can examine the contents of the window (scrolling if necessary) and use the window's menus. If you call _wclose() with _WINNOPERSIST, the window will simply disappear.

By default, when your program exits, its windows remain on the screen until the user chooses to exit from the

menu. You can change this behavior by calling _wsetexit(_WINEXITNOPERSIST).

Behaving Under Windows

While your QuickWin program is running, other Windows programs may not get a chance to execute until your program stops for input. To remedy this, you may want to call _wyield() during processing. This will allow other Windows applications to share the processor with you.

If you want to yield inside a loop, you may not want to call _wyield() on each iteration since this may slow your processing. Often, you will only yield after the loop has run a certain number of times. For example:

```
while (work_to_do)
  {
  static int yield_ctr=10;
  if (!--yield_ctr)
    {
    yield_ctr=10;
    _wyield();
    }
  /* do your thing here */
  }
```

Split-Personality Programs

By default, QuickWin provides a DEF file (see Chapter 2) for your program. However, you can provide your own DEF file if you wish. Just copy Microsoft's CL.DEF to

another file and modify it. Then, you supply the name of your DEF file to the compiler. This process is useful if you want to bind a DOS and Windows program together. Consider the EZHOWDY program (Listing 1-3). This program will compile under QuickWin or you can compile it for DOS. Suppose you compile EZHOWDY for DOS and rename the EXE file to DOSHOW.EXE. You can replace the line in your DEF file that contains:

```
STUB 'WINSTUB.EXE'
```

to:

```
STUB 'DOSHOW.EXE'
```

After you compile and link EZHOWDY with QuickWin, your EZHOWDY.EXE file will have an interesting property. If you run it from Windows, you'll run the QuickWin version. Running EZHOWDY from DOS, however, will cause the DOS version (which resides inside EZHOWDY.EXE) to execute. Of course, you can do this with any Windows and DOS program, not just QuickWin programs.

QuickWin Graphics

Graphics windows are similar to text windows except that you create them with _wgopen(), which always returns a file handle. Before you draw to the window, you need to pass the graphic window's file handle to the _wgsetactive() function. The standard Microsoft C graphics

routines will then draw to the window. You can close a graphic window with _wgclose(). If you don't create a graphic window, your output goes to a default window. QuickWin creates this default window when you make your first graphics call.

Be careful if you are porting an existing graphics program. Some calls work differently than their DOS counterparts. For example, QuickWin graphics only support left-to-right horizontal text.

QuickWin graphic windows are not as flexible as text windows. You can't programatically resize or position graphics windows, for example. Also, the window may be larger than the graphic workspace. This can lead to odd-looking displays since your graphics won't fill the window. The user can ask QuickWin to scale the graphics to fill the window, but this may distort the image.

Microsoft doesn't document this, but you can read and write standard Windows BMP files using QuickWin. The standard _getimage() and _putimage() calls will read and write bitmaps in a memory array. The memory array corresponds to a BMP file. However, QuickWin ignores the color palette information.

QuickWin Limitations

While QuickWin is very powerful, it does have some serious limitations. Your QuickWin programs can't call the Windows API, which could be a problem if QuickWin won't do something you need to do (like print, for example). You can't directly respond to the mouse or display your own menus.

QuickWin doesn't have good keyboard handling. You can't use functions like getch() to read single characters. (You can use _inchar() in graphics windows.) You can't even use kbhit() to check for an available keystroke. Programs that depend on trapping the break key won't work under QuickWin either.

Although QuickWin programs can allocate multiple megabytes of memory via Windows, single objects still can't be larger than 64K (unless you resort to huge pointers). Still, many DOS programs will gain increased memory capacity under QuickWin.

QuickWin windows appear to be file handles. However, don't take this analogy too seriously. Many file calls (like dup() or dup2(), for example) don't work with QuickWin pseudofile handles. If you want to redirect stdout to another window, you'll need to resort to something like this:

```
FILE temp;  /* Note: not pointer */
FILE *newwin=_fwopen(NULL,NULL,"w+");
temp=*stdout;
*stdout=*newwin;
printf("This goes to new window");
/* more code here */
*stdout=temp;
```

Although QuickWin has help, you can't customize it; it is only about the QuickWin user interface. Still, for many cases, QuickWin is a very simple way to write DOS-like programs (or port existing programs). If you need menus

or direct access to Windows, you might consider using the TWIN library instead.

A QuickWin Program

Any simple DOS program (like EZHOWDY.C, Listing 1-3) will compile and run under QuickWin with no changes. However, you can write more interesting programs by using the QuickWin extensions.

The QVIEW program (Listing 4-1) displays the AUTOEXEC.BAT and CONFIG.SYS files from the root directory of your C drive. In addition, it displays a simple graphic for no apparent reason. Since QVIEW doesn't want a standard output window, it closes the window using this line of code:

```
_wclose(_fileno(stdout),_WINNOPERSIST);
```

Figure 4-1 shows the QVIEW program in operation.

Using TWIN

TWIN supplies DOS-like functions for creating or porting text-based programs to Windows (see Figure 4-2). Although TWIN attempts to duplicate the look of some DOS functions, it extends some functions and is not as transparent as the other products examined in this chapter. However, it offers some unique capabilities.

Figure 4-1. The QVIEW Program

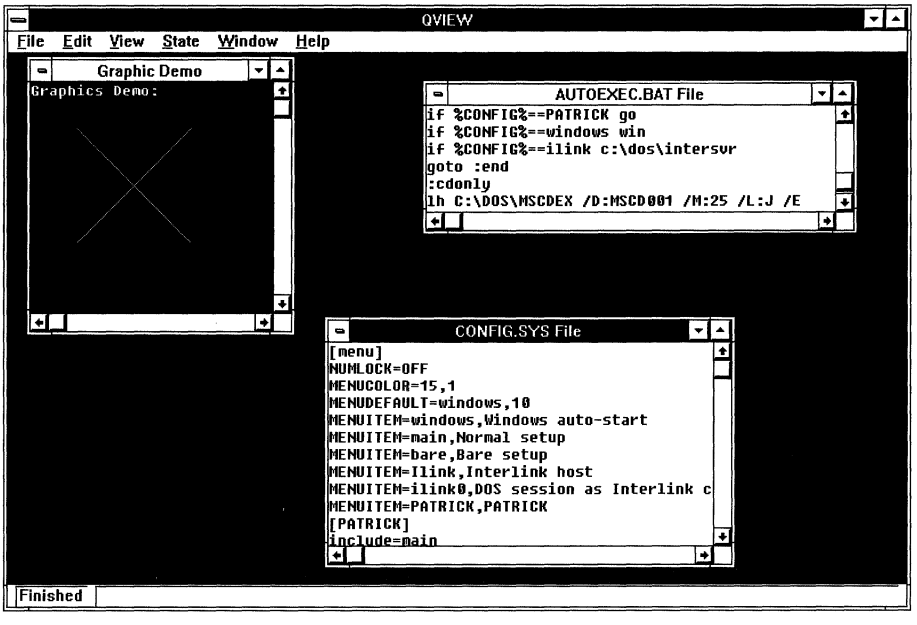


Figure 4-2. TWIN Calls

Note: many TWIN calls have aliases to standard library calls. For example, puts() maps to twin_puts().

```
void twin_create(TWIN_INFO *old,char *title,int wid,int hi);
```

Create TWIN window.

```
void twin_excreate(TWIN_INFO *old,char *title,DWORD style,
                   int x,int y,int wid,int hi,
                   HWND parent, HANDLE menu,int twid,int thi);
```

Create TWIN window—arguments same as CreateWindow().

(Cont.)

Figure 4-2. TWIN Calls (Cont.)

```
void twin_active(TWIN_INFO *old,TWIN_INFO *new);
```
Switch active window.

```
void twin_puts(char *s);
```
Write string to window.

```
void twin_putc(int c);
```
Write character to window.

```
void twin_goxy(int x,int y);
```
Set cursor position.

```
int twin_wherex(void);
```
Get cursor X position.

```
int twin_wherey(void);
```
Get cursor Y position.

```
void twin_show(void);
```
Update display (only useful if you print with TF_HOLD set).

```
int twin_fflush(FILE *s);
```
Aliased to fflush(). If s is stdout, this maps to twin_show(). Otherwise, twin_fflush() calls fflush().

```
void twin__putc(int c);
```
Write character without showing it (used internally).

```
void twin_cls(void);
```
Clear window.

(Cont.)

Figure 4-2. TWIN Calls (Cont.)

```
void twin_clreol(int x,int y);
```
Clear to end of line.

```
void twin_yield(void);
```
Yield time to Windows.

```
int twin_keyhit(void);
```
Check for keystrokes waiting.

```
int twin_getch(int *scan);
```
Get character and scan code.

```
int twin_getche(int *scan);
```
Get character and scan code and echo to current window.

```
int twin_setflag(int flagword);
```
Set TWIN flags.

```
int twin_gets(char *buf,unsigned int siz);
```
Get input string.

```
int twin_printf(char *fmt,...);
```
Printf-style output to window.

```
int twin_print(void);
```
Print current window to printer.

```
void twin_exit(int rv);
```
Exit TWIN program.

TWIN Configuration

TWIN programs must include TWIN.H (Listing 4-2) instead of STDIO.H. Before including this file, you'll need to set some defines that control TWIN's operation.

The source file that contains your main() function must define TWIN_MAIN before it includes TWIN.H. You can also define several constants to set window titles, default sizes, and so on (see Table 4-2).

Table 4-2. TWIN Configuration Definitions

Define	Optional?	Default Value	Description
TWIN_NAME	Yes	"TWINAPP"	Window class name
TWIN_TITLE	Yes	"TWIN Application"	Window title
TWIN_WIDTH	Yes	80	Default text buffer width
TWIN_HEIGHT	Yes	25	Default text buffer height
TWIN_ICON	Yes	""	Default program icon
TWIN_ABOUT	Yes	"A Twin App..."	About box text
TWIN_MENU	Yes	"TWIN_MENU_-DEFAULT"	Default menu's resource ID
TWIN_NOALIAS	Yes	None	If defined, do not provide aliases for printf(), etc.
TWIN_NOPRINT	Yes	None	Do not provide twin_print() and print menu.
NOEDITMENU	Yes	None	Pass to RC to prevent default EDIT menu.

Table 4-3. TWIN Flags

Flag	Meaning
TF_INCRXLAT	Translate input \r to \n
TF_OUTCRXLAT	Translate output \r to \n
TF_HOLD	Don't update display

The TWIN_WIDTH and TWIN_HEIGHT parameters determine the rows and columns of text—not the window's actual size. If the window is too small to display the text, TWIN will provide scroll bars as appropriate. The maximum size depends on how large an edit control can be. For Windows 3.1, the maximum width is 1,022 characters. The size of the buffer can't exceed 64K and will probably be less since the buffer must reside in your local heap.

Some TWIN options are set at runtime by flags (see Table 4-3). You modify TWIN flags by calling twin_setflag(). If the value you pass to twin_setflag() contains the value TF_SET or TF_RESET, the other flags you specify will be set or reset respectively. If you omit both TF_SET and TF_RESET, TWIN will use the flags you pass. For example:

```
twin_setflag(TF_INCRXLAT|TF_OUTCRXLAT);
```

will set the TF_INCRXLAT and TF_OUTCRXLAT flags and reset all the other flags. However:

```
twin_setflag(TF_SET|TF_HOLD);
```

will set the TF_HOLD flag and leave the other flags undisturbed.

TWIN Global Variables

You may find some of TWIN's global variables useful. If the twin_terminate variable is set when TWIN closes a window, it terminates the application. If you create and close multiple windows (you'll see how to do it soon), you'll want to set this variable to 0 (it equals 1 by default).

The twin_win variable contains a handle to the currently active window. The twin_edit variable contains a handle to the edit control that TWIN uses to display text. You'll only need these variables if you want to directly access the Windows API. You might also need the hINSTANCE variable, which contains the program's instance handle.

TWIN Menus

If you don't set the TWIN_MENU define, TWIN provides a default menu and a function, twin_menu(). However, you can supply your own menu and menu function, which must be named twin_menu(). When the user selects a menu item, TWIN will call twin_menu() pass, the command identifier as an argument.

Advanced Use of TWIN

TWIN supports multiple windows via the TWIN_INFO data structure. When you create a new window (twin_create() or twin_excreate(), or switch to another window (twin_active()), you must supply a pointer to a

TWIN_INFO structure. This structure receives the data that applies to the current TWIN window. You can then pass the structure to the twin_active() function to reactivate the window later. All TWIN functions act on the current active window.

The twin_create() function allows you to create a window by specifying its title, text width, and text height (the window may be smaller, in which case TWIN provides scroll bars). With the twin_excreate() function, you can specify most of the same parameters you use for the normal CreateWindow() call.

Listing 4-3 contains TMENU.C—a simple menu program that uses TWIN for Windows or Borland C for DOS. To compile it for DOS, use this command:

```
bcc -v TMENU.C
```

For Windows, use:

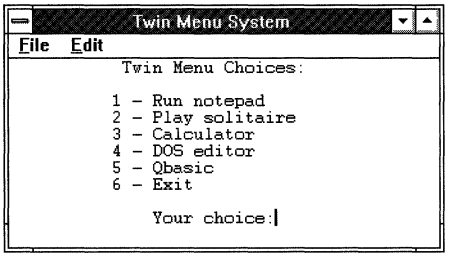
```
bcc -v -W -DWIN TMENU.C TWIN.C
rc TWIN.RC TMENU.EXE
```

or:

```
cl -Zi -GA -DWIN TMENU.C TWIN.C TWIN.DEF
rc TWIN.RC TMENU.EXE
```

You'll find TWIN.C, TWIN.RC, and TWIN.DEF in Listings 4-4, 4-5, and 4-6.

Figure 4-3. TMENU



Notice that you can use normal Windows calls like MessageBox() inside TWIN programs. The initial window could have been a MessageBox() dialog, but instead, TMENU uses an alternate TWIN window.

Of course, most of the DOS-specific code could have been left in place. The DOS exit-confirmation code, for example, would work equally well under TWIN. However, the MessageBox() dialog looks better, and was easy to add.

Figure 4-3 shows TMENU in operation. Notice that the EDIT menu is not very appropriate for an application like TMENU. If you don't want the EDIT menu to appear, use this RC command:

```
rc -DNOEDITMENU TWIN.RC TMENU.RC
```

How TWIN Works

TWIN subclasses the standard Windows edit control to provide text manipulation services. After TWIN creates the edit control, it

sets the text buffer to a private memory area allocated with LocalAlloc(). TWIN sets the control to use a fixed pitch font and limits its size to the exact size of the TWIN window.

The TWIN edit-control subclass intercepts all WM_CHAR messages. It stores the keys in the kqueue array so that twin_getch() can return them. Since TWIN doesn't send the WM_CHAR messages to the edit control, the user can't directly modify it. TWIN forwards all other messages to the original edit-control function.

Each TWIN window consists of a main window (twin_win) and a child subclassed edit control (twin_edit). TWIN stores the child window handle as a window property of the parent window. Given the main window, TWIN can find the child edit window by using:

```
twin_edit=GetProp(twin_win,"EDIT_CHILD");
```

Of course, you can reverse the operation with GetParent():

```
twin_win=GetParent(twin_edit);
```

With the edit-control handling scroll bars, clipboard actions, and text drawing, all of TWIN's functions become very simple. By modifying the private buffer, TWIN can alter what appears in the control.

Summary

Although these toolkits are for porting DOS programs, don't ignore them for writing simple Windows programs from scratch. EZWIN is simplistic, but QuickWin and TWIN can handle moderately sophisticated programs. QuickWin can even do graphics. (Be sure to look at VWIN in Chapter 7 for another simple graphics technique.)

Listing 4-1. QVIEW.C

```
/*********************************************************
 *                                                       *
 * File: QVIEW.C                                         *
 *                                                       *
 * Microsoft QuickWin file viewer.                       *
 *                                                       *
 * Required to Compile:                                  *
 * QVIEW.C, Microsoft C with QuickWin libraries.         *
 *                                                       *
 *********************************************************/
#include <stdio.h>
#include <graph.h>
#include <io.h>

/* Open window structures */
struct _wopeninfo oinfo_a =
  {
  _QWINVER, "AUTOEXEC.BAT File", _WINBUFINF
  };

struct _wopeninfo oinfo_c =
  {
  _QWINVER, "CONFIG.SYS File", _WINBUFINF
  };


main()
  {
  FILE *autoexec, *awin;
```

```
FILE *config, *cwin;
  int gwin;
  char line[1024];
  /* Close stdout window */
  _wclose(_fileno(stdout), _WINNOPERSIST);
  /* Close default graphics window */
  _wgclose(_wggetactive());
  /* Set about text */
  _wabout("QVIEW system file viewer by Al Williams");
  /* Draw graphics for no good reason */
  gwin = _wgopen("Graphic Demo");
  _wgsetactive(gwin);
  _setvideomode(_MAXRESMODE);
  _outtext("Graphics Demo:");
  _moveto(40, 40);
  _lineto(140, 140);
  _moveto(140, 40);
  _lineto(40, 140);
  /* open window for autoexec and config.sys */
  awin = _fwopen(&oinfo_a, NULL, "w+");
  cwin = _fwopen(&oinfo_c, NULL, "w+");
  /* open files */
  autoexec = fopen("c:\\autoexec.bat", "r");
  config = fopen("c:\\config.sys", "r");

  /* print text to correct window */
  while (fgets(line, sizeof(line), autoexec))
    fprintf(awin, "%s", line);

  while (fgets(line, sizeof(line), config))
```

```
    fprintf(cwin, "%s", line);

  /* Make sure windows hang around (this is the default
   * anyway) */
  _wsetexit(_WINEXITPERSIST);
  /* We're done! */
  exit(0);
  }
```

Listing 4-2. TWIN.H

```
/*********************************************************
 *                                                       *
 * File: TWIN.H                                          *
 *                                                       *
 * Header for TWIN.C                                     *
 *                                                       *
 * Required to Compile:                                  *
 * TWIN.C TWIN.H TWIN.RC TWIN.DEF                        *
 *                                                       *
 *********************************************************/
#ifndef TWIN_HEADER
#define TWIN_HEADER

#include <windows.h>
#ifndef RC
#include <stdio.h>
#endif

/* TWIN Flags */
#define TF_INCRXLAT 1       /* translate input CR to \n */
```

```
#define TF_OUTCRXLAT 2      /* translate output \n to
              * \r\n */
#define TF_HOLD 4     /* don't show output yet */
#define TF_RESET 0x8000     /* reset bits specified */
#define TF_SET 0x4000       /* set bits specified */

/* TWIN data */
typedef struct
  {
int bufsize;
  HWND twin_win;
  int editdirty;
  int twin_flags;
  HANDLE bstr;
  HANDLE editbuf;
  int cursoroff;
  int twin_width;
  int twin_height;
  }       TWIN_INFO;


/* Global variables for current window */
extern HWND twin_edit;
extern HWND twin_win;
extern int twin_terminate;
extern HINSTANCE hINSTANCE;

void twin_puts(char *s);
void twin_putc(int c);
void twin_goxy(int x, int y);
```

```
int twin_wherex(void);
int twin_wherey(void);
void twin_show(void);
int twin_fflush(FILE * s);
void twin__putc(int c);
void twin_cls(void);
void twin_clreol(int x, int y);
void twin_yield(void);
int twin_keyhit(void);
int twin_getch(int *scan);
int twin_getche(int *scan);
int twin_setflag(int flagword);
int twin_gets(char *buf, unsigned int siz);
int twin_printf(char *fmt,...);
int twin_print(void);
void twin_exit(int rv);
void twin_create(TWIN_INFO * old, char *title, int wid,
                 int hi);
void twin_excreate(TWIN_INFO * old, char *title,
              DWORD style, int x, int y, int wid, int hi,
              HWND parent, HANDLE menu, int twid, int thi);
     void twin_active(TWIN_INFO * old, TWIN_INFO * new);
     int twin_main(HWND w);
     int twin_menu(int menuitem);


#if defined(TWIN_MAINMODULE)||!defined(TWIN_MAIN)
/* Non-main module extern's */
     extern char twin_name[];
     extern char twin_title[];
```

```
      extern int twin_width;
      extern int twin_height;
      extern char twin_menuname[];
      extern char twin_about[];
      extern char twin_icon[];
#else
/* Main module, declare variables with defaults */
char twin_name[] =
#ifdef TWIN_NAME
      TWIN_NAME;
#else
      "TWINAPP";
#endif
      char twin_title[] =
#ifdef TWIN_TITLE
      TWIN_TITLE;
#else
      "TWIN Application";
#endif
      int twin_width =
#ifdef TWIN_WIDTH
      TWIN_WIDTH;
#else
      80;
#endif
      int twin_height =
#ifdef TWIN_HEIGHT
      TWIN_HEIGHT;
#else
      25;
```

```
#endif

     char twin_icon[] =
#ifdef TWIN_ICON
     TWIN_ICON;
#else
"";
#endif

     char twin_about[] =
#ifdef TWIN_ABOUT
     TWIN_ABOUT;
#else
     "A Twin Application -- TWIN by Al Williams";
#endif

     char twin_menuname[] =
#ifdef TWIN_MENU
     TWIN_MENU;
#else
     "TWIN_MENU_DEFAULT";

/* Default menu handler */
     int twin_menu(int cmd)
  {
  if (cmd == 101)
       {
       DestroyWindow(twin_win);
       }
  else if (cmd == 102)
```

```
        {
        MessageBox(twin_win, twin_about, "About",
         MB_OK | MB_ICONINFORMATION);
        }
   else if (cmd == 103)
{
        SendMessage(twin_edit, WM_COPY, 0, 0);
        }
   else if (cmd == 104)
        {
        SendMessage(twin_edit, EM_SETSEL, 0,
                    MAKELONG(0, 32767));
        }
   else if (cmd == 105)
        {
        twin_print();
        }
   return 1;
   }

#endif


#endif

#ifndef TWIN_NOALIAS
#define clrscr() twin_cls()
#define goxy(x,y) twin_goxy(x,y)
#define wherey() twin_wherey()
#define wherex() twin_wherex()
```

```
#define clreol(x,y) twin_clreol(x,y)
#define fflush(s) twin_show(s)
#undef putch
#define putch(c) twin_putc(c)
#define getche() twin_getche(NULL)
#define getch() twin_getch(NULL)
#define kbhit() twin_keyhit()
#define gets(s) twin_gets(s,0xFFFF)
#define printf twin_printf
#define exit(a) twin_exit(a)
#define main twin_main
#endif

/* End of header protection if */
#endif
```

Listing 4-3. TMENU.C

```
/*****************************************************
 *                                                   *
 * File: TMENU.C                                     *
 *                                                   *
 * Example menu program using TWIN.                  *
 * Use /DWIN to create Window version.               *
 * Without /DWIN, this will work with Borland C      *
 *                                                   *
 * Required to Compile:                              *
 * TMENU.C TWIN.C TWIN.H TMENU.RC TWIN.RC TWIN.DEF   *
 *                                                   *
 *****************************************************/
#ifdef WIN
```

```
/* TWIN configuration */
#define TWIN_NAME "TMENU"
#define TWIN_TITLE "Twin Menu System"
#define TWIN_ABOUT "This menu program demonstrates" \
  " the TWIN text windowing system. Select a choice by" \
  " pressing the number next to it."

#define TWIN_MAIN 1
#define TWIN_HEIGHT 10
#define TWIN_WIDTH 40
#include "twin.h"


#define WIDTH twin_width
#define HEIGHT twin_height
#else
/* DOS stuff */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define WIDTH 80
#define HEIGTH 25
#endif

#include <string.h>

/* menu items */
struct
  {
```

```
  char *item;
  char *action;
  } menu[] =
    {
      { "Run notepad", "Notepad\0" },
      { "Play solitaire", "Sol" },
      { "Calculator", "Calc" },
  /* NOTE: must have .COM extension for WinExec! */
      { "DOS editor", "Edit.com" },
      { "Qbasic", "Qbasic" },
      { "Exit", NULL }
    };

#define NRITEMS (sizeof(menu)/sizeof(menu[0]))

/* Print string in center */
void cenprint(char *s)
  {
  int width = (WIDTH - strlen(s)) / 2;
  printf("%*s%s", width, " ", s);
  }

#ifdef WIN
main(HWND w)
#else
main()
#endif
  {
  int i, key;
#ifdef WIN
```

```
  /* Demonstrate multiple windows under TWIN */
  TWIN_INFO stdwin, window2;
  twin_create(&stdwin, "New Window", 35, 2);
  printf("TWIN Menu system by Al Williams\n");
  printf("Press any key to start");
  getch();
  twin_terminate = 0;
  DestroyWindow(twin_win);
  twin_terminate = 1;
  twin_active(NULL, &stdwin);
#else
  printf("TWIN Menu system by Al Williams\n");
  printf("Press any key to start");
  if (!getch())
    getch();
#endif
  while (1)
      {
      clrscr();
      cenprint("Twin Menu Choices:");
      printf("\n\n");
      for (i = 0; i < NRITEMS; i++)
       {
       int width = WIDTH / 2 - 10;
       printf("%*s%1d - %s\n", width, " ", i + 1,
              menu[i].item);
       }
      printf("\n");
      cenprint("Your choice:");
      while ((key = getch()) < '1' || key > '9');
```

```
        putch(key);
        key = key - '1';
        if (!menu[key].action)
          {
#ifdef WIN
          if (MessageBox(twin_win, "Quit?", "Confirm",
                         MB_YESNO | MB_ICONSTOP) == IDYES)
            exit(0);
          else
            continue;
#else
          int c;
          printf("\nReally quit? (Y/N) ");
          do
            {
            c = getch();
            if (!c)
               getch();
            } while (c != 'y' && c != 'Y' &&
                     c != 'n' && c != 'N');
          if (c == 'y' || c == 'Y')
             exit(0);
          else
             continue;
#endif
          }
#ifdef WIN
        if (WinExec(menu[key].action, SW_SHOW) < 32)
           MessageBox(w, "Can't perform action",
                      NULL, MB_OK | MB_ICONSTOP);
```

```
#else
        system(menu[key].action);
#endif
        }
    }
```

Listing 4-4. TWIN.C

```
/*****************************************************
 *                                                   *
 * File: TWIN.C                                      *
 *                                                   *
 * Text emulation library.                           *
 *                                                   *
 * Required to Compile:                              *
 * TWIN.C TWIN.H TWIN.RC TWIN.DEF                    *
 * Plus your program                                 *
 *                                                   *
 * Use -DOLD on command line if you want Windows 3.0 *
 * printing instead of Windows 3.1+                  *
 *                                                   *
 *****************************************************/
#include <windows.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#ifndef OLDWIN
#include <commdlg.h>
#endif
#define TWIN_MAINMODULE 1
#define TWIN_NOALIAS 1
```

```
#include "twin.h"

#ifndef GET_WM_COMMAND_ID
#define GET_WM_COMMAND_ID(wp,lp) (wp)
#endif

#ifndef GET_WM_COMMAND_CMD
#define GET_WM_COMMAND_CMD(wp,lp) HIWORD(lp)
#endif



/* win_proc is the callback function (window proc) */
long FAR PASCAL _export win_proc(HWND, unsigned,
                        WPARAM, LONG);


/* Global variables */
/* edit window */
HWND twin_edit;
/* main window */
HWND twin_win;
/* flags */
int twin_flags = TF_INCRXLAT | TF_OUTCRXLAT;
/* instance handle */
HINSTANCE hINSTANCE;
/* terminate on close window? */
int twin_terminate = 1;
/* buffer is dirty */
static int editdirty;
```

```
/* Catch/throw buffer for exit() */
static CATCHBUF xitbuf;
/* value to return on exit() */
static int xitvalue;
/* string of blanks */
static HANDLE bstr;
static char *bstring;
/* edit buffer */
static HANDLE editbuf;
static char *editbufp;
static int bufsize;
/* offset to cursor */
static int cursoroff;

/* number of tickyield() calls before true yield */
#define FORCEYIELD (50)
static int yieldtick = FORCEYIELD;
/* Yield every FORCEYIELD times */
#define tickyield()  if (--yieldtick==0) \
          { twin_yield(); yieldtick=FORCEYIELD;   }
/* edit subclass procedure */
static FARPROC editproc;

/* Keyboard queue */
static unsigned int kqueue[16];
static volatile int head;
static volatile int tail;


/* Main function */
```

```
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdLine, int nCmdShow)
  {
  WNDCLASS wndClass;
  HWND hWnd;

  hINSTANCE = hInstance;
  if (!hPrevInstance)
      {
      /* register class */
      wndClass.style = CS_HREDRAW | CS_VREDRAW;
      wndClass.lpfnWndProc = (WNDPROC) win_proc;
      wndClass.cbClsExtra = 0;
      wndClass.cbWndExtra = 0;
      wndClass.hInstance = hInstance;
      if (*twin_icon)
         wndClass.hIcon = LoadIcon(hINSTANCE, twin_icon);
      else
         wndClass.hIcon = LoadIcon(NULL, IDI_ASTERISK);
      wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
      wndClass.hbrBackground =
         GetStockObject(WHITE_BRUSH);
      wndClass.lpszMenuName = twin_menuname;
      wndClass.lpszClassName = twin_name;

      if (!RegisterClass(&wndClass))
         return FALSE;
      }

  /* create twin window */
```

```
  twin_create(NULL, twin_title, twin_width, twin_height);
  /* Catch/Throw work like setjmp/longjmp. The exit()
   * function comes here */
  if (Catch(xitbuf))
    return xitvalue;


  /* call main function and return value */
  /* see twin_yield() for event loop! */
  return twin_main(hWnd);
  }


void twin_yield()
  {
  MSG m;
  yieldtick = FORCEYIELD;
  /* Just in case */
  if (InSendMessage())
    ReplyMessage(0);
  /* While messages are waiting, process them */
  while (PeekMessage(&m, NULL, 0, 0, PM_REMOVE))
      {
      TranslateMessage(&m);
      DispatchMessage(&m);
      /* if message was a quit, exit */
      if (m.message == WM_QUIT)
        {
        Throw(xitbuf, 1);
        }
```

```
        }
    /* No more messages, return to program */
    }


/* Create default twin window */
void twin_create(TWIN_INFO * old, char *title,
                 int wid, int hi)
    {
    twin_excreate(old, title, WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                  NULL, NULL, wid, hi);
    }

/* Extended twin window create */
void twin_excreate(TWIN_INFO * old, char *title,
                   DWORD style, int x, int y,
                   int wid, int hi, HWND parent,
                   HANDLE menu, int twid, int thi)
    {
    /* Store current window */
    twin_active(old, NULL);
    /* set up environment */
    twin_width = twid;
    twin_height = thi;
    editdirty = 0;
    twin_flags = TF_INCRXLAT | TF_OUTCRXLAT;
    cursoroff = 0;
    /* Create and show window */
    CreateWindow(twin_name, title,
```

```
            style, x, y,
            wid, hi, parent, menu,
            hINSTANCE, NULL);
    ShowWindow(twin_win, SW_SHOW);
    UpdateWindow(twin_win);
    }

/* Pass activation between two twin windows. The old pointer
 * gets the old window data and the new pointer contains
 * the new window data. Either pointer may be NULL. */
void twin_active(TWIN_INFO * old, TWIN_INFO * new)
    {
    if (old)
        {
        old->bufsize = bufsize;
        old->twin_width = twin_width;
        old->twin_height = twin_height;
        old->editdirty = editdirty;
        old->twin_flags = twin_flags;
        old->bstr = bstr;
        old->editbuf = editbuf;
        old->cursoroff = cursoroff;
        old->twin_win = twin_win;
        }
    if (new)
        {
        bufsize = new->bufsize;
        twin_width = new->twin_width;
        twin_height = new->twin_height;
        editdirty = new->editdirty;
```

```
        twin_flags = new->twin_flags;
        bstr = new->bstr;
        editbuf = new->editbuf;
        cursoroff = new->cursoroff;
        twin_win = new->twin_win;
        twin_edit = GetProp(twin_win, "EDIT_CHILD");
        SetFocus(twin_edit);
        }
    }


/* Twin out of memory error */
static nomem(HWND hWnd)
    {
    MessageBox(hWnd, "Out of memory", NULL,
          MB_OK | MB_ICONHAND | MB_SYSTEMMODAL);
    PostQuitMessage(1);
    return 0;
    }


/* New edit control window procedure. This function makes a
 * subclass of the existing edit control */
long FAR PASCAL _export newedproc(HWND hWnd,
        unsigned Message,
        WPARAM wParam, LONG lParam)
    {
    int i;
    /* on WM_CHAR, stuff keystroke in kqueue */
    if (Message == WM_CHAR)
```

```
        {
        i = LOWORD(lParam);
        while (i--)
          {
          if ((tail + 1) & 15 == head)
            {
            MessageBeep(0);    /* Queue full! */
            break;
            }
          kqueue[tail] = wParam | ((lParam & 0xff0000) >> 8);
          tail = (tail + 1) & 15;
          }
        return 0;
        }
    return CallWindowProc((FARPROC) editproc, hWnd, Message,
           wParam, lParam);
    }


long FAR PASCAL _export win_proc(HWND hWnd,
        unsigned Message, WPARAM wParam,
        LONG lParam)
    {
    static FARPROC ep;
    static POINT maxxy;
    HANDLE tmp;
    switch (Message)
        {
        RECT r;
        /* Menu pick or edit memory error */
```

```
  case WM_COMMAND:
    if (GET_WM_COMMAND_CMD(wParam, lParam)
        == EN_ERRSPACE &&
        GET_WM_COMMAND_ID(wParam, lParam) == 1)
      {
      nomem(hWnd);
      return 0;
      }
    twin_menu(GET_WM_COMMAND_ID(wParam, lParam));
    return 0;

    /* Set minimum/maximum size */
  case WM_GETMINMAXINFO:
      {
      POINT pt;
      LPPOINT rgpt = (LPPOINT) lParam;
      HDC hDC;
      HFONT font;
      long tmp;
/* Get screen window since ours might not be around
 * yet */
      hDC = GetDC(GetDesktopWindow());
/* Set fixed font */
      font = GetStockObject(ANSI_FIXED_FONT);
      font = SelectObject(hDC, font);
/* How big is X? */
      tmp = GetTextExtent(hDC, "X", 1);
      pt = MAKEPOINT(tmp);
/* Compute screen size */
      pt.x *= twin_width;
```

```
        pt.y *= twin_height;
    /* add overhead for menu, scroll bars, etc. */
        maxxy.x = pt.x += GetSystemMetrics(SM_CXVSCROLL)
        + 2 * GetSystemMetrics(SM_CXFRAME);
        maxxy.y = pt.y += GetSystemMetrics(SM_CYMENU) +
        GetSystemMetrics(SM_CYHSCROLL) +
        GetSystemMetrics(SM_CYCAPTION) +
        2 * GetSystemMetrics(SM_CYFRAME);
    /* Don't let size exceed screen! */
        pt.x = min(pt.x, GetSystemMetrics(SM_CXSCREEN));
        pt.y = min(pt.y, GetSystemMetrics(SM_CYSCREEN));
        rgpt[1] = pt;
        rgpt[2].x = rgpt[2].y = 0;
        rgpt[4] = pt;
        SelectObject(hDC, font);
        ReleaseDC(GetDesktopWindow(), hDC);
        break;
        }

     case WM_CREATE:
       /* Set twin_win... can't do this:
        * twin_win=CreateWindow(....) since this will
        * execute with twin_win not set yet */
       twin_win = hWnd;
       /* compute size of buffer */
       bufsize = (twin_width + 2) * (twin_height - 1) +
                  twin_width;
       /* Allocate it */
       editbuf = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT,
             bufsize + 1);
```

```
bstr = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT,
   twin_width + 3);
bstring = LocalLock(bstr);
if (!editbuf || !bstring || !bstr)
   nomem(hWnd);
/* set buffer to spaces with CRLF ending */
memset(bstring + 2, ' ', twin_width);
bstring[0] = '\r';
bstring[1] = '\n';
LocalUnlock(bstr);
twin_cls();
cursoroff = 0;
GetClientRect(hWnd, &r);
/* Create window */
twin_edit = CreateWindow("edit", "",
      WS_CHILD | WS_HSCROLL | WS_VSCROLL |
      ES_AUTOHSCROLL | ES_AUTOVSCROLL |
      ES_MULTILINE | WS_VISIBLE
      ,0, 0, r.right, r.bottom, hWnd,
      (HMENU) 1, hINSTANCE, 0);
if (!twin_edit)
   nomem(hWnd);
/* Subclass edit control */
if (!ep)
   ep = MakeProcInstance((FARPROC) newedproc,
         hINSTANCE);
 editproc = (FARPROC) SetWindowLong(twin_edit,
          GWL_WNDPROC,
          (LONG) ep);
```

```
    /* Attach edit window handle to window as EDIT_CHILD
     * property */
    SetProp(twin_win, "EDIT_CHILD", twin_edit);
    tmp = SendMessage(twin_edit, EM_GETHANDLE, 0, 0L);
    /* Set control's buffer */
    SendMessage(twin_edit, EM_SETHANDLE, editbuf, 0L);
    /* Set text limit to the exact size needed */
    SendMessage(twin_edit, EM_LIMITTEXT, bufsize, 0L);
    if (tmp)
      LocalFree(tmp);
    /* Set fixed font */
    SendMessage(twin_edit, WM_SETFONT,
       GetStockObject(ANSI_FIXED_FONT), (LONG) TRUE);
    break;

    /* If window gets focus, find its EDIT_CHILD property
     * and pass focus to it */
  case WM_SETFOCUS:
      {
      HWND w = GetProp(hWnd, "EDIT_CHILD");
      if (w)
         SetFocus(w);
       break;
       }

    /* Resize... resize child edit window, too */
  case WM_SIZE:
      {
      RECT r;
      DWORD flag, old;
```

```
      HWND w = GetProp(hWnd, "EDIT_CHILD");
/* But not if iconic */
      if (!w || wParam == SIZEICONIC)
         break;
      GetWindowRect(hWnd, &r);
/* Compute if scroll bars needed */
      flag = GetWindowLong(w, GWL_STYLE);
      if (maxxy.x <= r.right - r.left)
         flag &= ~WS_HSCROLL;
      else
         flag |= WS_HSCROLL;
      if (maxxy.y <= r.bottom - r.top)
         flag &= ~WS_VSCROLL;
      else
         flag |= WS_VSCROLL;
      SetWindowLong(w, GWL_STYLE, flag);
/* Make scroll bars appear or disappear */
      ShowWindow(w, SW_MINIMIZE);
      ShowWindow(w, SW_RESTORE);
      MoveWindow(w, 0, 0, LOWORD(lParam),
                  HIWORD(lParam), TRUE);
/* Force text to appear in proper place */
      old = SendMessage(w, EM_GETSEL, 0, 0);
      SendMessage(w, EM_SETSEL, 0, 0);
      SendMessage(w, EM_SETSEL, 0, old);
      }
   break;


 case WM_CLOSE:
```

```
      if (!twin_terminate)
         DestroyWindow(hWnd);
      else
         twin_exit(1);
      break;

    case WM_DESTROY:
      DestroyWindow(GetProp(hWnd, "EDIT_CHILD"));
      RemoveProp(hWnd, "EDIT_CHILD");
      if (bstr)
        {
        LocalFree(bstr);
        bstr = 0;
        }
      break;

    default:
      return DefWindowProc(hWnd, Message,
            wParam, lParam);
      }
  return 0;
  }



/* clear "screen" */
void twin_cls()
  {
  int i;
```

```
  editbufp = LocalLock(editbuf);
  if (!editbufp)
    nomem(hINSTANCE);
  memset(editbufp, ' ', bufsize);
  for (i = twin_width; i < bufsize; i += twin_width + 2)
      {
      editbufp[i] = '\r';
      editbufp[i + 1] = '\n';
      }
  LocalUnlock(editbuf);
  twin_goxy(0, 0);
  }

/* Set cursor position */
void twin_goxy(int x, int y)
  {
  cursoroff = y * (twin_width + 2) + x;
  }

/* read cursor's y position */
int twin_wherey()
  {
  return cursoroff / (twin_width + 2);
  }

/* read cursor's x position */
int twin_wherex()
  {
  return cursoroff -
    (cursoroff / (twin_width + 2)) * (twin_width + 2);
```

```
  }

/* set twin flags */
int twin_setflag(int flagword)
  {
  int f = twin_flags;
  if (flagword & TF_RESET)
       {
       flagword = ~(flagword & ~TF_RESET);
       twin_flags &= flagword;
       }
  else if (flagword & TF_SET)
       {
       flagword &= ~TF_SET;
       twin_flags |= flagword;
       }
 else
    twin_flags = flagword;
  return f;
  }

/* clear to end of line from x,y position */
void twin_clreol(int x, int y)
  {
  int s = (twin_width + 2) * y + x;
  editbufp = LocalLock(editbuf);
  if (!editbufp)
    nomem(hINSTANCE);
  memset(editbufp + s, ' ', (y + 1) *
        (twin_width + 2) - 2 - s);
```

```
  LocalUnlock(editbuf);
  twin_show();
  }

/* alias for fflush(stdout) */
int twin_fflush(FILE * s)
  {
  if (s == stdout)
      {
      twin_show();
      return 0;
      }
  else
    return fflush(s);
  }


/* show changes to window (usually internal) */
void twin_show()
  {
  tickyield();
  if (twin_flags & TF_HOLD)
    return;
  if (editdirty)
      {
      InvalidateRect(twin_edit, NULL, 0);
      SendMessage(twin_edit, EM_SETSEL, 0,
        MAKELONG(cursoroff, cursoroff));
      editdirty = 0;
      }
```

```
  }

/* put character on screen */
void twin_putc(int c)
  {
  twin__putc(c);
  twin_show();
  }

/* put character on screen (no show) */
void twin__putc(int c)
  {
  int x, y;
  DWORD p;
  char s[2];
  tickyield();
  editdirty = 1;
  p = cursoroff;
  y = LOWORD(p) / (twin_width + 2);
  x = LOWORD(p) - y * (twin_width + 2);
  if (c != '\n' && c != '\r')
      {
      editbufp = LocalLock(editbuf);
      if (!editbufp)
        nomem(twin_win);
      editbufp[cursoroff] = c;
      x++;
      LocalUnlock(editbuf);
      }
  if (c == '\r')
```

```
    x = 0;
  if (c == '\n' || x > (twin_width - 1))
      {
      if (twin_flags & TF_OUTCRXLAT)
         x = 0;
      if (++y > twin_height - 1)
         {
         y = twin_height - 1;
     /* SCROLL SCREEN */
        editbufp = LocalLock(editbuf);
        if (!editbufp)
          nomem(twin_win);
        memcpy(editbufp, editbufp + twin_width + 2,
               bufsize - (twin_width + 2));
         bstring = LocalLock(bstr);
         memcpy(editbufp + (twin_height - 1) *
                (twin_width + 2) - 2,
                bstring, twin_width + 3);
          LocalUnlock(bstr);
          LocalUnlock(editbuf);
        }
      }
  cursoroff = x + y * (twin_width + 2);
  }

/* Put string to screen */
void twin_puts(char *s)
  {
  while (*s)
    twin__putc(*s++);
```

```
  twin_show();
  }

/* Get character with echo (scan code to int *) */
int twin_getche(int *scan)
  {
  int rc;
  rc = twin_getch(scan);
  twin_putc(rc);
  return rc;
  }

/* Get character without echo */
int twin_getch(int *scan)
  {
  int rc;
  if (editdirty)
    twin_show();
  while (head == tail)
    twin_yield();
  rc = kqueue[head];
  head = (head + 1) & 15;
  if (scan)
    *scan = (rc & 0xff00) >> 8;
  rc &= 0xff;
  if ((twin_flags & TF_INCRXLAT) && rc == '\r')
    rc = '\n';
  return rc;
  }
```

```
/* test for key in buffer */
int twin_keyhit()
  {
  tickyield();
  return head != tail;
  }


/* Get string */
int twin_gets(char *s, unsigned int siz)
  {
  int c, x, y, oflags;
  unsigned ct = 0;
  siz--;
  oflags = twin_setflag(TF_INCRXLAT | TF_OUTCRXLAT);
  while ((c = twin_getch(NULL)) != '\n' && c != '\x1b')
      {
      if (c == 8)
   //backspace
     {
     if (!ct)
         {
         MessageBeep(0);
         continue;
         }
     ct--;
     x = twin_wherex();
     y = twin_wherey();
     y = x ? y : y - 1;
     x = x ? x - 1 : twin_width - 1;
```

```
      twin_goxy(x, y);
      twin__putc(' ');
      twin_goxy(x, y);
      editdirty = 1;
      continue;
      }
      if (ct >= siz)
        {
        MessageBeep(0);
        continue;
        }
      if (!isprint(c))
        continue;
      twin__putc(c);
      editdirty = 1;
      s[ct++] = c;
      }
  if (c == '\n')
    twin_putc(c);
  s[ct] = '\0';
  twin_setflag(oflags);
  return c == '\n' ? ct : -1;
  }


/* buffer for printf() */
static char printbuf[1025];

/* printf to window */
int twin_printf(char *fmt,...)
```

```
  {
  int rc;
  va_list args;
  va_start(args, fmt);
  rc = vsprintf(printbuf, fmt, args);
  twin_puts(printbuf);
  va_end(args);
  return rc;
  }

static BOOL FAR PASCAL _export win_kill(HWND w, DWORD lp)
  {
  if (!w)
    return 0;
  DestroyWindow(w);
  return 1;
  }

/* exit program */
void twin_exit(int rv)
  {
  FARPROC fp;
  fp = MakeProcInstance((FARPROC) win_kill,
         hINSTANCE);
  EnumTaskWindows(GetCurrentTask(), fp, 0);
  FreeProcInstance(fp);
  xitvalue = rv;
  Throw(xitbuf, 1);
  }
```

```
#ifndef TWIN_NOPRINT
/* printing stuff */
static int print_abort;
static HWND print_dialog;

BOOL FAR PASCAL _export print_dlg(HWND dlg, WORD msg,
                        WPARAM wParam,
                        LONG lParam)
  {
  switch (msg)
      {
    case WM_INITDIALOG:
      /* turn off close */
      EnableMenuItem(GetSystemMenu(dlg, FALSE),
           SC_CLOSE, MF_GRAYED);
      return TRUE;

    case WM_COMMAND:
      /* Abort button! */
      print_abort = 1;
      EnableWindow(GetParent(dlg), TRUE);
      DestroyWindow(dlg);
      print_dialog = NULL;
      return TRUE;
      }
  return FALSE;
  }

BOOL FAR PASCAL _export abort_proc(HDC pcd, short code)
  {
```

```
  MSG msg;
  /* print abort proc */
  while (!print_abort &&
         PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    if (!print_dialog ||
        !IsDialogMessage(print_dialog, &msg))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }
  return !print_abort;
  }

/* print current buffer */
int twin_print()
  {
  int err = 0;
  char pname[81], *device, *driver, *output;
  FARPROC abort, printproc;
  TEXTMETRIC tm;
  HDC printer;
  short chary, lineperpage, totalpage, page,
        line, linenr = 0;
  /* Get printer info */
#ifdef OLDWIN
  GetProfileString("windows", "device", ",,,",
         pname, sizeof(pname));
  device = strtok(pname, ",");
  driver = strtok(NULL, ",");
  output = strtok(NULL, ",");
```

```
  /* Create print DC */
  if (device && driver && output)
    printer = CreateDC(driver, device, output, NULL);
  else
    return 1;
#else
  DOCINFO docinfo;
  PRINTDLG pdialog;
  memset(&pdialog, 0, sizeof(PRINTDLG));
  pdialog.lStructSize = sizeof(PRINTDLG);
  pdialog.Flags = PD_RETURNDC | PD_USEDEVMODECOPIES |
     PD_NOSELECTION | PD_NOPAGENUMS;
  if (!PrintDlg(&pdialog))
      {
      if (CommDlgExtendedError())
   MessageBox(NULL, "Can't Open Printer", NULL,
         MB_OK | MB_ICONSTOP);
      return 1;
      }
  printer = pdialog.hDC;
#endif
  /* compute metrics */
  GetTextMetrics(printer, &tm);
  chary = tm.tmHeight + tm.tmExternalLeading;
  lineperpage = GetDeviceCaps(printer, VERTRES) / chary;
  totalpage = (twin_height + lineperpage - 1) / lineperpage;
  EnableWindow(twin_win, FALSE);
  print_abort = 0;
  printproc = MakeProcInstance(print_dlg, hINSTANCE);
  print_dialog = CreateDialog(hINSTANCE, "PrintDialogBox",
```

```
                twin_edit, printproc);
  abort = MakeProcInstance(abort_proc, hINSTANCE);
#ifdef OLDWIN
  Escape(printer, SETABORTPROC, 0, (LPSTR) abort, NULL);
#else
  SetAbortProc(printer, abort);
#endif
  /* get text buffer */
  editbufp = LocalLock(editbuf);
  if (!editbufp)
    nomem(hINSTANCE);

  /* Do printing */
#ifdef OLDWIN
  if (Escape(printer, STARTDOC, 14,
        (LPSTR) "TWIN Print Job", NULL) > 0)
#else
  /* Win 3.1+ printing */
  docinfo.cbSize = sizeof(DOCINFO);
  docinfo.lpszDocName = "TWIN Print Job";
  docinfo.lpszOutput = NULL;
  if (StartDoc(printer, &docinfo) > 0)
#endif
    for (page = 0; page < totalpage; page++)
        {
#ifndef OLDWIN
        StartPage(printer);
#endif
        for (line = 0;
           line < lineperpage && linenr < twin_height;
```

```
            linenr++, line++)
            TextOut(printer, 0, chary * line,
               editbufp + line * (twin_width + 2),
               twin_width);

#ifdef OLDWIN
          if (Escape(printer, NEWFRAME,
                     NULL, NULL, NULL) < 0)
            {
            err = 1;
            break;
            }
#else
          EndPage(printer);
#endif
          if (print_abort)
            break;
          }
     else
        err = 1;
  /* Done */
     if (!err)
#ifdef OLDWIN
        Escape(printer, ENDDOC, 0, NULL, NULL);
#else
     EndDoc(printer);
#endif
     if (!print_abort)
       {
       EnableWindow(twin_win, TRUE);
```

```
      DestroyWindow(print_dialog);
      }
    if (err || print_abort)
    MessageBox(twin_win, "Can't Print", NULL,
               MB_OK | MB_ICONSTOP);
    FreeProcInstance(printproc);
    FreeProcInstance(abort);
    DeleteDC(printer);
    LocalUnlock(editbuf);
    return err | print_abort;
    }

#endif
```

Listing 4-5. TWIN.RC

```
/*****************************************************
 *                                                   *
 * File: TWIN.RC                                     *
 *                                                   *
 * Resources for TWIN programs.                      *
 *                                                   *
 * Required to Compile:                              *
 * TWIN.C TWIN.H TWIN.RC TWIN.DEF                    *
 *                                                   *
 *****************************************************/
#define RC
#include "twin.h"

TWIN_MENU_DEFAULT  MENU
BEGIN
```

```
  POPUP "&File"
  BEGIN
    MenuItem  "&About", 102
    MenuItem  "e&Xit", 101
  END
#ifndef NOEDITMENU
  POPUP "&Edit"
  BEGIN
    MenuItem  "&Copy", 103
    MenuItem  "&Select All", 104
#ifndef TWINNOPRINT
    MenuItem  "&Print",105
#endif
  END
#endif
END


#ifndef TWINNOPRINT
PrintDialogBox   DIALOG 100,100,120,40
  STYLE WS_POPUP|WS_SYSMENU|WS_VISIBLE|WS_DLGFRAME
BEGIN
  CTEXT "Now printing", -1, 4, 6, 120, 12
  DEFPUSHBUTTON "Cancel", IDCANCEL, 44,22,32,14
END

#endif
```

Listing 4-6. TWIN.DEF

```
NAME TWIN
DESCRIPTION 'by Al Williams'
CODE MOVEABLE
DATA MOVEABLE MULTIPLE
HEAPSIZE 2048
STACKSIZE 2048
```

5

Objects of Desire

WHAT'S IN THIS CHAPTER

This chapter familiarizes you with how to use object-oriented programming (OOP) libraries—particulary Borland's OWL—to simplify Windows programming.

PREREQUISITES

You should have a good understanding of ordinary C++ programming. Some understanding of conventional Windows SDK programming will also be helpful.

Object-oriented programming (OOP) can offer some relief from the complexities of Windows programming. However, OOP is no panacea; the simplicity in OOP programs depends on reusing existing classes. Although C++ is object oriented, it provides no classes useful for Windows programming. If you are intrepid, you can build your own class libraries and then benefit from OOP concepts in your future programs.

Of course, there is a better way. Many vendors offer class libraries designed to simplify Windows programming. Some of these class libraries are for Windows only; some allow you to write programs for Windows and other platforms.

Two of the most important class libraries are Borland's Object Window Library (OWL) and Microsoft's Foundation Classes (MFC). MFC (see Chapter 6) is more comprehensive than the current version of OWL, but it is also correspondingly more difficult to learn.

OWL uses C++ classes to represent windows, but OWL programs still require paint routines, window handles, and other Windows necessities. OWL's advantage comes from building on predefined classes. Borland supplies a complete text editor, for example, that you can easily extend to suit your needs. If you need something that Borland doesn't supply, you still have to resort to conventional Windows programming techniques. OWL and C++ help you organize your programs and reuse your code, but they don't do much to simplify Windows programming per se.

Constructing an Application

All OWL programs require you to define at least one class that represents your application (the application object). In its simplest form, the application object must only create the program's main window. Your application object inherits many things from its base class, TApplication. TApplication provides a default event loop, methods for running dialogs, and other useful functions. For an example of a simple application class see Listing 5-1.

The InitMainWindow() method creates the program's main window (by setting the MainWindow instance variable). This simple application uses a text-editor window (TFileWindow). However, many programs define their own subclass of a standard window class like TWindow or TFileWindow. Still, with the standard TFileWindow (and its default menu and accelerator), this simple application class provides a complete text-editing program (see TWEDIT.CPP and TWEDIT.RC in Listings 5-1, and 5-2). You can compile TWEDIT using the OWLCOMP batch file (Listing 5-3). Just type:

```
OWLCOMP TWEDIT
```

Make sure you replace the directory names in OWLCOMP.BAT with ones appropriate for your system.

OWL programs still require a WinMain() function. The WinMain() function creates an instance of the application object and calls its Run member function. Table 5-1

shows the most important member functions that TApplication provides.

Table 5-1. Important TApplication Member Functions and Variables

Member	Description
HAccTable	Handle to application accelerator table
MainWindow	Pointer to WindowsObject for main window
CanClose()	Returns TRUE if application can close
InitApplication()	Performs first-time initialization
InitInstance()	Performs instance initialization
InitMainWindow()	Constructs main window
MessageLoop()	Default message loop

OWL Windows

Each nondialog window in your program is an instance of TWindow or a subclass of TWindow. Usually, you have to create a new subclass of TWindow to provide useful windows. However, windows you create from the TFileWindow or TEditWindow classes are directly useful.

The SimpleApp class (see Listing 5-1) uses a TFileWindow to provide a functional text editor. If you need a special window type, you must create a new subclass of one of the existing classes.

Managing Resources

To attach a menu to an OWL window, call the window's AssignMenu() member function. You can pass Assign-

Menu() a resource ID or a string. You can also use accelerators if your application overrides TApplication's InitInstance() function. During InitInstance(), you have to load the accelerator table. For example:

```
HAccTable=LoadAccelerators(hInstance,"FileCommands");
```

OWL supplies default menus and accelerator tables for its predefined windows that need them (TFileWindow, for example).

You can also set custom icons for your program by overriding the GetWindowClass() method of your window class. The new method has to call the original method and set the hIcon field of the WNDCLASS structure that is passed to GetWindowClass(). For example:

```
void MyWindow::GetWindowClass(WNDCLASS& classinfo)
  {
  TWindow::GetWindowClass(classinfo);
  classinfo.hIcon=icon_handle;
  }
```

You can also use this method to change the window style, cursor, and background color.

Commando OWL Programming

From the commando point of view, OWL programming isn't much different from conventional Windows development.

Dialogs and menus still figure prominently in commando programming. The big advantage comes when you can start with a predefined class—either one that comes with OWL or one you have created in the past.

OWL uses its TDialog class to represent dialogs. When you use a TDialog object, you define a transfer buffer to go with it. The transfer buffer is a structure that represents the dialog controls that you want to read or write.

For example, consider a dialog with two edit controls:

```
ssndialog dialog 0, 0, 150, 150
style ws_child/ws_visible
begin
 edit text NAME.FIELD 39, 17, 95, 12
 edit text SS.FIELD 39, 36, 95, 12
          .
          .
          .
end
```

This dialog might have the following transfer buffer:

```
struct
 {
 char name[33];
 char ssn[12];
 } xferbuf;
```

You have to define the elements of the structure in the same order in which the controls appear in the RC file. Finally, you have to construct a subclass of the TDialog object to represent the dialog. In the new classes constructor, you have to create a control object for each control in the dialog and set the transfer buffer. Here is what the SSNDialog subclass looks like:

```
class SSNDialog : public TDialog
 {
 public:
 SSNDialog(PTWindowsObject parent,xferbuf *xfer);
 }

SSNDialog::SSNDialog(PTWindowsObject parent,
                              xferbuf *xfer) :
                 TDialog(parent,SSNDIALOG)
 {
 new TEdit(this,NAMEFIELD,33);
 new TEdit(this,SSFIELD,13);
 TransferBuffer=(void *)xfer;
 }
```

To create the dialog, you create a dialog object using new and call your application object's ExecDialog() method. For example:

```
SSNDialog *ssn=new SSNDialog(this,&xfer);
if (GetModule()->ExecDialog(ssn)==IDCANCEL) return;
```

Always create dialog objects with new. Never declare them on the stack as local variables. Modeless dialogs operate the same way except that you call the MakeWindow() function instead of ExecDialog().

When the dialog starts, each edit field will contain the strings that are in the corresponding transfer buffer fields. When the dialog exits, you can read the strings that were in the edit buffers by reading the transfer buffer.

By default, most control types work with transfer buffers. You can prevent a control from working with a transfer buffer by calling its DisableTransfer() function. Static text fields don't use transfer buffers by default. If you want to set a static element, you have to call its EnableTransfer() member function. For example, in the constructor, you can write:

```
(new TStatic(this,STATICFLD,"",10,10,100,30,40))
     ->EnableTransfer();
```

Your programs can use dialogs as main windows. Simply derive your main window class from TDialog. This is similar to the technique that the PHONE program in Chapter 3 uses.

Just as OWL supplies predefined windows, it also has several predefined dialogs that you can use. The TInputDialog class, for example, creates a simple line-input window (similar to the win_input() routine in Chapter 3). Another predefined class, the TFileDialog

dialog, allows you to select a filename. You'll see an example of this shortly.

Creating MDI Applications

You can use OWL to create multiple document interface (MDI) programs. Use the TMDIFrame class (or a subclass of it) to construct your main window. Your program's window (or windows) then becomes a child window of TMDIFrame. The example in this chapter illustrates the technique.

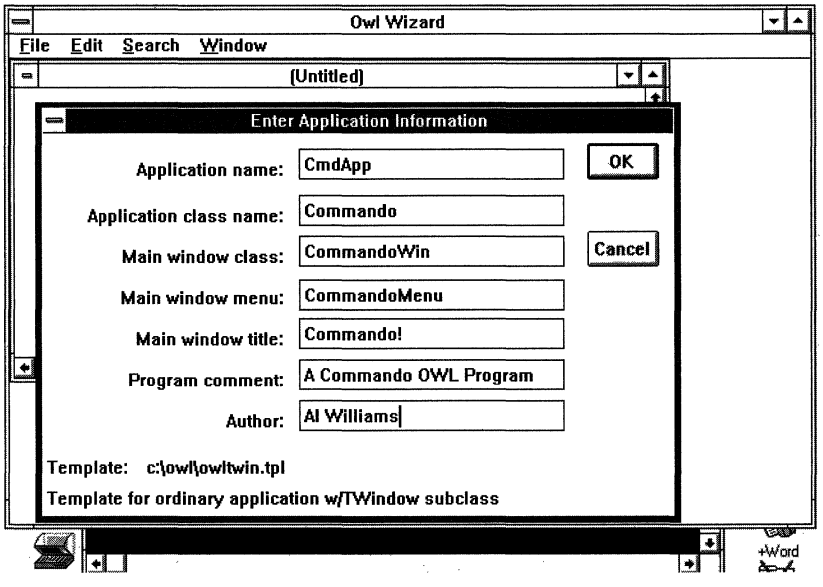
A Full OWL Application

OWLWIZ (Listings 5-4 to 5-6) is an MDI OWL application that also uses a modal dialog. OWLWIZ is a simple text editor that can also build the skeleton of an OWL application for you. When you ask OWLWIZ to create a new application, you must also select a template file. This file provides the basic structure of different types of OWL programs. The template contains references to variables that OWLWIZ will supply.

OWLWIZ then presents a dialog box so you can enter application information (see Figure 5-1 on the following page). OWLWIZ uses this input as the values of the template variables. Then OWLWIZ places the completed application in an edit window that you can save.

OWLWIZ defines five classes: OwlWiz, OwlWizWin, MDIFrame, WizDlg, and OwlTemplate. The OwlWiz class

Figure 5-1. OWLWIZ Dialog Box



is the required application class. MDIFrame is the MDI frame window. The OwlWizWin class is a subclass of the TFileWindow class. OwlWizWin is the main window class of OWLWIZ. WizDlg manages the OWLWIZ main input dialog. Finally, the OwlTemplate class reads template files and provides methods that replace variables with text.

OLWWIZ Templates

Listings 5-7 and 5-8 are sample OWL templates. The first line of the template is a comment; it doesn't appear in the

final output. Each subsequent line will appear in the output with its variables replaced by text. Each variable consists of three characters: two grave marks (") followed by a single character. Table 5-2 shows the available variables. The OwlTemplate class manages the templates. You can easily customize the supplied templates or create entirely new ones.

Table 5-2. OWLWIZ Variables

Variable	Description
"1	Application name
"2	Application class name
"3	Main window class
"4	Main window menu
"5	Main window title
"6	Program comment
"7	Author
"d	Current date/time

OWL Summary

While OWL offers some features of interest to the commando programmer, it isn't very useful unless you can use the predefined classes it supplies. Of course, if you are writing conventional Window programs anyway,

OWL can help you reuse the code you have already written.

This chapter is too short to cover OWL in its entirety. To learn more, check out the Borland OWL manuals.

Listing 5-1. TWEDIT.CPP

```
/*******************************************************
 *                                                     *
 * File: TWEDIT.CPP                                    *
 *                                                     *
 * Simple OWL text editor                              *
 *                                                     *
 * Required to Compile:                                *
 * TWEDIT.CPP TWEDIT.RC OWLCOMP.BAT                    *
 *                                                     *
 *******************************************************/
#include <owl.h>
#include <filewnd.h>

/* Application class */
 class SimpleApp : public TApplication
  {
  public:
  SimpleApp(LPSTR name,HINSTANCE hInst,
            HINSTANCE hPrev,
            LPSTR lpCmdLine, int nCmdShow)
            : TApplication(name,hInst,hPrev,
              lpCmdLine,nCmdShow) {};
  virtual void InitMainWindow();
  };

 // make file edit window
void SimpleApp::InitMainWindow()
  {
  MainWindow=new TFileWindow(NULL,"Simple editor","");
```

```
  ((TWindow *)MainWindow)->AssignMenu("FILECOMMANDS");
  }

int PASCAL WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
  {
  SimpleApp app("Simple Application",
                hInstance, hPrevInstance,
                lpCmdLine, nCmdShow);
  app.Run();
  return app.Status;
  }
```

Listing 5-2. TWEDIT.RC

```
/*******************************************************
 *                                                     *
 * File: TWEDIT.RC                                     *
 *                                                     *
 * Resources for simple text editor (all default       *
 * OWL resources).                                     *
 *                                                     *
 * Required to Compile:                                *
 * TWEDIT.CPP OWLCOMP.BAT                              *
 *                                                     *
 *******************************************************/
#include <windows.h>
#include <owlrc.h>
#include <filedial.dlg>
```

```
#include <filemenu.rc>
#include <fileacc.rc>
#include <stdwnds.dlg>
```

Listing 5-3. OWLCOMP.BAT

```
REM Compile OWL application with single source file
REM Change directory to match your system
SET BCDIR=d:\borlandc
IF NOT EXIST %1.cpp goto err
SET OWLINC=%BCDIR\include
SET OWLINC=%OWLINC%;%BCDIR%\classlib\include
SET OWLINC=%OWLINC%;%BCDIR%\owl\include
SET OWLLIB=%BCDIR\lib;%BCDIR%\classlib\lib;%BCDIR%\owl\lib
SET OWLLIBS=owlwl.lib tclassl.lib
SET STDLIBS=import mathwl cwl
SET LOPT=-Tw -v -n -x -c -C -L%OWLLIB%
REM Compile
bcc -ml -c -v -WE -DWIN31 -I%OWLINC% %1.cpp
IF ERRORLEVEL 1 goto cerr
echo %BCDIR%\LIB\cOwl.obj %1,%1, >OWLCOMP.TMP
echo %OWLLIBS% %STDLIBS% >>OWLCOMP.TMP
REM Link
tlink %LOPT% @OWLCOMP.TMP
IF ERRORLEVEL 1 goto cerr
REM RC
rc -r -DWIN31 -I%OWLINC% %1.rc %1.res
IF ERRORLEVEL 1 goto cerr
REM Link RC
rc %1.res %1.exe
IF ERRORLEVEL 1 goto cerr
```

```
goto end
:cerr
echo An error occured
goto end
:err
echo Usage: OWLCOMP filename
echo DO NOT include extension in filename argument
echo For example, to compile owlwiz.cpp use:
echo      OWCOMP owlwiz
:end
if EXIST OWLCOMP.TMP ERASE OWLCOMP.TMP
```

Listing 5-4. OWLWIZ.CPP

```
/*****************************************************
 *                                                   *
 * File: OWLWIZ.CPP                                  *
 *                                                   *
 * OWLWIZ creates skeletal OWL applications.         *
 *                                                   *
 * Required to Compile:                              *
 * OWLWIZ.H OWLWIZ.RC OWLCOMP.BAT                    *
 *                                                   *
 *****************************************************/
#include <combobox.h>
#include <filewnd.h>
#include <inputdia.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
```

```
#include <io.h>
#include <time.h>
#include "owlwiz.h"

#define NRFIELDS 7
#define VARLEN 81

class OwlWiz : public TApplication
   {
   public:
   OwlWiz(LPSTR name,HINSTANCE hInstance,
          HINSTANCE hPrevInstance,
          LPSTR lpCmdLine, int nCmdShow)
    : TApplication(name,hInstance,hPrevInstance,
       lpCmdLine,nCmdShow)
      {
      };
   virtual void InitMainWindow();
   virtual void InitInstance();
   };

/* Transfer buffer */
struct xferbuf
  {
  char vars[NRFIELDS][VARLEN]; /* in */
  char title[VARLEN];  /* 2 "outputs" */
  char comment[VARLEN];
  };

/* Main window (not MDI frame) */
```

```
class OwlWizWin : public TFileWindow
   {
   char templ[VARLEN];  // template
   xferbuf xfer;        // transfer to dialog
   public:
   OwlWizWin(PTWindowsObject AParent,
             LPSTR ATitle, LPSTR FileName);
   char *get_templ(void) { return templ; };
// create new application
   void newapp();
};

// MDI frame
class MDIFrame : public TMDIFrame
  {
  public:
  MDIFrame(LPSTR title) :
     TMDIFrame(title,"FILECOMMANDS") {};
  virtual PTWindowsObject CreateChild();
  virtual void NewFile(RTMessage Msg) =
                 [CM_FIRST + CM_MDIFILENEW];
  virtual void OpenFile(RTMessage Msg) =
                 [CM_FIRST + CM_MDIFILEOPEN];
  virtual void Close(RTMessage Msg) =
                 [CM_FIRST + CM_CLOSE];
  virtual void CMApplication(RTMessage Msg)=
                 [CM_FIRST+CM_APPL];
  };

void MDIFrame::NewFile(RTMessage)
```

```
  {
  GetApplication()->MakeWindow(new OwlWizWin
            (this,"Owl Wizard", ""));
  }

/* Respond to "Open" command by constructing,
   creating, and setting up a  new TFileWindow
   MDI child */
void MDIFrame::OpenFile(RTMessage)
  {
  char filename[MAXPATH];
  if ( GetApplication()->ExecDialog(
       new TFileDialog(this, SD_FILEOPEN,
       strcpy(filename, "*.*"))) == IDOK )
          GetApplication()->MakeWindow(new
              OwlWizWin(this,"",filename));
  }

/* Close current window (if any) */
void MDIFrame::Close(RTMessage)
  {
  if (ActiveChild)
   ActiveChild->CloseWindow();
  }


/* Create new child */
PTWindowsObject MDIFrame::CreateChild()
  {
  return GetApplication()->
```

```
    MakeWindow(new OwlWizWin(this,"",NULL));
  }


/* Template class */
class OwlTemplate
  {
FILE *fp;
// pointer to data (not comment)
  char *buffer;
// actual length of data in buffer
  unsigned long buflen;
// size of buffer
  unsigned bufsiz;
  int OwlTemplate::adjustbuf(int size);
// pointer to first line (comment) */
  char *first;
  public:
  OwlTemplate(char *filename);
  ~OwlTemplate()
    {
    if (buffer) free(buffer);
    if (fp) fclose(fp);
    };
  int status(void) { return fp!=NULL; };
  char *textbuffer(void) { return buffer; };
  char *firstline(void) { return first; };
/* Replace variable with string */
  int replace(int n,char *str);
  };
```

```
/* Dialog class */
class WizDlg : public TDialog
  {
  public:
WizDlg(PTWindowsObject AParent,
         char *comment,xferbuf *xfer);
  };



OwlWizWin::OwlWizWin(PTWindowsObject AParent,
           LPSTR ATitle, LPSTR FileName)
         : TFileWindow(AParent,ATitle,FileName)
   {
   }

/* New application:
   create window and call its newapp method */
void MDIFrame::CMApplication(RTMessage Msg)
  {
  OwlWizWin *w=(OwlWizWin *)CreateChild();
  w->newapp();
  }

void OwlWizWin::newapp()
  {
  PTInputDialog appnamedlg;
/* Check for dirty buffer and prompt
```

```
 This will never happen unless you add
 code that calls this for an existing
 edit control */
if (Editor->IsModified())
  {
  if (MessageBox(HWindow,
       "Overwrite current file?",
       "Confirm",MB_YESNO|MB_ICONQUESTION)
       !=IDYES) return;
  Editor->ClearModify();
  }

/* Select template */
strcpy(templ,"*.tpl");
TFileDialog *fd=new TFileDialog(this,SD_FILEOPEN,
                        templ);
fd->SetCaption("Select Application Template");
if (GetApplication()->ExecDialog(fd)!=IDOK) return;
/* Open template object */
OwlTemplate temp(templ);
if (!temp.status()) return;

/* Do dialog to get info */
WizDlg *dlg=new WizDlg(this,
                temp.firstline(),&xfer);
if (GetModule()->ExecDialog(dlg)==IDCANCEL)
    return;
/* Set date */
char *tstr;
time_t tm;
```

```
  time(&tm);
  tstr=ctime(&tm);
  tstr[24]='\0';  /* remove \n */
temp.replace('d',tstr);
  /* Set replaceable text */
  for (int i=0;i<NRFIELDS;i++)
     if (temp.replace(i+'0'+1,xfer.vars[i])<0) return;
/* Place in editor window */
  Editor->Insert(temp.textbuffer());
  if (IsNewFile)
    {
    char tmp[66];
    strcpy(tmp,xfer.vars[0]);
    strcat(tmp,".CPP");
    SetFileName(tmp);
    }
  }


void OwlWiz::InitMainWindow()
   {
   MainWindow=new MDIFrame("Owl Wizard");
   ((MDIFrame *)MainWindow)->ChildMenuPos=3;
   }

// Load accelerators
void OwlWiz::InitInstance()
   {
   TApplication::InitInstance();
   HAccTable=LoadAccelerators(hInstance,"FileCommands");
```

```
}

OwlTemplate::OwlTemplate(char *filename)
   {
/* open template file */
   fp=fopen(filename,"rb");
   if (fp)
     {
/* get memory */
     buflen=filelength(fileno(fp));
     buffer=NULL;
     if (buflen<0xF000L)
       {
       bufsiz=(unsigned int)buflen+128;
       buffer=(char *)malloc(bufsiz);
       }
     if (!buffer)
       {
       MessageBox(NULL,"Out of memory",
                  "Error",MB_OK|MB_ICONSTOP);
       fclose(fp);
       fp=NULL;
       }
     else
       {
       char *p;
/* read it all */
       fread(buffer,1,buflen,fp);
/* consume ^Z if present */
p=strchr(buffer,'\x1A');
```

```
        if (p)
          *p='\0';
        else
          buffer[buflen++]='\0';
/* point buffer past comment */
        first=buffer;
        buffer=strchr(buffer,'\n')+1;
        *(buffer-2)='\0';
        }
      }
    else
      {
      MessageBox(NULL,"Can't open template",
                 "Error",MB_OK|MB_ICONSTOP);
      }
    }

/* private method to resize buffer */
int OwlTemplate::adjustbuf(int size)
  {
  while ((buflen-3)+size>=bufsiz)
    {
    unsigned int bufoff=buffer-first;
    first=(char *)realloc(first,bufsiz+=128);
    if (!first)
      {
      MessageBox(NULL,"Out of memory",
                 NULL,MB_OK|MB_ICONSTOP);
                      return -1;
      }
```

```
    buffer=first+bufoff;
    }
  return 0;
  }


/* replace variable with text
  to replace variable ''x use:
  replace('x',"Text")
  if n==-1 put in front
  if n==-2 add to end */
int OwlTemplate::replace(int n,char *str)
  {
  char *point;
  char target[4];
  int replen=strlen(str);
  int ct=0;
  target[0]=target[1]='';
  target[2]=n;
  target[3]='\0';
  point=buffer;
/* if replacing with nothing, use blanks */
  if (!replen)
    {
    str="   ";
    replen=3;
    }
  if (n==-2) // add to end
    {
    /* make sure enough room */
```

```
    if (adjustbuf(buflen+replen)==-1) return -1;
    strcat(buffer,str);
    buflen+=replen;
    return 1;
    }
  if (n==-1) // add to begininning
    {
/*    make sure enough room */
    if (adjustbuf(buflen+replen)==-1) return -1;
    // move data in
    memmove(buffer+replen,buffer,buflen);
    memmove(buffer,str,replen);
    buflen+=replen;
    return 1;
    }
  while (point)
    {
    int offset;
    point=strstr(point,target);
    if (point)
      {
      ct++;
      if (replen<=3)
        {
/* short replacement -- fill with spaces */
        memcpy(point,str,replen);
        if (replen<3)
          memset(point+replen,' ',3-replen);
        continue;
        }
```

```
      offset=point-buffer;
      if (adjustbuf(buflen-3+replen)==-1)
        return -1;
/* Just in case realloc() moved buffer */
      point=buffer+offset;
      memmove(point+replen,point+3,
              buflen-(point-buffer)-3);
      memcpy(point,str,replen);
      buflen+=replen-3;
      }
    }
  return ct;
  }

/* Initial contents of dialog */
char *dlginit[]=
  {
  "OwlApp",
  "OwlApp",
  "OwlWindow",
  "MainMenu",
  "Main Window",
  "Another OWLWIZ Program",
  "Your Name Here"
  };
/* Dialog constructor */
WizDlg::WizDlg(PTWindowsObject parent,char *comment,
               xferbuf *xfer)
      : TDialog(parent,OWLWIZDLG)
  {
```

```
  new TEdit(this,APPID,VARLEN);
  new TEdit(this,APPNAME,VARLEN);
  new TEdit(this,WINDCLASS,VARLEN);
  new TEdit(this,MAINMENU,VARLEN);
  new TEdit(this,WINDTITLE,VARLEN);
  new TEdit(this,PROGCOM,VARLEN);
  new TEdit(this,AUTHOR,VARLEN);
// Init the two statics...
  new TStatic(this,TEMPNAME,VARLEN)->EnableTransfer();
  new TStatic(this,TEMPCOM,VARLEN)->EnableTransfer();
  for (int i=0;i<NRFIELDS;i++)
    strcpy(xfer->vars[i],dlginit[i]);
  strcpy(xfer->title,
     ((OwlWizWin *)parent)->get_templ());
  strcpy(xfer->comment,comment);
  TransferBuffer=(void *)xfer;
  }


int PASCAL WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
  LPSTR lpCmdLine, int nCmdShow)
  {
  OwlWiz OwlWizApp ("OwlWiz", hInstance,
    hPrevInstance, lpCmdLine, nCmdShow);
  OwlWizApp.Run();
  return OwlWizApp.Status;
  }
```

Listing 5-5. OWLWIZ.H

```
/*******************************************************
 *                                                     *
 * File: OWLWIZ.H                                      *
 *                                                     *
 * Constants for OWLWIZ.CPP                            *
 *                                                     *
 * Required to Compile:                                *
 * OWLWIZ.CPP OWLWIZ.RC OWLCOMP.BAT                    *
 *                                                     *
 *******************************************************/
#define CM_APPL 1
#define CM_CLOSE 2

#define OWLWIZDLG       101
#define TEMPNAME        102
#define TEMPCOM 103
#define APPID   109
#define APPNAME 110
#define WINDCLASS       111
#define MAINMENU        112
#define WINDTITLE       113
#define PROGCOM 114
#define AUTHOR  115
```

Listing 5-6. OWLWIZ.RC

```
/*******************************************************
 *                                                     *
 * File: OWLWIZ.RC                                     *
 *                                                     *
```

```
 * Resources for OWLWIZ                                          *
 *                                                               *
 * Required to Compile:                                          *
 * OWLWIZ.CPP OWLWIZ.H OWLCOMP.BAT                               *
 *                                                               *
 *****************************************************/
#include <windows.h>
#include <owlrc.h>
#include <filedial.dlg>
#include <stdwnds.dlg>
#include "owlwiz.h"

OWLWIZDLG DIALOG 10, 17, 248, 149
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Enter Application Information"
BEGIN
        CONTROL "OwlApp", APPID, "EDIT", ES_LEFT |
          ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE |
          WS_BORDER | WS_TABSTOP, 101, 6, 104, 12
        CONTROL "OwlApp", APPNAME, "EDIT", ES_LEFT
        | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_TABSTOP, 101, 24, 104, 12
        CONTROL "OwlWindow", WINDCLASS, "EDIT",
          ES_LEFT | ES_AUTOHSCROLL | WS_CHILD |
          WS_VISIBLE | WS_BORDER | WS_TABSTOP, 101,
          40, 104, 12
        CONTROL "MainMenu", MAINMENU, "EDIT",
          ES_LEFT | ES_AUTOHSCROLL | WS_CHILD |
          WS_VISIBLE | WS_BORDER | WS_TABSTOP, 101,
          57, 104, 12
```

```
CONTROL "Owl Window", WINDTITLE, "EDIT",
  ES_LEFT | ES_AUTOHSCROLL | WS_CHILD |
  WS_VISIBLE | WS_BORDER | WS_TABSTOP, 101,
  72, 104, 12
CONTROL "Program created by OWLWIZ",
  PROGCOM, "EDIT", ES_LEFT | ES_AUTOHSCROLL
  | WS_CHILD | WS_VISIBLE | WS_BORDER |
  WS_TABSTOP, 101, 88, 104, 12
CONTROL "Your Name Here", AUTHOR, "EDIT",
  ES_LEFT | ES_AUTOHSCROLL | WS_CHILD |
  WS_VISIBLE | WS_BORDER | WS_TABSTOP, 101,
  104, 104, 12
DEFPUSHBUTTON "OK", IDOK, 214, 4, 28, 14,
  WS_CHILD | WS_VISIBLE | WS_TABSTOP
PUSHBUTTON "Cancel", IDCANCEL, 214, 38, 28,
  14, WS_CHILD | WS_VISIBLE | WS_TABSTOP
RTEXT "Application class name:", -1, 12,
  28, 83, 8, SS_RIGHT | WS_CHILD |
  WS_VISIBLE | WS_GROUP
RTEXT "Main window class:", -1, 12, 44, 83,
  8, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
RTEXT "Main window menu:", -1, 12, 60, 83,
  8, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
RTEXT "Main window title:", -1, 12, 76, 83,
  8, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
RTEXT "Program comment:", -1, 12, 92, 83, 8
  , SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
RTEXT "Author:", -1, 12, 108, 83, 8,
  SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "Template:", -1, 2, 126, 35, 8
```

```
        CONTROL "", TEMPNAME, "STATIC", SS_SIMPLE |
           SS_NOPREFIX | WS_CHILD | WS_VISIBLE |
           WS_GROUP, 41, 126, 201, 8
        CONTROL "", TEMPCOM, "STATIC", SS_SIMPLE |
          SS_NOPREFIX | WS_CHILD | WS_VISIBLE |
          WS_GROUP, 2, 138, 241, 8
        RTEXT "Application name:", -1, 12, 10, 83,
          8, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
END

/* This is a modified copy of the standard filemenu.rc */
FILECOMMANDS  MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MenuItem  "&New", CM_MDIFILENEW
    MenuItem  "&Open...", CM_MDIFILEOPEN
    MenuItem  "&Save", CM_FILESAVE
    MenuItem  "Save &As...", CM_FILESAVEAS
    MenuItem  "&Close", CM_CLOSE
    MenuItem  "C&lose all", CM_CLOSECHILDREN
    MenuItem  SEPARATOR
    MenuItem "New A&pplication..", CM_APPL
    MenuItem SEPARATOR
    MenuItem  "E&xit", CM_EXIT
  END
  POPUP "&Edit"
  BEGIN
    MenuItem  "&Undo\aAlt+BkSp", CM_EDITUNDO
    MenuItem  SEPARATOR
```

```
    MenuItem  "&Cut\aShift+Del", CM_EDITCUT
    MenuItem  "C&opy\aCtrl+Ins", CM_EDITCOPY
    MenuItem  "&Paste\aShift+Ins", CM_EDITPASTE
    MenuItem  "&Delete\aDel", CM_EDITDELETE
    MenuItem  "C&lear All\aCtrl+Del", CM_EDITCLEAR
  END
  POPUP "&Search"
  BEGIN
    MenuItem  "&Find...", CM_EDITFIND
    MenuItem  "&Replace...", CM_EDITREPLACE
    MenuItem  "&Next\aF3", CM_EDITFINDNEXT
  END
  POPUP "&Window"
    BEGIN
      MENUITEM "&Tile", CM_TILECHILDREN,
      MENUITEM "&Cascade", CM_CASCADECHILDREN,
      MENUITEM "Arrange &Icons", CM_ARRANGEICONS,
      MENUITEM "C&lose All", CM_CLOSECHILDREN
    END
END

#include <fileacc.rc>
#include <inputdia.dlg>
```

Listing 5-7. OWLTWIN.TPL

```
Template for ordinary application w/TWindow subclass
// Program ''1 Created ''d by ''7
// ''6

#include <owl.h>
```

```
/* Application class */
class ''2 : public TApplication
  {
  public:
  ''2(LPSTR n,HINSTANCE hi,HINSTANCE hprev,
      LPSTR cmd,int show)
    : TApplication(n,hi,hprev,cmd,show) {};

  virtual void InitMainWindow();
  virtual void InitInstance();
  };

/* Window class */
class ''3 : public TWindow
  {
  public:
  ''3(PTWindowsObject parent, LPSTR title);
/* Add your methods here... */
  };

''3::''3(PTWindowsObject parent, LPSTR title)
   : TWindow(parent,title)
  {
/* Set menu */
  AssignMenu("''4");
/* Do other window constructor things here... */
  }

/* Create main window */
```

```
void ''2::InitMainWindow()
  {
  MainWindow=new ''3(NULL,"''5");
  }

/* Load accelerators. Doesn't matter if there isn't
   one */
void ''2::InitInstance()
  {
  TApplication::InitInstance();
  HAccTable=LoadAccelerators(hInstance,"YourAccel");
// Other init here...
  }

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE
    hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
  {
  ''2  Main("''1",hInstance,hPrevInstance,
           lpCmdLine,nCmdShow);
  Main.Run();
  return Main.Status;
  }
```

Listing 5-8. OWLMDI.TPL

```
Template for MDI application w/TWindow subclass
// Program ''1 Created ''d by ''7
// ''6

#include <owl.h>
#include <string.h>
```

```
#include <filedial.h>

/* Application class */
class ''2 : public TApplication
  {
  public:
  ''2(LPSTR n,HINSTANCE hi,HINSTANCE hprev,
      LPSTR cmd,int show)
    : TApplication(n,hi,hprev,cmd,show) {};

  virtual void InitMainWindow();
  virtual void InitInstance();
  };

class MDIFrame : public TMDIFrame
  {
  public:
  MDIFrame(LPSTR title) : TMDIFrame(title,"''4") {};
  virtual PTWindowsObject CreateChild();
  virtual void NewFile(RTMessage Msg) =
     [CM_FIRST + CM_MDIFILENEW];
  virtual void OpenFile(RTMessage Msg) =
                 [CM_FIRST + CM_MDIFILEOPEN];
  };


/* Main child window class */
class ''3 : public TWindow
  {
  public:
```

```
  ''3(PTWindowsObject parent, LPSTR title);
/* Add your methods here... */
  };

void MDIFrame::NewFile(RTMessage)
  {
  GetApplication()->MakeWindow(new ''3(this,"''5"));
  }

/* Respond to "Open" command by constructing, creating,
  and setting up a new  MDI child */
void MDIFrame::OpenFile(RTMessage)
  {
  char filename[66];
  if ( GetApplication()->ExecDialog(new
       TFileDialog(this, SD_FILEOPEN,
       strcpy(filename, "*.*"))) == IDOK )
          GetApplication()->MakeWindow(new
            ''3(this,filename));
  }



PTWindowsObject MDIFrame::CreateChild()
  {
  return GetApplication()->MakeWindow(
         new ''3(this,""));
  }

''3::''3(PTWindowsObject AParent, LPSTR ATitle)
```

```
        : TWindow(AParent,ATitle)
    {
    }

void ''2::InitMainWindow()
    {
    MainWindow=new MDIFrame("''1");
/*********************************************/
/* Set ChildMenuPos to correct number here... */
    ((MDIFrame *)MainWindow)->ChildMenuPos=3;
    }

void ''2::InitInstance()
    {
    TApplication::InitInstance();
/*********************************************/
/* Set your accelerator table here            */
    HAccTable=LoadAccelerators(hInstance,"Youracc");
    }

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance,
  LPSTR lpCmdLine, int nCmdShow)
  {
  ''2 Main ("''1", hInstance, hPrevInstance,
    lpCmdLine, nCmdShow);
  Main.Run();
  return Main.Status;
  }
```

6

Quick on the Draw: Programming Visually

WHAT'S IN THIS CHAPTER

This chapter familiarizes you with how to use visual programming tools (especially Microsoft's Visual C++) to create Windows applications.

PREREQUISITES

To get the most from this chapter, you should have a working knowledge of C++ and basic Windows programming.

Several vendors offer products that generate code from your sketches of windows, dialogs, and menus. Some of these products do very little; others are quite ambitious.

Borland's ProtoGen, for example, allows you to create a main window, menu, and dialog boxes. Protogen will create a skeletal application for C or Borland's OWL (see Chapter 5). This skeleton has the code necessary to create your window and dialogs. ProtoGen also writes code that allows menu items and dialog buttons to activate dialogs. You have to write all remaining code yourself.

Perhaps the best-known product of this type is the Microsoft Visual C++ (VC++) environment. The entire user interface of this product is geared toward visual Windows programming. Still, there is a price to pay: VC++ only supports visual programming with the Microsoft Foundation Classes (MFC), a C++ class library that partially replaces the Windows API. If you don't want to use MFC, VC++ is just the Microsoft C/C++ compiler with a Windows-based front end.

What VC++ Isn't

While Microsoft calls VC++ a "visual" language, it isn't actually very visual. If you are expecting a Visual Basic environment or something similar to the NeXTSTEP Interface Builder, you will be disappointed.

Most visual products treat your user interface as the central part of the program—code resides in user interface elements or in objects that link to interface elements. VC++ is code centered. As with any resource editor, you

visually design dialogs, menus, and other resources. VC++ allows you to automatically link objects and interfaces, but the process is not truly visual.

Elements of VC++

The core of VC++ is the Visual Workbench. The workbench is a Windows text editor and project manager. It contains a project manager that uses standard make files to build your program. You can modify files and use a simple debugger from within the text editor. The workbench also provides a source browser that allows you to cross-reference your code.

The App Studio is Microsoft's resource editor. You can create and modify dialogs, menus, bitmaps, tables, etc. Although App Studio is a separate program, it works best if you launch it from the workbench. The studio is adequate, but it is missing some things that other vendors offer (text entry in icons, for example).

So far, none of these tools are significantly different from other C/C++ compilers. However, VC++ also comes with MFC and with special tools dedicated to it.

The first MFC tool you will encounter is App Wizard. This is essentially a super editor macro that builds skeletal programs for you. You supply the program's name, select some options (see Figure 6-1), and App Wizard copies some boilerplate code into a directory substituting appropriate names throughout. Notice that App Wizard doesn't write any unique code; it only selects chunks of

boilerplate code and substitutes text strings in the right places.

After App Wizard builds your files, you must edit them to provide the functions you need. You'll also need to modify the default resources App Wizard supplies. App Wizard uses its own indentation style. If you use a different style, you will either have to manually change it or change the way you indent code. On the other hand, if you don't make changes, you can easily tell what code App Wizard wrote and what code you added.

Figure 6-1. App Wizard option dialog

While adding resources, you will find Class Wizard—the final major component of VC++—useful. Class Wizard allows you to connect user interface items and Windows messages to objects in your program. Like App Wizard, Class Wizard doesn't really write any *real* code. It just leaves function stubs in your program, which you must complete. Even a simple action like popping up a dialog in response to a menu click will require you to write code.

Features Offered by App Wizard

App Wizard gives a basic MFC application many features for free. For example, you can get a tool bar and status bar with practically no effort. You can also use Visual Basic 1.0 VBX controls (see the sidebar *C++ Meets Basic*). App Wizard also knows how to build basic Multiple Document Interface (MDI) and Object Linking and Embedding (OLE) applications.

App Wizard can put basic printing code in your application. However, in nearly all cases, you will want to modify its code to properly scale the printed image.

MFC provides several predefined classes to create common documents like forms and text editors. Unfortunately, App Wizard doesn't know about these classes. If you want to use them, you have to modify App Wizard's code. (You'll see how shortly.)

MFC in Detail

MFC applications all have a similar architecture. MFC programs center around documents. An MFC document is a data structure that the user needs to view and possibly modify.

Every MFC program has:

- *Exactly one application object (similar to WinMain).*
- *A Frame window object (controls the application window).*
- *One or more document templates (controls document creation).*
- *One or more document objects (contains open documents).*
- *One or more view objects (displays a document and accepts commands that relate to it).*

You need one document template for each type of document your program handles. Each open document has its own document object and may have one or more views. With multiple views, you can have several windows open on one document simultaneously.

MFC provides root classes for these objects. Ordinarily, you (or App Wizard) will subclass them to create similar objects that have additional functions specific to your program.

If this sounds like the traditional Windows programming model, that's because it is. If you recall Chapter 1, document objects correspond to the application model; the

view object stands in for a traditional program's update routine and some program logic.

Managing MFC Documents

Most MFC objects derive from a common ancestor, CObject. CObject provides some basic runtime support for objects. Perhaps the most important operation CObject supports is serialization—that is, reading and writing representations of objects. You can serialize an object to a disk file, for example, and later read the object back from the disk. (You can also serialize an object over a network connection or modem to another program—but that is another subject.)

MFC uses serialization as its primary means of reading and writing documents to disk. When you open a document, MFC reads the file as a serialization archive. This creates copies of all the objects in the document in the same state in which they were saved. As long as each document object you use knows how to read and write itself to an archive (a special type of stream), MFC will take care of your file open and save commands automatically.

Message Handling in MFC

Traditional Windows programs use giant switch statements to process incoming messages. For example:

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

```
    ...
case WM_CREATE:
    ...
}
```

MFC dispenses with switch-statement message dispatchers. Instead, MFC and Class Wizard automatically manage message maps for you. For example, you can tell Class Wizard to route all WM_CREATE messages to a member function in an object (OnCreate()). For WM_COMMAND messages, you can select specific IDs to associate with a member function. For example, when the user selects a menu item with an ID of IDM_CROP, the OnCrop() function can automatically get control.

Using Dialog Boxes

Class Wizard allows you to easily use dialog boxes from MFC. The idea is that each control in a dialog box (except buttons) can have a corresponding variable in your dialog object (derived from CDialog). When you start the dialog, it sets the values for the fields from these variables. When a modal dialog ends, MFC automatically transfers the values back into the variables. For a modeless dialog, you can force a transfer either way, any time.

MFC can also validate values in dialog fields. You could specify that an edit field only accept integers between 0 and 100, for example. MFC will automatically enforce the range at runtime.

The Bottom Line

If you moved from C to C++ expecting to simplify programming, you were probably disappointed. Sure, there are advantages to writing in C++ (encapsulation and object reuse, for example). But, your C++ programs probably are not substantially simpler to create than comparable C programs. The trick to increasing productivity is to reuse existing objects.

Migrating from the SDK to MFC is similar to moving from C to C++. MFC is little more than an object-oriented Windows API. Although there are some advantages, your programs won't just write themselves. However, you can get some relief from Microsoft's predefined view classes. Also, as time passes, you should be building your own library of useful objects.

Three Special Views

Like other class libraries (for example, Borland's OWL in Chapter 5), MFC provides some predefined classes to simplify certain types of programs. MFC supplies three interesting view objects: CScrollView, a window that automates scrolling; CEditView, a text editor; and CFormView, a dialog-style form.

The CScrollView object automates scroll bar handling for you. You can ask it to scroll a large document in a small window or scale the window to fit. CScrollView manipulates the origin of your drawing surface so that you simply draw the entire document regardless of the scroll bar position. While this can be handy, it also may be too slow to be of use for many programs.

CEditView is unusual because it essentially combines the document object and the view object—that is, the CEditView object holds the edited text and displays it. You have to create a small document object that delegates all of the work to the view object. (You'll see how soon.)

CFormView allows you to create form-based programs using a dialog template. This process is similar to the effect achieved by PHONE.C (see Chapter 3). Unlike PHONE.C, however, a CFormView document is a proper window. This allows MFC to automatically provide scroll bars, for example.

A Simple Example

GROUPEXE is a complete MFC application that uses the CEditView class. It takes a list of programs and executes them on demand. App Wizard built most of it (see Table 6-1). Only the lines marked with the grey shadows in the listings were added or modified. See the sidebar *Creating GROUPEXE* for more details on using the visual tools of VC++ to duplicate GROUPEXE.

Table 6-1. GROUPEXE Source Files.

File	Contents	Listing	Manual Changes
GROUPEXE.H	Header	Listing 6-1	None
GROUPEXE.CPP	Main framework file	Listing 6-2	None

(Cont.)

Table 6-1. GROUPEXE Source Files (Continued)

File	Contents	Listing	Manual Changes
GROUPDOC.H	Document object header	Listing 6-3	Yes
GROUPDOC.CPP	Document object	Listing 6-4	Yes
GROUPVW.H	View object header	Listing 6-5	Yes
GROUPVW.CPP	View object	Listing 6-6	Yes
RUNSTATE.H	Header for dialog	Listing 6-7	Yes
RUNSTATE.CPP	Dialog box code	Listing 6-8	Yes
MAINFRM.H	Main window header	Listing 6-9	None
MAINFRM.CPP	Main window object	Listing 6-10	None
STDAFX.H	MFC header	Listing 6-11	None
GROUPEXE.RC	App Studio resources	Listing 6-13	None
RESOURCE.H	Other resources	Listing 6-14	None
GROUPEXE.RC2	Non-App Studio resources	Listing 6-15	None
GROUPEXE.DEF	Linker DEF file	Listing 6-16	None

Note: See the Sidebar "Creating GROUPEXE" for more details. In the listings, a grey shadow indicates areas of code that require manual changes.

GROUPEXE uses CEditView to provide a complete text editor. Since App Wizard doesn't know how to create a CEditView object, your first task is to change all occurrences of CView to CEditView in GROUPVW.H and GROUPVW.CPP (Listings 6-5 and 6-6).

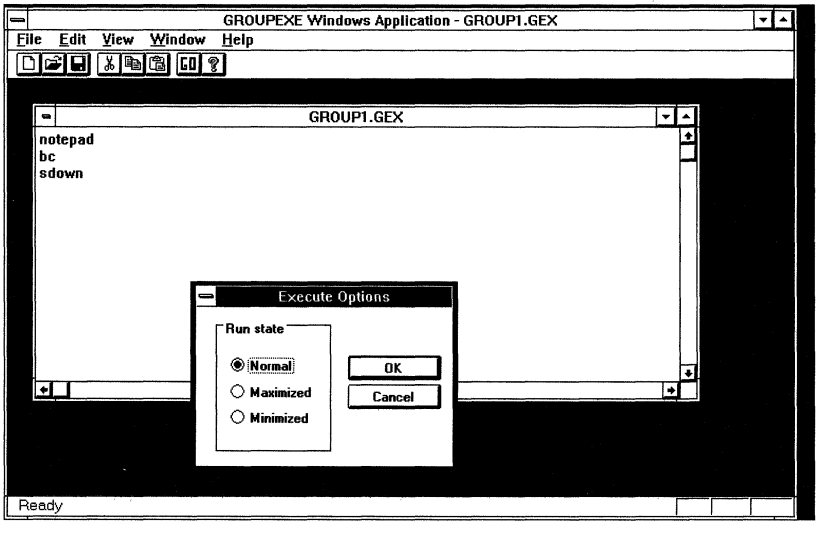
Although CEditView is the true document, MFC still requires a separate document object. GROUPDOC.CPP

(Listing 6-4) provides a simple surrogate document that passes all of its work to the GroupView object.

Notice that GroupDoc's Serialize function is unusual. Ordinarily, documents read and write their internal representations to a binary format file. Text-editing programs like GROUPEXE often want to read and write ASCII files. The SerializeRaw() function of CEditView provides this capability.

GROUPEXE uses a dialog box to set the mode it uses when running programs (see Figure 6-2 and Listing 6-8). You can easily sketch this dialog box with App Studio. (The dialog box's ID is IDD_STATEDLG.) Then, using Class Wizard, you can create a new object to represent the dialog. First, derive CRunStateDlg from the standard CDialog class. Then, attach the radio buttons to the runstate

Figure 6-2. GROUPEXE options dialog



variable in the new class. This variable takes on values from 0 to 2, depending on which radio button is active.

You can also use App Studio to add the two nonstandard menu items (ID_EXEC and ID_EXECOPT). Use Class Wizard to automatically put function stubs in your view object to correspond with these menu items. Note that you must write the code that makes the actions happen—no matter how simple they are.

Creating GROUPEXE

If you want to build GROUPEXE yourself, you can use App Wizard and Class Wizard to create the source files. These instructions assume you have some basic familiarity with VC++'s tools. Just follow these steps:

1. Run App Wizard.
2. From the initial App Wizard dialog box, enter GROUPEXE as the project name and select the directory you want to use (see below).

App Wizard main dialog for GROUPEX

3. Push App Wizard's Options button. Turn off the checkbox that enables printing and print preview. Return to the main dialog by pressing the OK button.

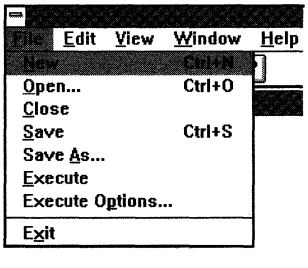
4. Push the Classes button. Change the class names as follows:

Change	To
CGroupexeApp	CGroupApp
CGroupexeDoc	CGroupDoc
CGroupexeView	CGroupView

Select the CGroupView class and set the file extension to .GEX. Change the document name from Groupe to Group. Return to the main dialog by pressing the OK button.

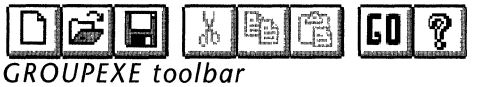
5. Press the OK button followed by the Create button. App Wizard will generate most of the source files.

6. Although you told App Wizard you didn't want printing code, it fails to remove the print icon from the toolbar.Edit MAINFRM. CPP and change ID_FILE_PRINT to ID_EXEC in the buttons[] array. You can also invoke App Studio to modify the toolbar's printer bitmap (see below).



GROUPEXE file menu

7. Use App Studio to modify the IDR_GROUPTYPE File menu to match the one shown below. Set the identifiers of the new menu items to ID_EXEC and ID_EXECOPT. Use "Execute programs in list" and "Set execution options" as prompts for the new menu items.



GROUPEXE toolbar

8. Activate Class Wizard (use App Studio's Resource menu) and connect ID_EXEC's COMMAND message to the OnExec() function in the GroupView object (use Class Wizard's Add Function button). Then, connect ID_EXECOPT to OnExecOpt(). Close Class Wizard with the OK button.

9. Use App Studio to create a dialog similar to the one in Figure 6-2. Use IDD_STATEDLG as the dialog ID. The radio buttons should have IDs of IDC_NORMAL, IDC_MAX, and IDC_MIN. Only the IDC_NORMAL button should have its group and tab-stop options set.

10. Start Class Wizard from the App Studio Resource menu and create the CRunStateDlg class to represent the dialog. Enter CRunStateDlg in the Class Name field and press the Create Class button.

11. Use Class Wizard's Edit Variables button to attach the first radio button (IDC_NORMAL) to the runstate variable in the CRunStateDlg class. Use the Add Variable button, set the member variable name to runstate, and press OK. Then, press Close and OK to return to App Studio.

12. Edit the new files so that they match Listings 6-1 to 6-16. The areas you need to change are shown in a grey shade. Due to the listing format, your files may have different line breaks and comments from the listings. These difference will not affect your program.

Is VC++ for You?

Of course, VC++ is too complex to cover in one short chapter. Still, you now should understand how VC++ and MFC work together. If you are comfortable with C++ and willing to give us the standard SDK, you may be very happy with VC++.

However, VC++ isn't a true commando tool. It isn't much easier to write an MFC application than to write a conventional SDK program. You *do* get the advantages of C++ (class reuse, for example), but most programs that are simple to write for MFC are simple to write with the SDK (with a few exceptions).

On the other hand, MFC *does* simplify advanced features like MDI, OLE, and print preview. However, the resulting programs are not *simple*—just simpler than their SDK counterparts.

C++ Meets Basic

One of the most touted features of VC++ is its ability to use Visual Basic VBX controls. VBX controls can be simple (three-dimensional push buttons) or complex (multimedia controllers—or even spreadsheets). With the surge in popularity of Visual Basic, many vendors now offer VBX controls.

Like ordinary controls, you can create VBX controls dynamically at runtime or design them into dialogs using App Studio. However, be aware of the following:

- *VC++ only supports VBXs that work with Visual Basic 1.0. VC++ can't use version 2.0 controls.*
- *To allow VBX controls to run, MFC emulates a portion of Visual Basic. However, this emulation is not complete. Some controls may not work; others may behave erratically.*
- *If you distribute applications that use VBX controls, you must include the .VBX files that contain them. Different vendors may have different policies on redistribution of their VBX controls.*

Once you install a VBX control into App Studio, you can use it just like any other control. A VBX control's property sheet has an extra page that contains control properties. For example, the

CIRC3.VBX control that comes with VC++ has properties like Caption and CircleShape. The Caption property determines what text the circle will display. CircleShape causes the control to be perfectly circular if it is 0 and elliptical if it is 1.

You can set control properties from App Studio or from inside your program. If the control is in a dialog, you need to use Class Wizard's Add Variables command to set a pointer to the VBX control in your program. Then, you can use the SetNumProperty() and SetStrProperty() member functions to change any properties you like. You also can assign variables to specific properties using Class Wizard. Each VBX control defines different properties; you'll need the control's documentation to learn about its properties.

VBX controls also have their own messages (like VBN_CLICKIN, for example). Class Wizard can associate these messages with functions just as it does for ordinary Windows messages. Again, each control defines its own messages, so you'll need its documentation to learn about a particular control.

Listing 6-1. GROUPEXE.H

```
// groupexe.h : main header file for the GROUPEXE
// application
//

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

/////////////////////////////////////////////////////////
// CGroupApp:
// See groupexe.cpp for the implementation of this class
//

class CGroupApp : public CWinApp
{
public:
        CGroupApp();

// Overrides
        virtual BOOL InitInstance();

// Implementation

        //{{AFX_MSG(CGroupApp)
        afx_msg void OnAppAbout();
// NOTE - the ClassWizard will add and remove member
// functions here. DO NOT EDIT what you see in these
```

```
// blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////
```

Listing 6-2. GROUPEXE.CPP

```
// groupexe.cpp : Defines the class behaviors for
// the application.

#include "stdafx.h"
#include "groupexe.h"

#include "mainfrm.h"
#include "groupdoc.h"
#include "groupvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////
// CGroupApp

BEGIN_MESSAGE_MAP(CGroupApp, CWinApp)
        //{{AFX_MSG_MAP(CGroupApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
```

```
// NOTE - the ClassWizard will add and remove
// mapping macros here.
// DO NOT EDIT what you see in these blocks
// of generated code !
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////
// CGroupApp construction

CGroupApp::CGroupApp()
{
        // TODO: add construction code here,
        // Place all significant initialization
        // in InitInstance
}

/////////////////////////////////////////////////////
// The one and only CGroupApp object

CGroupApp NEAR theApp;

/////////////////////////////////////////////////////
// CGroupApp initialization

BOOL CGroupApp::InitInstance()
{
```

```
        // Standard initialization
        // If you are not using these features and
        // wish to reduce the size  of your final
        // executable, you should remove from the following
        // the specific initialization routines you
        // do not need.
// set dialog background color to gray
        SetDialogBkColor();
// Load standard INI file options (including MRU)
        LoadStdProfileSettings();

// Register the application's document templates.
// Document templates serve as the connection between
// documents, frame windows and views.

        AddDocTemplate(new CMultiDocTemplate(IDR_GROUPTYPE,
                        RUNTIME_CLASS(CGroupDoc),
// standard MDI child frame
                        RUNTIME_CLASS(CMDIChildWnd),
                        RUNTIME_CLASS(CGroupView)));

        // create main MDI Frame window
        CMainFrame* pMainFrame = new CMainFrame;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
                return FALSE;
        pMainFrame->ShowWindow(m_nCmdShow);
        pMainFrame->UpdateWindow();
        m_pMainWnd = pMainFrame;

// enable file manager drag/drop and DDE Execute open
```

```
        m_pMainWnd->DragAcceptFiles();
        EnableShellOpen();
        RegisterShellFileTypes();

        // simple command line parsing
        if (m_lpCmdLine[0] == '\0')
        {
                // create a new (empty) document
                OnFileNew();
        }
        else if ((m_lpCmdLine[0] == '-' ||
                  m_lpCmdLine[0] == '/') &&
                 (m_lpCmdLine[1] == 'e' ||
                  m_lpCmdLine[1] == 'E'))
        {
// program launched embedded - wait for DDE or OLE open
        }
        else
        {
// open an existing document
        OpenDocumentFile(m_lpCmdLine);
        }

        return TRUE;
}

/////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
```

```
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

// Implementation
protected:
        // DDX/DDV support
        virtual void DoDataExchange(CDataExchange* pDX);
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CGroupApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////
// CGroupApp commands
```

Listing 6-3. GROUPDOC.H

```
// groupdoc.h : interface of the CGroupDoc class
//
/////////////////////////////////////////////////////////////

class CGroupDoc : public CDocument
{
protected: // create from serialization only
        CGroupDoc();
        DECLARE_DYNCREATE(CGroupDoc)

// Attributes
```

```
public:

// Operations
public:

// Implementation
public:
        virtual ~CGroupDoc();
// overridden for document i/o
        virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
        virtual BOOL    OnNewDocument();
// Generated message map functions
protected:
        //{{AFX_MSG(CGroupDoc)
// NOTE - the ClassWizard will add and remove member
// functions here. DO NOT EDIT what you see in these
// blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

Listing 6-4. GROUPDOC.CPP

```
// groupdoc.cpp : implementation of the CGroupDoc class
```

```
//

#include "stdafx.h"
#include "groupexe.h"

#include "groupdoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////
// CGroupDoc

IMPLEMENT_DYNCREATE(CGroupDoc, CDocument)

BEGIN_MESSAGE_MAP(CGroupDoc, CDocument)
        //{{AFX_MSG_MAP(CGroupDoc)
// NOTE - the ClassWizard will add and
// remove mapping macros here.
// DO NOT EDIT what you see in these blocks
// of generated code !
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////
// CGroupDoc construction/destruction

CGroupDoc::CGroupDoc()
```

```
{
        // TODO: add one-time construction code here
}

CGroupDoc::~CGroupDoc()
{
}

BOOL CGroupDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        return TRUE;
}

/////////////////////////////////////////////////////
// CGroupDoc serialization

void CGroupDoc::Serialize(CArchive& ar)
{
  POSITION tmppos;
  CEditView *cv;
/* Get first view */
  tmppos=GetFirstViewPosition();
  cv=(CEditView *)GetNextView(tmppos);
/* Make it serialize in ASCII */
  cv->SerializeRaw(ar);
```

```
}


/////////////////////////////////////////////////
// CGroupDoc diagnostics

#ifdef _DEBUG
void CGroupDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CGroupDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////
// CGroupDoc commands
```

Listing 6-5. GROUPVW.H

```
// groupvw.h : interface of the CGroupView class
//
/////////////////////////////////////////////////

class CGroupView : public CEditView
{
protected: // create from serialization only
```

```
        CGroupView();
        DECLARE_DYNCREATE(CGroupView)

// Attributes
protected:
// current run state (normal, min, max)
        int runstate;
public:
        CGroupDoc* GetDocument();

// Operations
public:

// Implementation
public:
        virtual ~CGroupView();
// overridden to draw this view
        virtual void OnDraw(CDC* pDC);
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif


// Generated message map functions
protected:
        //{{AFX_MSG(CGroupView)
        afx_msg void OnExec();
        afx_msg void OnExecopt();
        //}}AFX_MSG
```

```
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in groupvw.cpp
inline CGroupDoc* CGroupView::GetDocument()
   { return (CGroupDoc*) m_pDocument; }
#endif

/////////////////////////////////////////////////
```

Listing 6-6. GROUPVW.CPP

```
// groupvw.cpp : implementation of the CGroupView class
//

#include "stdafx.h"
#include <ctype.h>
#include "groupexe.h"

#include "groupdoc.h"
#include "groupvw.h"
#include "runstate.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////
// CGroupView
```

```
IMPLEMENT_DYNCREATE(CGroupView, CEditView)

BEGIN_MESSAGE_MAP(CGroupView, CEditView)
        //{{AFX_MSG_MAP(CGroupView)
        ON_COMMAND(ID_EXEC, OnExec)
        ON_COMMAND(ID_EXECOPT, OnExecopt)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////
// CGroupView construction/destruction

CGroupView::CGroupView()
{
        runstate=0;
}

CGroupView::~CGroupView()
{
}

/////////////////////////////////////////////////
// CGroupView drawing

void CGroupView::OnDraw(CDC* pDC)
{
        CEditView::OnDraw(pDC);
}
```

```
////////////////////////////////////////////////
// CGroupView diagnostics

#ifdef _DEBUG
void CGroupView::AssertValid() const
{
        CEditView::AssertValid();
}

void CGroupView::Dump(CDumpContext& dc) const
{
        CEditView::Dump(dc);
}

// non-debug version is inline
CGroupDoc* CGroupView::GetDocument()
{
        ASSERT(m_pDocument->
              IsKindOf(RUNTIME_CLASS(CGroupDoc)));
        return (CGroupDoc*) m_pDocument;
}

#endif //_DEBUG

////////////////////////////////////////////////
// CGroupView message handlers

void CGroupView::OnExec()
{
        CEdit editor;
```

```
        char buf[66],*bufp;
        int lnno=0, max=GetEditCtrl().GetLineCount();
        int l,runstyle;
        runstyle=runstate==0?SW_NORMAL:((runstate==1)?
                SW_SHOWMAXIMIZED:SW_SHOWMINIMIZED);
        while (max--)
          {
          l=GetEditCtrl().GetLine(lnno++,buf,sizeof(buf));
          if (*buf==';') continue;
          bufp=buf;
          buf[l]='\0';  // GetLine doesn't NULL terminate!
          while (*bufp&&isspace(*bufp)) bufp++;
          if (*bufp=='\0') continue;
          if (WinExec(bufp,runstyle)<32)
            AfxMessageBox("Can't execute command",
              MB_OK|MB_ICONSTOP);
          }

}

void CGroupView::OnExecopt()
{
        CRunStateDlg dlg;
        dlg.runstate=runstate;
        if (dlg.DoModal()==IDOK) runstate=dlg.runstate;
}
```

Listing 6-7. RUNSTATE.H

```
// runstate.h : header file
//
/////////////////////////////////////////////////////////////
// CRunStateDlg dialog

class CRunStateDlg : public CDialog
{
// Construction
public:
// standard constructor
        CRunStateDlg(CWnd* pParent = NULL);

// Dialog Data
        //{{AFX_DATA(CRunStateDlg)
        enum { IDD = IDD_STATEDLG };
        int             runstate;
        //}}AFX_DATA

// Implementation
protected:
// DDX/DDV support
        virtual void DoDataExchange(CDataExchange* pDX);

        // Generated message map functions
        //{{AFX_MSG(CRunStateDlg)
// NOTE: the ClassWizard will add member functions here
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Listing 6-8. RUNSTATE.CPP

```
// runstate.cpp : implementation file
//

#include "stdafx.h"
#include "groupexe.h"
#include "runstate.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////
// CRunStateDlg dialog

CRunStateDlg::CRunStateDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CRunStateDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CRunStateDlg)
        runstate = 0;
        //}}AFX_DATA_INIT
}

void CRunStateDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CRunStateDlg)
        DDX_Radio(pDX, IDC_NORMAL, runstate);
        //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CRunStateDlg, CDialog)
        //{{AFX_MSG_MAP(CRunStateDlg)
// NOTE: the ClassWizard will add message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////
// CRunStateDlg message handlers
```

Listing 6-9. MAINFRM.H

```
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////

class CMainFrame : public CMDIFrameWnd
{
        DECLARE_DYNAMIC(CMainFrame)
public:
        CMainFrame();

// Attributes
public:

// Operations
public:

// Implementation
public:
```

```
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif


protected:      // control bar embedded members
        CStatusBar      m_wndStatusBar;
        CToolBar        m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT
lpCreateStruct);
// NOTE - the ClassWizard will add and remove
// member functions here. DO NOT EDIT what you see
// in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////
```

Listing 6-10. MAINFRM.CPP

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
```

```
#include "groupexe.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
// NOTE - the ClassWizard will add and remove
// mapping macros here. DO NOT EDIT what you see in
// these blocks of generated code !
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
```

```
        ID_FILE_NEW,
        ID_FILE_OPEN,
        ID_FILE_SAVE,
                ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
                ID_SEPARATOR,
        ID_EXEC,
        ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
// status line indicator
        ID_SEPARATOR,
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
```

```
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE("Failed to create toolbar\n");
                return -1;              // fail to create
        }

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE("Failed to create status bar\n");
                return -1;              // fail to create
        }

        return 0;
}


////////////////////////////////////////////////////
```

```
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////
// CMainFrame message handlers
```

Listing 6-11. STDAFX.H

```
// stdafx.h : include file for standard system include
// files, or project specific include files that are
// used frequently, but are changed infrequently
//

// MFC core and standard components
#include <<afxwin.h>>
// MFC extensions (including VB)
#include <<afxext.h>>
```

Listing 6-12. STDAFX.CPP

```
// stdafx.cpp : source file that includes just the
// standard includes
//       stdafx.pch will be the pre-compiled header
//       stdafx.obj will contain the pre-compiled
//       type information

#include "stdafx.h"
```

Listing 6-13. GROUPEXE.RC

```
//Microsoft App Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////
//
// TEXTINCLUDE
//
```

```
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
// non-App Studio edited resources\r\n"
    "#include ""res\\groupexe.rc2""
    "\r\n"
    "#include ""afxres.rc""  // Standard components\r\n"
    "\0"
END

//////////////////////////////////////////////////
#endif    // APSTUDIO_INVOKED


//////////////////////////////////////////////////
//
// Icon
//

IDR_MAINFRAME  ICON    DISCARDABLE     "RES\\GROUPEXE.ICO"
```

```
IDR_GROUPTYPE  ICON    DISCARDABLE     "RES\\GROUPDOC.ICO"

//////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME   BITMAP  MOVEABLE PURE   "RES\\TOOLBAR.BMP"

//////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",          ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",      ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "Recent File",   ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                      ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",            ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",         ID_VIEW_STATUS_BAR
```

```
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About GROUPEXE...",       ID_APP_ABOUT
    END
END

IDR_GROUPTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",              ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",          ID_FILE_OPEN
        MENUITEM "&Close",                    ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",             ID_FILE_SAVE
        MENUITEM "Save &As...",              ID_FILE_SAVE_AS
        MENUITEM "&Execute",                  ID_EXEC
        MENUITEM "Execute O&ptions...",       ID_EXECOPT
        MENUITEM SEPARATOR
        MENUITEM "Recent File",      ID_FILE_MRU_FILE1,
                  GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",             ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",        ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",         ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",        ID_EDIT_COPY
```

```
        MENUITEM "&Paste\tCtrl+V",      ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",            ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",         ID_VIEW_STATUS_BAR
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade",            ID_WINDOW_CASCADE
        MENUITEM "&Tile",               ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",      ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About GROUPEXE...", ID_APP_ABOUT
    END
END


//////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "N",            ID_FILE_NEW,  VIRTKEY,CONTROL
    "O",            ID_FILE_OPEN, VIRTKEY,CONTROL
    "S",            ID_FILE_SAVE, VIRTKEY,CONTROL
```

```
    "Z",            ID_EDIT_UNDO, VIRTKEY,CONTROL
    "X",            ID_EDIT_CUT,  VIRTKEY,CONTROL
    "C",            ID_EDIT_COPY, VIRTKEY,CONTROL
    "V",            ID_EDIT_PASTE,VIRTKEY,CONTROL
    VK_BACK,        ID_EDIT_UNDO, VIRTKEY,ALT
    VK_DELETE,      ID_EDIT_CUT,  VIRTKEY,SHIFT
    VK_INSERT,      ID_EDIT_COPY, VIRTKEY,CONTROL
    VK_INSERT,      ID_EDIT_PASTE,VIRTKEY,SHIFT
    VK_F6,          ID_NEXT_PANE, VIRTKEY
    VK_F6,          ID_PREV_PANE, VIRTKEY,SHIFT
END


//////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  34, 22, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About GROUPEXE"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT           "by Al Wil-
liams",IDC_STATIC,16,24,119,8
    DEFPUSHBUTTON   "OK",IDOK,176,6,32,14,WS_GROUP
    LTEXT           "GROUPEXE - Group execution applica-
tion"
                    ,IDC_STATIC,17,7,147,15
END
```

```
IDD_STATEDLG DIALOG DISCARDABLE  0, 0, 139, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Execute Options"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL  "Normal",IDC_NORMAL,"Button",
             BS_AUTORADIOBUTTON |WS_GROUP | WS_TABSTOP,
             19,29,34,10
    CONTROL          "Maximized",IDC_MAX,"Button",
                     BS_AUTORADIOBUTTON,19,44,44,10
    CONTROL          "Minimized",IDC_MIN,"Button",
                     BS_AUTORADIOBUTTON,19,59,43,10
    DEFPUSHBUTTON    "OK",IDOK,82,28,50,14
    PUSHBUTTON       "Cancel",IDCANCEL,82,45,50,14
    GROUPBOX         "Run state",IDC_STATIC,11,7,62,77
END


//////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME     "GROUPEXE Windows Application"
    IDR_GROUPTYPE     "\nGroup\nGROUP Document\n\
GROUP Files (*.gex)\n.gex\nGroupFileType\nGROUP File Type"
END
```

```
STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "GROUPEXE Windows Application"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_FILE_NEW     "Create a new document"
 ID_FILE_OPEN    "Open an existing document"
 ID_FILE_CLOSE   "Close the active document"
 ID_FILE_SAVE    "Save the active document"
 ID_FILE_SAVE_AS "Save the active document with a new
name"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_APP_ABOUT "Display program information, version \
```

```
number and copyright"
    ID_APP_EXIT "Quit the application; prompts\
to save documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_NEXT_PANE   "Switch to the next window pane"
 ID_PREV_PANE   "Switch back to the previous window pane"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_WINDOW_NEW  "Open another window\
 for the active document"
 ID_WINDOW_ARRANGE "Arrange icons at the\
 bottom of the window"
 ID_WINDOW_CASCADE "Arrange windows so they overlap"
 ID_WINDOW_TILE_HORZ "Arrange windows as \
non-overlapping tiles"
 ID_WINDOW_TILE_VERT "Arrange windows as \
non-overlapping tiles"
```

```
 ID_WINDOW_SPLIT "Split the active window into panes"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_EDIT_CLEAR "Erase the selection"
 ID_EDIT_CLEAR_ALL "Erase everything"
 ID_EDIT_COPY "Copy the selection and put \
it on the Clipboard"
 ID_EDIT_CUT  "Cut the selection and put \
it on the Clipboard"
 ID_EDIT_FIND  "Find the specified text"
 ID_EDIT_PASTE "Insert Clipboard contents"
 ID_EDIT_REPEAT "Repeat the last action"
 ID_EDIT_REPLACE "Replace specific text with\
different text"
 ID_EDIT_SELECT_ALL "Select the entire document"
 ID_EDIT_UNDO "Undo the last action"
 ID_EDIT_REDO "Redo the previously undone action"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_VIEW_TOOLBAR    "Show or hide the toolbar"
 ID_VIEW_STATUS_BAR "Show or hide the status bar"
END

STRINGTABLE DISCARDABLE
BEGIN
 AFX_IDS_SCSIZE   "Change the window size"
```

```
 AFX_IDS_SCMOVE   "Change the window position"
 AFX_IDS_SCMINIMIZE "Reduce the window to an icon"
 AFX_IDS_SCMAXIMIZE "Enlarge the window to full size"
 AFX_IDS_SCNEXTWINDOW "Switch to the next document window"
 AFX_IDS_SCPREVWINDOW "Switch to the previous \
document window"
AFX_IDS_SCCLOSE  "Close the active window and prompts\
to save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
 AFX_IDS_SCRESTORE  "Restore the window to normal size"
 AFX_IDS_SCTASKLIST       "Activate Task List"
 AFX_IDS_MDICHILD         "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
 ID_EXEC     "Execute programs in list"
 ID_EXECOPT  "Set execution options"
END


#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
// non-App Studio edited resources
```

```
#include "res\groupexe.rc2"

#include "afxres.rc"  // Standard components

/////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

Listing 6-14. RESOURCE.H

```
{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by GROUPEXE.RC
//
#define IDR_MAINFRAME                   2
#define IDR_GROUPTYPE                   3
#define IDD_ABOUTBOX                    100
#define IDC_NORMAL                      101
#define IDD_STATEDLG                    101
#define IDC_MAX                         102
#define IDC_MIN                         103
#define ID_EXEC                         32768
#define ID_EXECOPT                      32769

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         32770
#define _APS_NEXT_CONTROL_VALUE         104
```

```
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Listing 6-15. GROUPEXE.RC2

```
// GROUPEXE.RC2 - resources App Studio doesn't edit
//

#ifdef APSTUDIO_INVOKED
        #error this file is not editable by App Studio
#endif //APSTUDIO_INVOKED

////////////////////////////////////////////////////
// Version stamp for this .EXE

#include "ver.h"

VS_VERSION_INFO     VERSIONINFO
  FILEVERSION       1,0,0,1
  PRODUCTVERSION    1,0,0,1
  FILEFLAGSMASK     VS_FFI_FILEFLAGSMASK
#ifdef _DEBUG
  FILEFLAGS VS_FF_DEBUG|VS_FF_PRIVATEBUILD|VS_FF_PRERE-
LEASE
#else
  FILEFLAGS         0 // final version
#endif
  FILEOS            VOS_DOS_WINDOWS16
  FILETYPE          VFT_APP
  FILESUBTYPE       0   // not used
```

```
BEGIN
        BLOCK "StringFileInfo"
        BEGIN
//
// Lang=US English, CharSet=Windows Multilingual
         BLOCK "040904E4"
           BEGIN
           VALUE "CompanyName",     "\0"
           VALUE "FileDescription",
                 "GROUPEXE MFC Application\0"
           VALUE "FileVersion",     "1.0.001\0"
           VALUE "InternalName",    "GROUPEXE\0"
           VALUE "LegalCopyright",  "\0"
           VALUE "LegalTrademarks", "\0"
           VALUE "OriginalFilename","GROUPEXE.EXE\0"
           VALUE "ProductName",     "GROUPEXE\0"
           VALUE "ProductVersion",  "1.0.001\0"
           END
        END
        BLOCK "VarFileInfo"
        BEGIN
// English language (0x409) and
// the Windows ANSI codepage (1252)
          VALUE "Translation", 0x409, 1252
        END
END

////////////////////////////////////////////////
// Add additional manually edited resources here...
```

```
////////////////////////////////////////////////
```

Listing 6-16. GROUPEXE.DEF

```
; groupexe.def : Declares the module parameters
; for the application.

NAME          GROUPEXE
DESCRIPTION   'GROUPEXE Windows Application'
EXETYPE       WINDOWS

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024    ; initial heap size
; Stack size passed as argument to linker's /STACK option
```

7

Biting the Bullet (Or How I Learned to Stop Worrying and Love the SDK)

WHAT'S IN THIS CHAPTER

You will learn how to apply commando principles to traditional Windows programming.

PREREQUISITES

To get the most from this chapter, you'll need a working knowledge of how to program Windows applications in C using the SDK.

As powerful as the commando techniques are, sometimes you have to bite the bullet and write traditional Windows programs. This may occur when your program is too complex (a word processor, for example). Although Visual C++ can help with complex programs, if you aren't comfortable with C++, you can't expect too much help from it.

Remember Commando Commandment IX in Chapter 2. Before you turn to the Windows Software Development Kit (SDK), be sure you have exhausted your simpler options. Could you use a non-C application generator like Toolbook or Visual Basic?

Even if you must use the SDK, parts of your program may still benefit from some commando techniques. For example, the win_printf() and win_input() routines in Chapter 3 are usually helpful. How much functionality can you put in dialogs or text-edit windows (Chapters 3 and 4)?

Down with WM_PAINT!

When you write a traditional Windows program, you get stuck with many tasks that don't directly relate to your program's function. No matter what kind of program you want to write, you have to write code to manage your windows. WM_PAINT routines, scrolling functions, and all the other Windows overhead often take more effort than the actual algorithms of your program. Commando techniques, however, focus on making Windows do the

work via dialogs, edit controls, and other Windows features.

For more general programs, a special commando library can help. The Virtual Window Library, VWINL, will automatically manage your Windows 3.1, Windows NT, or Win32s windows. When you want to display something, you draw it with the usual Windows calls once. VWINL makes sure it stays there and can automatically manage scroll bars, scaling, and other common, tedious tasks. VWINL can even output your windows to a printer.

The Problem with the SDK

Part of the difficulty in writing for Windows lies in the architecture of the typical GUI program. (Chapter 2 covers this in detail.) Windows (and many other GUI systems) dictates a seemingly strange approach. Ordinary Windows programs don't manipulate their display in response to user input (or other events). Instead, they update an internal application model to reflect the program's current state. Upon request from Windows, the program (via its WM_PAINT handler) renders a representation (or view, if you prefer) of the model in a window.

This model of computing is counterintuitive to most programmers. For some applications (word processors and spreadsheets come to mind), this type of architecture works well. For these applications, the model correlates to what you need to create anyway (a document or an

array of values). But many programs don't *need* a model except to satisfy the WM_PAINT message.

New Age Programming

VWINL creates virtual drawing surfaces (VMAPs) that you can draw on with standard Windows GDI calls. You can optionally associate a VMAP with one or more physical windows. You can ask VWINL to scale the VMAP to fit in the window or show as much of the VMAP as will fit. If the VMAP is too large to fit in the window, VWINL can automatically manage scroll bars for you.

Each physical window has an independent view of its VMAP. You can have two (or more) windows on the screen viewing the same VMAP in different ways. For example, one window might show the VMAP scaled to fit, while two others are scrolled to show different areas of the VMAP without scaling.

Once you draw something to a VMAP with a window attached, you won't need to draw it again. If you iconify the window or cover it up and expose it, the image stays in place, with no further action on your part. As an extra bonus, screen updates are unusually fast—often faster than with traditional Windows programs.

Details, Details...

Listing 7-1 shows a very simple program that uses VWINL. (You'll find a summary of VWINL calls in Figure 7-1 and in Appendix B.) Notice that it includes the

VWINL.H file (Listing 7-2) and compiles with VWINL.DEF (Listing 7-3). VWINL programs have a main() function (which is more like a WinMain() function in form) and window callbacks like normal Windows programs. VWINL programs don't have WM_PAINT routines or event loops like ordinary Windows programs.

Figure 7-1. VWINL Calls

```
int Vcreate_window(char *title,DWORD style,int x,int y,int
width, int height,HWND parent,LPCSTR menu,long (*call-
back)(),unsigned vflags,HDC *dc,HWND *win,int show)
```

The Vcreate_window function mostly mimics CreateWindow(). The menu parameter is actually a resource name or ID. The vflags field is a VWINL flag (see Figure 3). The window handle returns via the win pointer, and the VMAP DC (if any) goes to the dc pointer (unless the dc pointer is NULL). The function returns zero upon success. Any other value indicates failure.

```
VMAP *Vcreate_map(int width, int height)
```

Creates a VMAP of the specified width and height. This VMAP will match your current display unless you have set the monochrome mode (see Vset_monomode()).

```
VMAP *Vget_map(HWND w)
```

Returns a pointer to the VMAP associated with the window.

```
void Vdestroy_map(VMAP *map)
```

Releases a VMAP's resources. When a window closes, VWINL attempts to free its VMAP unless the V_NOFREEMAP flag is set.

```
VMAP *Vselect_map(HWND w, VMAP *new)
```

Changes the VMAP associated with a window. If the VMAP pointer is NULL, the window will have no VMAP. The function returns a pointer to the previously selected VMAP.

(Cont.)

Figure 7-1. VWINL Calls (Cont.)

```
void Vcommit_draw(HWND w)
```

Forces the contents of the window's VMAP to appear in the window. Until you call Vcommit_draw(), any output to the VMAP may or may not be visible. This call is actually a macro.

```
HDC Vget_mdc(VMAP *map)
```

Returns the DC associated with the specified VMAP. This call is actually a macro.

```
int Vget_stretchmode(VMAP *map)
```

Returns the stretch mode for the specified VMAP. For more about stretch modes, see the SetStretchBltMode() function in the Windows API reference. This function is actually a macro.

```
void Vget_info(HWND w,MEMWINFO *info)
```

Returns a read-only structure of information pertaining to the window.

```
unsigned long Vset_flags(HWND w,unsigned long flags,int cmd)
```

You can use Vset_flags() to change a VWINL window's flags. You may need to call Vcommit_draw() after changing some flags. The cmd argument specifies how VWINL interprets the flag's argument. If cmd is VF_STO, VWINL copies the flags to the window. VF_SET sets the specified flags leaving the other bits unchanged; VF_CLR clears them. The VF_TOG command causes the specified flags to change state. The return value is the previous flag value.

```
void Vset_offset(HWND w,int x,int y)
```

Sets the offset of the specified window. When VWINL draws the VMAP to the window, it will use the offset as the VMAP's starting point (unless V_SCALE is set). The x and y parameters are in pixels.

(Cont.)

Figure 7-1. VWINL Calls (Cont.)

```
void Vget_offset(HWND w,int *x,int *y)
```
This function returns the window's offset (see Vset_offset() on the previous page).

```
HDC Vget_vdc(HWND w)
```
This function returns the VMAP dc associated with the given window.

```
int Vresize_winmap(HWND w,int width,int height)
```
Resizes the VMAP associated with the specified window. This function automatically adjuststhe window's scroll bars and handles other details.

```
int Vresize_map(VMAP *m,int wid,int hi)
```
Use Vresize_map() to change the size of a VMAP. If the VMAP is attached to a window, you will usually want to use Vresize_winmap() instead.

```
void Vset_scroll(VMAP *m,int xstep,int ystep,int xpage,int
ypage)
```
This function sets the scroll increments for a VMAP. By default, the xstep and ystep variables equal 1, and the page variables equal 10. This function causes smooth scrolling when you click the scroll bar arrows. When you scroll a page, ten pixels go by.

```
void Vclear_map(VMAP *m)
```
Use Vclear_map() to erase the entire drawing surface of a VMAP using the background color.

```
void Vclear_win(HWND *w)
```
A macro that clears the VMAP associated with a window.

(Cont.)

Figure 7-1. VWINL Calls (Cont.)

```
int Vset_stretchmode(VMAP *m,int mode)
```

Sets the VMAP's stretch mode (used when V_SCALE is set). For more about stretch modes, look up SetStretchBltMode() in the Windows API documentation. This function returns the previous stretch mode.

```
void Vdont_quit(void)
```

During a WM_CLOSE message, you may call Vdont_quit() to prevent VWINL from terminating the application.

```
int Vset_monomode(int mode)
```

Sets or clears VWINL's monochrome mode. When monochrome mode is set, all Vcreate_window() and Vcreate_map() calls create monochrome bitmaps. These bitmaps may take up less space, but they support only two colors.

```
HWND Vmodeless_dlg(HANDLE inst, LPSTR dlgname, HWND parent,
FARPROC fp)
```

This call works just like the standard CreateDialog() call except that it registers the modeless dialog with VWINL. Don't directly call CreateDialog().

```
int Vend_dlg(HWND w)
```

Use Vend_dlg() to terminate a modeless dialog created with Vmodeless_dlg().

```
Vuser_loop(int (*ul)())
```

You may install your own Windows event loop using this function. The event loop is exactly like an ordinary Windows event loop and completely replaces VWINL's default loop. You must either call this function in your main() routine or not at all.

(Cont.)

Figure 7-1. VWINL Calls (Cont.)

`Vprint_map(VMAP *)`
Invokes the standard print dialog to send the VMAP to the printer at its actual size. Vprint_map() prints multiple pages if required.

`Vsprint_map(VMAP *)`
Invokes the standard print dialog to send the VMAP, scaled to fit on a page, to the printer.

Although a VWINL callback looks like a conventional callback, there are several important differences:

- *You don't need to export a VWINL callback.*
- *You don't need a WM_PAINT case.*
- *You will handle the WM_VCREATE message instead of WM_CREATE.*
- *You will need to take special steps if you don't want a WM_DESTROY message to terminate your application.*

The main() function is the place to create your primary application window. However, don't draw to it from inside main()—VWINL hasn't properly initialized the window yet. Use the WM_VCREATE message processing in your callback routine if you want to draw to the new window. VWINL callbacks don't receive WM_CREATE messages at all; they receive only WM_VCREATE messages.

Most VWINL main() functions are just calls to Vcreate_window() (see Figure 7-2). This call mimics CreateWindow() for the most part. One difference between Vcreate_window() and CreateWindow() is the menu parameter. CreateWindow() expects a handle to a menu. Vcreate_window() takes an ASCII string or resource ID just as in LoadMenu(). If you create a child window, cast the integer child ID to an LPCSTR and pass it as if it were a menu name.

Figure 7-2. The Vcreate_window() Function in Detail

```
int Vcreate_window(char *title,DWORD style,int x,int y,
    int wid, int hi, HWND parent, LPCSTR menu,
long (*cb)(HWND,UINT,UINT,LONG),
    unsigned long vflags,HDC *dc,HWND *win, int show);
```

Parameters:

title - The windows title that appears in the caption bar.

style - The same style bits used by CreateWindow.

x - The X coordinate for the window; often CW_USEDEFAULT.

y - The Y coordinate for the window.

wid - The width of the window; often CW_USEDEFAULT.

hi - The window's height.

parent - A handle to the window's parent window. If NULL, create a top-level window.

menu - If the parent is NULL, this is a string that identifies the window's menu (or NULL if there is no menu). If the parent is not NULL, this is the child window ID (see CreateWindow).

(Cont.)

cb - Pointer to your callback function. Unlike a normal callback, you don't need to export this function or call MakeProcInstance() to get the pointer.

vflags - VWINL flags (see Figure 3).

dc - A pointer to the new window's VMAP DC (use NULL if you don't need this value).

win - Pointer to an HWND that receives the new window handle. You must supply this pointer.

show - Same as the nShow parameter in CreateWindow.

Returns: Zero if successful; nonzero on failure.

The other major difference between Vcreate_window() and CreateWindow() is the addition of a parameter for VWINL flags. These flags control the operation of VWINL (see Figure 7-3). For example, the V_SCALE flag causes VWINL to scale a VMAP to fit its window. The flags are set for each window. By default, Vcreate_window() creates both a window and VMAP simultaneously. However, you can specify the V_NOMAP flag to create a bare window. You'll then need to use Vselect_map() to associate a VMAP with the window.

Figure 7-3. VWINL Flags

V_SCALE - Causes VWINL to scale the window's VMAP to fit the window's client area. If this flag is not set, VWINL clips the VMAP to the window. When clipping, VWINL can offset the VMAP (see Vset_offset()) or automatically manage scroll bars.

V_RESIZE - Causes the window's VMAP to automatically resize when the window resizes. This causes the VMAP's size to always match the window's size.

(Cont.)

Figure 7-3. VWINL Flags (Cont.)

V_AUTOHSCROLL - When set, VWINL will automatically manage horizontal scroll bars for this window. When passed to Vcreate_window(), this flag forces the window to use the WS_HSCROLL style.

V_AUTOVSCROLL - When set, VWINL will automatically manage vertical scroll bars for this window. When passed to Vcreate_window(), this flag forces the window to use the WS_VSCROLL style.

V_NOMAP - Pass this flag to Vcreate_window to prevent VWINL from automatically creating a VMAP with the window. Presumably, you will use Vselect_map() to use a VMAP from another window or from Vcreate_map(). V_NOMAP is only meaningful during Vcreate_window().

V_NOQUIT - Ordinarily, closing a VWINL window will cause the entire application to terminate. If V_NOQUIT is set for a window, you may close the window without disturbing your applications.

V_NOFREEMAP - This flag prevents VWINL from automatically freeing the window's VMAP when you close the window. You are responsible for calling Vdestroy_map() yourself. This is useful when more than one window shares a VMAP.

V_KSCROLL - Allows VWINL to intercept scrolling keys and translate them into scroll bar events. This is especially useful in conjunction with V_AUTOHSCROLL and V_AUTOVSCROLL.

V_ZEROSELECT - If this flag is set, a Vselect_map() call will also force the display offsets to zero. This causes the top left corner of the image to be visible. If you are animating with Vselect_map(), you don't want this flag set.

V_INIT - V_INIT is an internal flag used by VWINL. Don't set this flag at home.

Note: V_SCALE is incompatible with V_RESIZE, V_AUTOHSCROLL, or V_AUTOVSCROLL. The V_RESIZE flag is not compatible with V_AUTOHSCROLL or V_AUTOVSCROLL.

During window creation, VWINL looks for a resource named VAPPICON to specify your application's icon. If you want to add accelerators, name the table VACCEL so that VWINL can find it.

When you want to draw to a VMAP, you obtain a device context using Vget_mdc() or Vget_vdc(). Use Vget_mdc() if you have a pointer to the VMAP, and Vget_vdc() if you have a window handle and want the underlying VMAP. You can freely use the device context with any GDI call. However, don't call ReleaseDC() or DestroyDC(). If you want to release the resources associated with a VMAP, call Vdestroy_map(). If more than one window is using the VMAP, the call will not do anything, so be careful to destroy VMAPs at the proper time. (VWINL attempts to destroy a window's VMAP when the window closes. More about that later.)

When you draw to a VMAP associated with a window, the changes may not be immediately visible. You can force the drawing to appear by calling Vcommit_draw(). Vselect_map() also forces the window to update.

Don't draw to a VMAP when you want to draw something transient (for example, when you drag a selection box or stretch an object in sync with the mouse). Instead, get the window's real DC (using GetDC() or another Windows call) and draw with it. Then, to restore the window to its original state, you can call Vcommit_draw().

Make sure you use a solid brush for your window backgrounds if you use the V_SCALE mode. A patterned brush will look strange when VWINL scales it to fit in the window.

Calling It Quits

When Windows sends your program a WM_DESTROY message, VWINL intercepts it. VWINL then sends your callback routine a WM_DESTROY message. If you want the program to end, you don't need to do anything. If you want the program to continue, call the Vdont_quit() function.

When VWINL detects a WM_DESTROY message, it will try to delete the window's VMAP (if it has one). Still, you should try to clean up any VMAPs you have open in your main WM_DESTROY routine. If you destroy a VMAP, detach it from its window, using Vselect_map(w,NULL), so VWINL will not try to destroy VMAP again.

Since VMAPs can be large, make sure your cleanup routine (or VWINL's) executes. For example, don't call PostQuitMessage() in response to an exit menu command. The application will terminate immediately, and you will lose memory. Instead, pass your main application window to DestroyWindow(). This will close the window, causing VWINL to cleanly terminate your program.

Fancy VWINL Tricks

VWINL uses a default event loop so you don't have to supply one.

However, if you need to use modeless dialogs, this default event loop can present a problem. There are two solutions:

- *Use Vmodeless_dlg() instead of CreateDialog(). Vmodeless_dlg() is a direct replacement for CreateDialog(), but it registers the dialog with VWINL. You can't have more than 25 modeless dialogs active at once (see the MAXMODELESS constant in VWINL.H). You can change this constant to any reasonable number. Always use Vend_dlg() to destroy a modeless dialog you have created this way.*
- *Call Vuser_loop() to install your own event loop from inside main(). Your event loop will replace the one VWINL normally uses. You can start with a copy of VWINL's event loop and modify it to suit your own purposes or roll your own from scratch.*

Breaking the Speed Limit

Although VWINL repaints the entire window on each WM_PAINT message, it still is fast. You may notice that many VWINL programs are faster than comparable ordinary programs when you resize them or restore them from an icon. An ordinary program to display text, for example, must redraw the text in the selected font each time it processes a WM_PAINT message. Windows must calculate the position of each pixel every time. VWINL programs calculate these coordinates only once when you first draw the text. On subsequent paints, the BitBlt() function rapidly transfers the pixels directly to the screen. This function often makes VWINL programs faster than their conventional counterparts.

Be careful if you use the V_SCALE flag to force VMAPs to fit into a window. The StretchBlt() call that VWINL uses to do scaling is much slower than the ordinary BitBlt(). StretchBlt() is especially slow when the window is much larger than the VMAP. You might consider making the VMAP larger than the maximum window size or restricting the window's size by intercepting the WM_MINMAXINFO message.

Of course, there is no free lunch. VWINL's increased speed and ease of use come at the expense of memory—lots of memory. If your application doesn't need color, you should consider calling Vset_monomode() in your main routine before calling Vcreate_window() or Vcreate_map(). By doing this, you will considerably reduce the number of bytes VWINL uses to store VMAPs (unless you are on a monochrome display anyway; then it won't make any difference).

A Practical Example

Listings 7-4 to 7-6 show a complete text-browser program, BROWSE. BROWSE displays an ASCII file with scroll bars if required. You can print the file by using BROWSE's menu. In reality, TWIN (see Chapter 4) would probably be a better choice for writing a program like BROWSE. However, since BROWSE demonstrates some strengths and weaknesses of VWINL, it is worth studying.

All of BROWSE's functionality is embodied in the open_file() function. Here, the standard-file open dialog

retrieves a file name. After opening the file, BROWSE makes two passes through it. First, the program computes the number of lines in the file and the width of the longest line. Next, it creates a VMAP that is large enough to hold the entire text. Finally, BROWSE reads the file and prints the text into the new VMAP. Now, VWINL handles all display and scrolling of the text buffer. Although an edit control can do this just as well, the VWINL program could easily contain graphics or multiple fonts.

The do_menu() function contains one line that prints the entire file:

```
Vprint_map(Vget_map(w));
```

This invokes the standard print dialog and writes the VMAP to the printer. VWINL scales the image so that a logical inch on the screen equates to an inch on the printed page. The printing routine will only create multiple pages for the y dimension of the VMAP; images too wide to print will be truncated.

Instead of Vprint_map(), you can call Vsprint_map() to print a VMAP. This call scales the image to force it to fit on the page. This doesn't work very well for BROWSE, but it is useful for many graphics programs.

The usr_cb() function is the VWINL callback. If BROWSE were content to use the default scrolling behavior, this function would be absurdly simple (it would only handle WM_COMMAND events). However, BROWSE adjusts

the page scroll size when the window changes size. To do this, usr_cb() must process WM_SIZE messages. A simple call to Vset_scroll() modifies the page size.

Limits

BROWSE illustrates two VWINL limitations: memory size and printer resolution. If a file is too large for memory or too large to fit in a bitmap, BROWSE can't display it.

If you print a file from BROWSE, you will quickly see the printer resolution problem. BROWSE essentially prints a screen dump of its text. The fonts are not high resolution as they would be if you printed text directly to the printer (see Figure 7-4).

For a page-oriented graphics program, you could make a VMAP that is the same size as a printed page and scale it to fit inside the window. Then when you use Vprint_map(), the printout would match the printer's resolution. However, BROWSE shouldn't display the entire file inside a window—it would be too difficult to read.

Figure 7-4. Browse Printed Output

```
#define MENU_OPEN 101
#define MENU_ABOUT 102
#define MENU_EXIT 103
#define MENU_PRINT 104
```

On the positive side, BROWSE does a lot of work with just a few hundred lines of code, thanks to the VWINL

library. With surprisingly little effort, you can write quite complex applications using VWINL.

Is VWINL for You?

VWINL can simplify many types of Windows programs. In the future, Windows (or another GUI) may support VMAP-style programming. With built-in support, the VMAPs could be stored as a sparse array and perhaps be compressed. Until then, you can use VWINL to experiment with this technique. You will notice that the source code of a VWINL program more closely resembles an ordinary DOS graphics program than a Windows application.

VWINL will work with Windows NT and Win32s. Since these 32-bit environments offer improved memory management, VWINL makes even more sense for them. The next time you write a Windows program, try VWINL and see how simple a Windows program can be.

If you want to know how VWINL works, continue reading. Otherwise, you may skip the remainder of this chapter.

How Does It Do That?

VMAPs take advantage of two special Windows features: bitmaps and memory device contexts. All GDI (drawing) functions operate on a device context (DC). Typically,

output to a DC appears on a window. VWINL uses the CreateCompatibleDC() function to create a memory device context. A memory device context must have a bitmap associated with it via a SelectObject() call. Drawing operations you perform against the memory DC don't appear anywhere on the screen. Instead, the drawing operations act on the associated bitmap.

Windows only allows bitmaps to be 65535 by 65535. VMAPs can't exceed this size. If you use the autoscroll feature, you must restrict your VMAPs to 32767 by 32767. Windows doesn't allow scroll bar ranges to exceed 32K.

The key to VWINL is its default WM_PAINT handler, do_paint(). This routine copies the bitmap from the window's VMAP to the client area. If the V_SCALE flag is set, VWINL uses StretchBlt() to scale the image as it copies it. Otherwise, the BitBlt() function simply copies the bitmap.

If the bitmap is smaller than the window's client area, do_paint() erases the region outside the bitmap using the PatBlt() function. This erasure ensures a consistent background when you resize the window.

Windows may send your program a WM_PAINT message for many different reasons. When you iconify your window and restore it, you'll get a WM_PAINT message. You'll also get a WM_PAINT when another window obscures yours and then moves to expose it again. The Vcommit_draw() function is a macro that calls InvalidateRect(). The InvalidateRect() call also generates WM_PAINT messages.

Listing 7-1. SIMPLE.C

```
/*******************************************************
 *                                                     *
 * File: SIMPLE.C                                      *
 *                                                     *
 * Very simple VWINL program.                          *
 *                                                     *
 * Required to Compile:                                *
 * VWIN.C VWINL.H SIMPLE.C                             *
 *                                                     *
 * To compile with Borland C:                          *
 *   bcc -W simple.c vwinl.c                           *
 * To compile with Microsoft C:                        *
 *   cl -GA simple.c vwinl.c vwinl.def                 *
 *                                                     *
 *******************************************************/

#include "vwinl.h"

/* User's callback */
long usr_cb(HWND hWnd, UINT Message,
            UINT wParam, LONG lParam)
  {
  if (Message == WM_VCREATE)
      {
      Vresize_winmap(hWnd, 150, 100);
      /* Why limit ourselves? */
      TextOut(Vget_vdc(hWnd), 5, 50, "Hello Universe", 14);
      Vcommit_draw(hWnd);
```

```
        }
    return DefWindowProc(hWnd, Message, wParam, lParam);
    }

/* Start here */
int main(HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpszCmdLine, int nCmdShow)
    {
    HWND hWnd;
    /* Create window or die */
    if (Vcreate_window("Simple Test Program",
              WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0,
              CW_USEDEFAULT, 0, NULL, NULL,
              usr_cb, V_SCALE,
              NULL, &hWnd, nCmdShow))
        {
        MessageBox(NULL, "Can't create window", NULL, MB_OK);
        return 0;
        }
    return 1;
    }
```

Listing 7-2. VWINL.H

```
/*******************************************************
 *                                                     *
 * File: VWINL.H                                       *
 *                                                     *
 * Virtual Window Library                              *
 *                                                     *
```

```
 * Required to Compile:                                        *
 * VWIN.C VWIN.H VWIN.DEF + your program                       *
 *                                                             *
 ***************************************************************/
#ifndef _VWINL_H
#define _VWINL_H
#include <windows.h>

/* Maximum # of modeless dialogs */
#define MAXMODELESS 25

#ifndef WIN32
#define APIENTRY FAR PASCAL
/* Check for message cracker definition if 1 is there
 * assume they all are */
#ifndef GET_WM_VSCROLL_CODE
#define GET_WM_VSCROLL_CODE(w,l) (w)
#define GET_WM_HSCROLL_CODE(w,l) (w)
#define GET_WM_VSCROLL_HWND(w,l) ((HWND)HIWORD(l))
#define GET_WM_HSCROLL_HWND(w,l) ((HWND)HIWORD(l))
#define GET_WM_VSCROLL_POS(w,l)  (LOWORD(l))
#define GET_WM_HSCROLL_POS(w,l)  (LOWORD(l))
#endif

#endif /* End of non-WIN32 definitions */

/* Flags */

/* V_SCALE doesn't make sense with V_RESIZE,
V_AUTOHSCROLL,
```

```
 * or V_AUTOVSCROLL. V_RESIZE, doesn't make sense with any
 * of the AUTOXSCROLL flags. V_NOMAP is only valid during
 * window creation. V_INIT is reserved for internal use. */

#define V_SCALE 1L
#define V_RESIZE 2L
#define V_AUTOHSCROLL 4L
#define V_AUTOVSCROLL 8L
#define V_NOMAP 0x10L
#define V_NOQUIT 0x20L
#define V_NOFREEMAP 0x40L
#define V_KSCROLL 0x80L
#define V_ZEROSELECT 0x100L
#define V_INIT 0x80000000L

/* Flags for Vset_flags() */
#define VF_STO 0
#define VF_SET 1
#define VF_CLR 2
#define VF_TOG 3

#define WM_VCREATE WM_USER
#define Vcommit_draw(w) InvalidateRect(w,NULL,FALSE)
/* Get VMAP dc */
#define Vget_mdc(m) ((m)->dc)
#define Vget_stretchmode(m) ((m)->stretch_mode)
#define Vclear_win(w) Vclear_map(Vget_map(w))

long APIENTRY VWndProc(HWND, UINT, UINT, LONG);
```

```
int main(HANDLE hInstance, HANDLE hPrevInstance,
          LPSTR lpszCmdLine, int nCmdShow);

typedef struct
  {
  HBITMAP bitmap;
  HDC dc;
  HBITMAP defbitmap;
  int xstep, ystep, xpage, ypage;
  unsigned refct;
  int stretch_mode;
  }       VMAP;


typedef struct
  {
  VMAP *map;
/* dimensions of bitmap (not window) */
  unsigned int width;
  unsigned int height;
/* flags */
  unsigned long flags;
/* display offset */
  unsigned int xoff;
  unsigned int yoff;
  long (*cb) (HWND, UINT, UINT, LONG);
  }       MEMWINFO;


void Vget_info(HWND w, MEMWINFO * info);
```

```
VMAP *Vget_map(HWND w);
VMAP *Vcreate_map(int wid, int hi);
void Vdestroy_map(VMAP * map);
VMAP *Vselect_map(HWND w, VMAP * new);
unsigned long Vset_flags(HWND w, unsigned long flags,
                         int cmd);
void Vset_offset(HWND w, int x, int y);
void Vget_offset(HWND w, int *x, int *y);
int Vcreate_window(char *title, DWORD style, int x, int y,
                   int wid, int hi, HWND parent,
                   LPCSTR menu,
long (*cb) (HWND, UINT, UINT, LONG),
unsigned long vflags, HDC * dc, HWND * win, int show);
HDC Vget_vdc(HWND w);
int Vresize_winmap(HWND w, int wid, int hi);
int Vresize_map(VMAP * m, int wid, int hi);
void Vdont_quit(void);
void Vset_scroll(VMAP * m, int xstp, int ystp, int xpg,
                 int ypg);
void Vclear_map(VMAP * m);
int Vset_stretchmode(VMAP * m, int mode);
int Vset_monomode(int mode);
HWND Vmodeless_dlg(HANDLE inst, LPSTR title, HWND parent,
                   FARPROC fp);
int Vend_dlg(HWND w);
/* Set user loop -- must call in main() or not at all */
void Vuser_loop(int (*ul) ());
int Vprint_map(VMAP *);
int Vsprint_map(VMAP *);
```

```
#ifndef __BORLANDC__
#define main vwin_main
int vwin_main(HANDLE, HANDLE, LPSTR, int);
#else
int main(HANDLE, HANDLE, LPSTR, int);
#endif

#endif
```

Listing 7-3. VWINL.DEF

```
NAME VWINAPP
DESCRIPTION 'by Al Williams'
CODE MOVEABLE PRELOAD
DATA MOVEABLE MULTIPLE PRELOAD
HEAPSIZE 8192
STACKSIZE 8192
EXPORTS VWndProc
```

Listing 7-4. BROWSE.C

```
/*******************************************************
 *                                                     *
 * File: BROWSE.C                                      *
 *                                                     *
 * File browser that uses VWINL library.               *
 *                                                     *
 * Required to Compile:                                *
 * BROWSE.C VWIN.C VWIN.H VWIN.RC BROWSE.RC VWIN.DEF   *
 * BROWSE.H MAKEFILE                                   *
 *                                                     *
```

```
 ****************************************************/


#include "vwinl.h"
#include "browse.h"
#include <stdio.h>
#include <string.h>
#include <commdlg.h>

#define TOPMARGIN 5
#define LEFTMARGIN 5


/* Filters for common dialog */
char filefilter[] = "All files (*.*)\0*.*\0"
   "Text files (*.txt)\0*.c\0"
   "C files (*.c)\0*.c\0"
   "H files (*.h)\0*.h\0"
   "C++ files (*.cpp)\0*.cpp\0"
   "DEF files (*.def)\0*.def\0"
   "RC files (*.rc)\0*.rc\0\0";

/* Scroll sizes */
int xscroll, yscroll, xpage, ypage;


/* This routine computes the text extent and properly
 * modifies the wid and hi variables for the caller. You
 * could use GetTextExtent() inline for this except that
 * WIN32 requires a slightly different approach */
```

```
void text_extent(HDC dc, char *s, UINT ct, UINT * wid,
                 UINT * hi)
  {
#ifdef WIN32
  SIZE extent;
  GetTextExtentPoint(dc, s, ct, &extent);
  if (wid)
    *wid = max(*wid, (UINT) extent.cx);
  if (hi)
    *hi += extent.cy;
#else
  DWORD extent;
  extent = GetTextExtent(dc, s, ct);
  if (wid)
    *wid = max(*wid, LOWORD(extent));
  if (hi)
    *hi += HIWORD(extent);
#endif
  }

/* Open file -- this does *ALL* the work! */
void open_file(HWND w)
  {
  OPENFILENAME ofile;
  FILE *in;
  VMAP *map;
  HCURSOR cursor;
  char fn[256], ft[256];
  int err;
  unsigned wid = 0, hi = 0, y = TOPMARGIN;
```

```
unsigned scr_wid = 0, scr_hi = 0;
HDC dc;
/* Use common dialog to get file name */
memset(&ofile, 0, sizeof(OPENFILENAME));
*fn = *ft = '\0';
ofile.lStructSize = sizeof(OPENFILENAME);
ofile.hwndOwner = w;
ofile.lpstrFilter = filefilter;
ofile.nFilterIndex = 1;
ofile.lpstrFile = fn;
ofile.nMaxFile = sizeof(fn);
ofile.lpstrFileTitle = ft;
ofile.nMaxFileTitle = sizeof(ft);
ofile.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
   OFN_FILEMUSTEXIST;
if (!(err = GetOpenFileName(&ofile)) ||
    !(in = fopen(fn, "r")))
    {
    /* If err is FALSE then might just be a cancel
     * CommDlgExtendedError returns 0 if it was a
       cancel */
    if (err || CommDlgExtendedError())
       MessageBox(w, "File open error", NULL,
                  MB_ICONSTOP | MB_OK);
    return;
    }
/* Set window title */
wsprintf(fn, "BROWSE - %s", (char far *) ft);
SendMessage(w, WM_SETTEXT, 0, (LONG) fn);
/* Wait cursor */
```

```
  cursor = SetCursor(LoadCursor(NULL, IDC_WAIT));
  /* Make window have no map */
  map = Vselect_map(w, NULL);
/* If there was a map here, kill it */
  if (map)
    Vdestroy_map(map);
  dc = GetDC(w);
  /* Read file to compute width and length of VMAP */
  while (fgets(ft, sizeof(ft), in))
      {
      int l;
      ft[l = (strlen(ft) - 1)] = '\0';
      /* Make empty lines have one blank */
      if (!l)
        {
        ft[0] = ' ';
        ft[1] = '\0';
        l = 1;
        }
      /* compute extents */
      text_extent(dc, ft, l, &wid, &hi);
      }
  ReleaseDC(w, dc);
  /* create map of the right size */
  map = Vcreate_map(wid + 2 * LEFTMARGIN,
          hi + 2 * TOPMARGIN);
  if (!map)
      {
      MessageBox(w, "File too large!", NULL,
                 MB_ICONSTOP | MB_OK);
```

```
        SendMessage(w, WM_SETTEXT, 0, (LONG) "BROWSE");
        SetCursor(cursor);
        return;
        }
    /* Start file over */
    rewind(in);
    /* Select map into window */
    Vselect_map(w, map);
    dc = Vget_vdc(w);
    /* Set scroll dimensions */
    text_extent(dc, " ", 1, &scr_wid, &scr_hi);
    Vset_scroll(map, xscroll = scr_wid, yscroll = scr_hi,
            xpage - scr_wid, ypage - scr_hi);
    /* Read file into VMAP */
    while (fgets(ft, sizeof(ft), in))
        {
        int l;
        ft[l = (strlen(ft) - 1)] = '\0';
        /* Empty lines = 1 space */
        if (!l)
          {
          ft[0] = ' ';
          ft[1] = '\0';
          l = 1;
          }
/* Instead of TextOut(), you could use any GDI calls here */
        TextOut(dc, LEFTMARGIN, y, ft, l);
        text_extent(dc, ft, l, NULL, &y);
        }
    /* Done with file */
```

```
  fclose(in);
  /* Restore cursor */
  SetCursor(cursor);
  /* Force drawing update */
  Vcommit_draw(w);
  }

/* Do menu functions */
void do_menu(HWND w, UINT wParam)
  {
  /* LOWORD for WIN32 compatibility */
  switch (LOWORD(wParam))
      {
    case MENU_ABOUT:
      MessageBox(w,
     "Browse -- A VWINL File Browser by Al Williams",
       "About BROWSE", MB_OK);
      break;

    case MENU_EXIT:
      DestroyWindow(w);
      break;

    case MENU_OPEN:
      open_file(w);
      break;

    case MENU_PRINT:
      Vprint_map(Vget_map(w));
      break;
```

```
      }
  }

/* Our callback */
long usr_cb(HWND w, UINT Message,
            UINT wParam, LONG lParam)
  {
  switch (Message)
      {
      VMAP *map;
      /* Handle menu commands */
    case WM_COMMAND:
      do_menu(w, wParam);
      break;

      /* If window resizes, recompute size of a scrolling
       * page */
    case WM_SIZE:
      xpage = LOWORD(lParam);
      ypage = HIWORD(lParam);
      map = Vget_map(w);
      if (map)
        Vset_scroll(map, xscroll, yscroll,
                    xpage - xscroll, ypage - yscroll);
      break;

    default:
      return DefWindowProc(w, Message, wParam, lParam);
      }
```

```
  return 0;
  }


/* VWINL main function */
int main(HANDLE hInstance, HANDLE hPrevInstance,
        LPSTR lpszCmdLine, int nCmdShow)
  {
  HWND w;
  /* Save space -- use monochrome mode */
  Vset_monomode(TRUE);
  if (Vcreate_window("BROWSE",
          WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0,
          CW_USEDEFAULT, 0, NULL, "mainmenu",
        usr_cb, V_AUTOHSCROLL | V_AUTOVSCROLL |
      V_NOMAP | V_KSCROLL, NULL, &w, nCmdShow))
      {
      MessageBox(NULL, "Can't create window", NULL, MB_OK);
      return 0;
      }
  return 1;
  }
```

Listing 7-5. BROWSE.H

```
*************************************************
 *                                               *
 * File: BROWSE.H                                *
 *                                               *
 * Header for BROWSE program.                    *
 *                                               *
```

```
 * Required to Compile:                                        *
 * BROWSE.C VWIN.C VWIN.H VWIN.RC BROWSE.RC VWIN.DEF *
 * BROWSE.H MAKEFILE                                           *
 *                                                             *
 ***************************************************/
#define MENU_OPEN 101
#define MENU_ABOUT 102
#define MENU_EXIT 103
#define MENU_PRINT 104
/***
```

Listing 7-6. BROWSE.RC

```
/* Resources for BROWSE.C */
#include "browse.h"
#include "vwin.rc"

mainmenu MENU PRELOAD
  BEGIN
  POPUP "&File"
    BEGIN
    MENUITEM "&Open",  MENU_OPEN
    MENUITEM "&Print", MENU_PRINT
    MENUITEM "&About", MENU_ABOUT
    MENUITEM "&Exit",  MENU_EXIT
    END
  END


vappicon ICON "vwin.ico" PRELOAD
```

Listing 7-7. VWIN.RC

```
/*******************************************************
 *                                                     *
 * File: VWIN.RC                                       *
 *                                                     *
 * Virtual Window Library                              *
 *                                                     *
 * Required to Compile:                                *
 * VWIN.C VWIN.H VWIN.DEF + your program               *
 *                                                     *
 *******************************************************/
#include <windows.h>

PrintDialogBox   DIALOG 100,100,120,40
  STYLE WS_POPUP|WS_SYSMENU|WS_VISIBLE|WS_DLGFRAME|WS_CAP-
TION
  CAPTION "Printing..."
BEGIN
  CTEXT "Now printing", -1, 4, 6, 120, 12
  DEFPUSHBUTTON "Cancel", IDCANCEL, 44,22,32,14
END
```

Listing 7-8. VWIN.C

```
/*******************************************************
 *                                                     *
 * File: VWIN.C                                        *
 *                                                     *
 * Virtual Window Library                              *
 *                                                     *
 * Required to Compile:                                *
```

```
 * VWIN.C VWIN.H VWIN.DEF + your program                    *
 *                                                          *
 ************************************************************/
#include "vwinl.h"
#include <windowsx.h>
#include <commdlg.h>
#include <string.h>

/* Local prototypes */
static void do_paint(HWND);
long WINAPI VWndProc(HWND w, UINT Message,
                     UINT wParam, LONG lParam);
static void save_info(HWND w, MEMWINFO * info);
static void set_sb(HWND w, MEMWINFO * minfo, UINT wid,
                   UINT hi, int save);
static void scrollit(HWND w, MEMWINFO * minfo, int type,
                     WORD code, HWND sb, WORD pos);
static void key_scroll(HWND w, UINT key);


/* Global variables */
static HANDLE Hinstance; /* Our instance */
static int monomode;  /* Make mono bitmaps? */
/* Flag to tell us if user allows us to quit */
static int V_quit = 0;
static HWND dlgtbl[MAXMODELESS];
static int (*user_loop) ();

/* VWINL WinMain -- this calls our main() function. If
 * main() returns 0, then we abort. */
```

```
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdLine, int nCmdShow)
  {
  WNDCLASS wndClass;
  MSG msg;
  HACCEL haccel;

  /* Register window class style if first instance of this
   * program. */
  Hinstance = hInstance;
  if (!hPrevInstance)
      {
      /* NOTE: VWINL assumes the window will have a common
       * DC! Don't use CS_OWNDC or CS_PARENTDC unless you
       * know what you are getting into! */
      wndClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
      wndClass.lpfnWndProc = (WNDPROC) VWndProc;
      wndClass.cbClsExtra = 0;
      wndClass.cbWndExtra =
      sizeof(MEMWINFO) + (sizeof(MEMWINFO) % 2);
      wndClass.hInstance = hInstance;
      wndClass.hIcon = LoadIcon(hInstance, "vappicon");
      wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
      wndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
      wndClass.lpszMenuName = NULL;
      wndClass.lpszClassName = "VWINL";
      if (!RegisterClass(&wndClass))
         return FALSE;
```

```
    }

if (!main(hInstance, hPrevInstance,
     lpszCmdLine, nCmdShow))
   return FALSE;
if (user_loop)
    {
    return
       user_loop(hInstance, hPrevInstance,
          lpszCmdLine, nCmdShow);
    }
/* Try to load an accelerator -- no big deal if it isn't
 * there */
haccel = LoadAccelerators(hInstance, "VACCEL");

/* Sorta ordinary message loop -- will translate
 * accelerators only if appropriate */
while (GetMessage(&msg, NULL, 0, 0))
    {
    int i;
    int dlgflag = 0;
    /* look for modeless dialogs */
    for (i = 0; i < MAXMODELESS; i++)
     {
     if (dlgtbl[i] && IsDialogMessage(dlgtbl[i], &msg))
       {
       dlgflag = 1;
       break;
       }
     }
```

```
      if (dlgflag)
         continue;
      if (!haccel ||
          !TranslateAccelerator(msg.hwnd, haccel, &msg))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }
      }
    return 0;
    }



/* Main Window procedure */
long APIENTRY VWndProc(HWND w, UINT Message,
                       UINT wParam, LONG lParam)
  {
  MEMWINFO minfo;
  Vget_info(w, &minfo);
  switch (Message)
      {
      /* Don't let user see WM_CREATE -- we aren't ready
       * for him yet */
    case WM_CREATE:
      return DefWindowProc(w, Message, wParam, lParam);

      /* Always do painting and don't tell user */
    case WM_PAINT:
      do_paint(w);
```

```
    return 0;

  case WM_SIZE:
    /* Catch 1st size */
    if (minfo.flags & V_INIT)
      {
      if ((LOWORD(lParam) != minfo.width ||
      HIWORD(lParam) != minfo.height) && minfo.map)
       {
       Vresize_winmap(w, LOWORD(lParam), HIWORD(lParam));
       }
     minfo.flags &= ~V_INIT;
     save_info(w, &minfo);
     break;
     }
    /* Set scrollbars */
    if (minfo.map)
      {
   /* Everytime you toggle a scroll bar from on to
    * off or off to on you get a WM_SIZE message!
    * This little state machine lets us turn off the
    * bars without getting into an endless loop. */

      static sizelock = 0;
      RECT r;
      if (sizelock == 1)
        return 0;
      if (sizelock != 2)
        {
```

```
        /* Turn off both scroll bars so set_sb() can
         * use the whole client area if that's what
         * it needs */
          sizelock = 1;
          if (minfo.flags & V_AUTOVSCROLL)
            SetScrollRange(w, SB_VERT, 0, 0, FALSE);
          if (minfo.flags & V_AUTOHSCROLL)
            SetScrollRange(w, SB_HORZ, 0, 0, FALSE);
         /* Size might have changed, so reset it */
          GetClientRect(w, &r);
          lParam = MAKELONG(r.right - r.left,
                            r.bottom - r.top);
          }
      /* Turn scroll bars on or off */
      sizelock = 2;
      set_sb(w, &minfo, LOWORD(lParam), HIWORD(lParam),
             TRUE);
      /* Size might have changed again, so reset it */
      GetClientRect(w, &r);
      lParam = MAKELONG(r.right - r.left, r.bottom -
r.top);
      sizelock = 0;
      }
       /* Handle V_RESIZE if active */
    if (minfo.flags & V_RESIZE && minfo.map)
      {
      Vresize_winmap(w, LOWORD(lParam), HIWORD(lParam));
      }
    break;

       /* Scroll cases */
```

```
case WM_VSCROLL:
  if (minfo.flags & V_AUTOVSCROLL)
    scrollit(w, &minfo, SB_VERT,
             GET_WM_VSCROLL_CODE(wParam, lParam),
             GET_WM_VSCROLL_HWND(wParam, lParam),
             GET_WM_VSCROLL_POS(wParam, lParam));
  break;

case WM_HSCROLL:
  if (minfo.flags & V_AUTOHSCROLL)
      scrollit(w, &minfo, SB_HORZ,
               GET_WM_HSCROLL_CODE(wParam, lParam),
               GET_WM_HSCROLL_HWND(wParam, lParam),
               GET_WM_HSCROLL_POS(wParam, lParam));
   break;

case WM_KEYDOWN:
  if (minfo.flags & V_KSCROLL)
     key_scroll(w, wParam);
  break;

case WM_DESTROY:
  /* pass to user if V_NOQUIT set */
  if (minfo.flags & V_NOQUIT)
     break;
  V_quit = 1;
  /* Pass to user, if V_quit is set, go ahead and kill
   * ourselves */
  if (!minfo.cb(w, Message, wParam, lParam) && V_quit)
     {
```

```
      /* Clean up window's resources here... */
          if (minfo.map && minfo.map->refct &&
             !(minfo.flags & V_NOFREEMAP))
             {
             minfo.map->refct--;
             Vdestroy_map(minfo.map);
             }
          PostQuitMessage(0);
          return 0;
          }
       return 0;
       }
   /* pass to user's callback */
   if (minfo.cb)
     return minfo.cb(w, Message, wParam, lParam);
   else
     return DefWindowProc(w, Message, wParam, lParam);

   }


/* Create a VWIN -- see text for description Returns 0 for
 * success */
int Vcreate_window(char *title, DWORD style, int x, int y,
            int wid, int hi, HWND parent, LPCSTR menu,
            long (*cb) (HWND, UINT, UINT, LONG),
            unsigned long vflags, HDC * dc, HWND * win,
            int show)
   {
   HWND w;
```

```
HMENU hMenu = NULL;
RECT r;
MEMWINFO minfo;
if (menu && !parent)
  hMenu = LoadMenu(Hinstance, menu);
else
  hMenu = (HMENU) menu;
/* Auto set scroll style */
if (vflags & V_AUTOHSCROLL)
  style |= WS_HSCROLL;
if (vflags & V_AUTOVSCROLL)
  style |= WS_VSCROLL;
memset(&minfo, 0, sizeof(MEMWINFO));
minfo.flags = vflags | V_INIT;
minfo.cb = cb;
w = *win = CreateWindow("VWINL", title, style, x, y,
           wid, hi, parent, hMenu, Hinstance, NULL);
if (!*win)
  return 1;
save_info(*win, &minfo);

SetScrollRange(*win, SB_HORZ, 0, 0, TRUE);
SetScrollRange(*win, SB_VERT, 0, 0, TRUE);


GetClientRect(w, &r);

/* create DC the same size */
minfo.xoff = minfo.yoff = 0;
minfo.width = r.right - r.left;
```

```
  minfo.height = r.bottom - r.top;
  if ((minfo.flags & V_NOMAP) == 0)
      {
      minfo.map = Vcreate_map(minfo.width, minfo.height);
      if (!minfo.map)
         return 2;
      if (dc)
         *dc = minfo.map->dc;
      }
  else
    minfo.map = NULL;
  /* Store cb and other data in extra words */
  save_info(w, &minfo);

  /* finish up */
  ShowWindow(*win, show);
  /* Call user's callback with WM_VCREATE */
  minfo.cb(*win, WM_VCREATE, 0, 0);
  UpdateWindow(*win);
  return 0;
  }


/* Make a VMAP -- respects monomode flag */
VMAP *Vcreate_map(int wid, int hi)
  {
  HDC dc;
  VMAP *bm;
  HWND w;
  w = GetDesktopWindow();  /* any window will do */
```

```
dc = GetDC(w);
if (!dc)
  return NULL;
/* temp use of malloc */
bm = (VMAP *) LocalAlloc(LPTR, sizeof(VMAP));
if (!bm)
  return NULL;
bm->dc = CreateCompatibleDC(dc);
/* release Desktop DC */
ReleaseDC(w, dc);
if (!bm->dc)
    {
    LocalFree((HLOCAL) bm);
    return NULL;
    }
/* Make the bitmap */
/* NOTE: Windows won't let you make an arbitrary colored
 * bitmap. You must have a device that corresponds to the
 * color-size of the bitmap */
if (!monomode)
  bm->bitmap = CreateBitmap(wid, hi,
           GetDeviceCaps(dc, PLANES),
           GetDeviceCaps(dc, BITSPIXEL), NULL);
else     /* mono */
  bm->bitmap = CreateBitmap(wid, hi, 1, 1, NULL);

if (!bm->bitmap)
    {
    DeleteDC(bm->dc);
    LocalFree((HLOCAL) bm);
```

```
        return NULL;
        }
    /* Note: This is supposed to be in .1 mm units, but since
     * no one else uses it, who cares! */
#ifndef WIN32
    SetBitmapDimension(bm->bitmap, wid, hi);
#else
    SetBitmapDimensionEx(bm->bitmap, wid, hi, NULL);
#endif
    bm->defbitmap = SelectObject(bm->dc, bm->bitmap);
    if (!bm->defbitmap)
        {
        DeleteDC(bm->dc);
        LocalFree((HLOCAL) bm);
        return NULL;
        }
    /* Set default stuff */
    bm->xstep = bm->ystep = 1;
    bm->xpage = bm->ypage = 10;
    bm->refct = 1;
    bm->stretch_mode = BLACKONWHITE;
    Vclear_map(bm);
    return bm;
    }

/* Free up a valid VMAP if its refct is 1 or 0 */
void Vdestroy_map(VMAP * map)
    {
    if (map->refct > 1)
```

```
    return;
  SelectObject(map->dc, map->defbitmap);
  DeleteObject(map->bitmap);
  DeleteDC(map->dc);
  LocalFree((HLOCAL) map);
  }

/* Associate a new map (or NULL for no map) with a window
 * -- returns the old map */
VMAP *Vselect_map(HWND w, VMAP * new)
  {
  MEMWINFO minfo;
  VMAP *rc;
#ifndef WIN32
  DWORD dims;
#else
  SIZE dims;
#endif
  RECT r;
  GetClientRect(w, &r);
  Vget_info(w, &minfo);
  rc = minfo.map;
  minfo.map = new;
  if (rc)
    rc->refct--;
  /* NULL is OK here in which case we don't do much */
  if (new)
      {
      new->refct++;
#ifndef WIN32
```

```
        dims = GetBitmapDimension(minfo.map->bitmap);
        minfo.width = LOWORD(dims);
        minfo.height = HIWORD(dims);
#else
        GetBitmapDimensionEx(minfo.map->bitmap, &dims);
        minfo.width = dims.cx;
        minfo.height = dims.cy;
#endif
        }
    else
        {
        minfo.width = minfo.height = 0;
        }
    if (minfo.flags & V_ZEROSELECT)
      minfo.xoff = minfo.yoff = 0;
    save_info(w, &minfo);
    set_sb(w, &minfo, r.right - r.left,
            r.bottom - r.top, TRUE);
    Vcommit_draw(w);
    return rc;
    }


/* Set stretch mode for map -- returns old mode */
int Vset_stretchmode(VMAP * m, int mode)
    {
    int rv;
    rv = m->stretch_mode;
    m->stretch_mode = mode;
    return rv;
```

```
  }

/* Sets global monomode flag which causes VWINL to create
 * monochrome bitmaps to save space. Returns old value (of
 * course) */
int Vset_monomode(int mode)
  {
  int rv;
  rv = monomode;
  monomode = mode;
  return rv;
  }


/* Get window's map */
VMAP *Vget_map(HWND w)
  {
  MEMWINFO minfo;
  Vget_info(w, &minfo);
  return minfo.map;
  }

/* Erase a map's surface */
void Vclear_map(VMAP * m)
  {
  HBRUSH brush;
  unsigned int wid, hi;
#ifndef WIN32
  DWORD dims;
#else
```

```
  SIZE dims;
#endif
  /* We store bitmap dimension this way. Units are pixels
   * contrary to the .1mm convention */
#ifndef WIN32
  dims = GetBitmapDimension(m->bitmap);
  wid = LOWORD(dims);
  hi = HIWORD(dims);
#else
  GetBitmapDimensionEx(m->bitmap, &dims);
  wid = dims.cx;
  hi = dims.cy;
#endif
  /* make background brush */
  brush = CreateSolidBrush(GetBkColor(m->dc));
  brush = SelectObject(m->dc, brush);
  /* Brush area */
  PatBlt(m->dc, 0, 0, wid, hi, PATCOPY);
  DeleteObject(SelectObject(m->dc, brush));
  }


/* Change a window's VWINL flags -- this only makes sense
 * for some flags. For example, V_NOMAP is meaningless
 * here. If you ever plan to set V_AUTOHSCROLL or
 * V_AUTOVSCROLL, make sure to set the scroll bar style
 * flags during Vcreate_window (this happens automatically
 * when you set V_AUTOxSCROLL during the create.
 *
 * Returns old flag value */
```

```
unsigned long Vset_flags(HWND w, unsigned long flags,
                         int cmd)
  {
  unsigned long rv;
  MEMWINFO minfo;
  RECT r;
  Vget_info(w, &minfo);
  GetClientRect(w, &r);
  rv = minfo.flags;
  switch (cmd)
      {
    case VF_SET:
      minfo.flags |= flags;
      break;
    case VF_CLR:
      minfo.flags &= ~flags;
      break;
    case VF_TOG:
      minfo.flags ^= flags;
      break;
    default:
      minfo.flags = flags;
      break;
      }
  save_info(w, &minfo);
  if (((rv & V_AUTOHSCROLL) ^ (minfo.flags &
V_AUTOHSCROLL))
      || ((rv & V_AUTOVSCROLL) ^
         (minfo.flags & V_AUTOVSCROLL)))
      {
```

```
        /* scroll changed */
        if (!(minfo.flags & V_AUTOHSCROLL))
           {
           minfo.xoff = 0;
           SetScrollRange(w, SB_HORZ, 0, 0, TRUE);
           }
        if (!(minfo.flags & V_AUTOVSCROLL))
           {
           minfo.yoff = 0;
           SetScrollRange(w, SB_VERT, 0, 0, TRUE);
           }
        save_info(w, &minfo);
        if (minfo.flags & (V_AUTOHSCROLL | V_AUTOVSCROLL))
           set_sb(w, &minfo, r.right - r.left,
                  r.bottom - r.top, TRUE);
        Vcommit_draw(w);
        }
     return rv;
     }

/* Set the VMAP offset in pixels */
void Vset_offset(HWND w, int x, int y)
     {
     MEMWINFO minfo;
     Vget_info(w, &minfo);
     minfo.xoff = x;
     minfo.yoff = y;
     save_info(w, &minfo);
     }
```

```
/* Read the VMAP pixel offsets */
void Vget_offset(HWND w, int *x, int *y)
  {
  MEMWINFO minfo;
  Vget_info(w, &minfo);
  if (x)
    *x = minfo.xoff;
  if (y)
    *y = minfo.yoff;
  }


/* Get window's VMAP dc */
HDC Vget_vdc(HWND w)
  {
  MEMWINFO minfo;
  Vget_info(w, &minfo);
  return minfo.map->dc;
  }


/* Resize a map Returns 0 if OK */
int Vresize_map(VMAP * m, int wid, int hi)
  {
  VMAP *newmap;
  int oldstate;
  HBITMAP oldbm = m->bitmap;
  newmap = Vcreate_map(wid, hi);
  if (!newmap)
    return 1;
```

```
  oldstate = SetMapMode(m->dc, MM_TEXT);
  BitBlt(newmap->dc, 0, 0, wid, hi, m->dc, 0, 0, SRCCOPY);
  /* copy the right parts from newmap */
  m->bitmap = newmap->bitmap;
  SetMapMode(m->dc, oldstate);
  /* Except for bitmap! */
  SelectObject(newmap->dc, newmap->defbitmap);
  SelectObject(m->dc, m->bitmap);
  /* Delete old dc and bitmap */
  DeleteDC(newmap->dc);
  DeleteObject(oldbm);
  LocalFree((HLOCAL) newmap);
  return 0;
  }

/* Resize VMAP attached to window -- returns 0 if OK */
int Vresize_winmap(HWND w, int wid, int hi)
  {
  MEMWINFO minfo;
  RECT r;
  GetClientRect(w, &r);
  Vget_info(w, &minfo);
  if (Vresize_map(minfo.map, wid, hi))
    return 1;
  minfo.width = wid;
  minfo.height = hi;
  save_info(w, &minfo);
  set_sb(w, &minfo, r.right - r.left,
         r.bottom - r.top, TRUE);
  return 0;
```

```
  }


/* Clear quit flag for user */
void Vdont_quit()
  {
  V_quit = 0;
  }


/* Get VWIN info -- public */
void Vget_info(HWND w, MEMWINFO * info)
  {
  int i;
  for (i = 0; i < sizeof(MEMWINFO); i += 2)
    *(unsigned short *) (((unsigned char *) info) + i)
       = GetWindowWord(w, i);
  }

/* Modeless dialog stuff */
HWND Vmodeless_dlg(HANDLE inst, LPSTR title,
     HWND parent, FARPROC fp)
  {
  int i;
  for (i = 0; i < MAXMODELESS; i++)
    if (!dlgtbl[i])
      break;
  if (i >= MAXMODELESS)
    return NULL;
```

```
  return (dlgtbl[i] = CreateDialog(inst, title, parent,
          fp));
  }

int Vend_dlg(HWND w)
  {
  int i;
  DestroyWindow(w);
  for (i = 0; i < MAXMODELESS; i++)
    if (dlgtbl[i] == w)
      break;
  if (i >= MAXMODELESS)     /* huh? */
    return 1;
  dlgtbl[i] = NULL;
  return 0;
  }

/* Set user loop -- must call in main() */
void Vuser_loop(int (*ul) ())
  {
  user_loop = ul;
  }

/* printing stuff */
static int print_abort;
static HWND print_dialog;

BOOL FAR PASCAL _export print_dlg(HWND dlg, WORD msg,
                        WPARAM wParam,
                        LONG lParam)
```

```
  {
  switch (msg)
      {
    case WM_INITDIALOG:
      /* turn off close */
      EnableMenuItem(GetSystemMenu(dlg, FALSE),
           SC_CLOSE, MF_GRAYED);
      return TRUE;

    case WM_COMMAND:
      /* Abort button! */
      print_abort = 1;
      EnableWindow(GetParent(dlg), TRUE);
      DestroyWindow(dlg);
      print_dialog = NULL;
      return TRUE;
      }
  return FALSE;
  }

BOOL FAR PASCAL _export abort_proc(HDC pcd, short code)
  {
  MSG msg;
  /* print abort proc */
  while (!print_abort &&
          PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    if (!print_dialog ||
        !IsDialogMessage(print_dialog, &msg))
        {
        TranslateMessage(&msg);
```

```
        DispatchMessage(&msg);
        }
  return !print_abort;
  }


static int print_map(VMAP * map, int mode)
  {
  int err = 0, blterr = 0;
#ifndef WIN32
  DWORD dims;
#else
  SIZE dims;
#endif
  FARPROC abort, printproc;
  HDC printer, temp;
  HBITMAP tempbit, orig;
  PRINTDLG prdlginfo;
  int wid, hi;
  RECT band;
  DOCINFO docinfo;
  int pgnr = 1;

  /* Create print DC */
  memset(&prdlginfo, 0, sizeof(PRINTDLG));
  prdlginfo.lStructSize = sizeof(PRINTDLG);
  prdlginfo.Flags = PD_RETURNDC | PD_USEDEVMODECOPIES |
     PD_NOSELECTION;
  if (mode)
    prdlginfo.Flags |= PD_NOPAGENUMS;
```

```
prdlginfo.nMinPage = prdlginfo.nFromPage = 1;
prdlginfo.nMaxPage = prdlginfo.nToPage = 9999;
/* Use standard print dialog */
if (!PrintDlg(&prdlginfo))
    {
    if (CommDlgExtendedError())
    MessageBox(NULL, "Can't Open Printer", NULL,
       MB_OK | MB_ICONSTOP);
    return 1;
    }
printer = prdlginfo.hDC;

/* Printer must support BitBlt */
if ((GetDeviceCaps(printer, RASTERCAPS) & RC_BITBLT) == 0)
    {
    MessageBox(NULL,
     "Printer doesn't support bit map images",
     NULL, MB_OK | MB_ICONSTOP);
    err = 1;
    goto dc_err;
    }

/* Create memory drawing surface */
tempbit = CreateCompatibleBitmap(printer,
        wid = GetDeviceCaps(printer, HORZRES),
        hi = GetDeviceCaps(printer, VERTRES));
if (tempbit)
    {
    temp = CreateCompatibleDC(printer);
    if (temp)
```

```
            orig = SelectObject(temp, tempbit);
        }
    if (!temp || !tempbit || !orig)
        {
        MessageBox(NULL, "Out of memory",
         NULL, MB_OK | MB_ICONSTOP);
        err = 1;
        goto dc_err;
        }
#ifndef WIN32
    dims = GetBitmapDimension(map->bitmap);
#else
    GetBitmapDimensionEx(map->bitmap, &dims);
#endif
    print_abort = 0;
    /* Set up printing abort dialog, etc. */
    printproc = MakeProcInstance(print_dlg, Hinstance);
    print_dialog = CreateDialog(Hinstance, "PrintDialogBox",
                  NULL, printproc);
    abort = MakeProcInstance(abort_proc, Hinstance);
    SetAbortProc(printer, abort);
    /* Start document */
    docinfo.cbSize = sizeof(DOCINFO);
    docinfo.lpszDocName = "VWIN Print Job";
    docinfo.lpszOutput = NULL;
    if (StartDoc(printer, &docinfo) > 0)
        {
        unsigned pgx, pgy, imx, imy;
        unsigned imlen, imwid, nrpgs = 1;
        unsigned int y = 0, yoff;
```

```
        pgx = GetDeviceCaps(printer, LOGPIXELSX);
        pgy = GetDeviceCaps(printer, LOGPIXELSY);
#ifndef WIN32
        imlen = HIWORD(dims);
        imwid = LOWORD(dims);
#else
        imlen = dims.cy;
        imwid = dims.cx;
#endif
        imx = GetDeviceCaps(map->dc, LOGPIXELSX);
        imy = GetDeviceCaps(map->dc, LOGPIXELSY);
        nrpgs = (imlen / imy * pgy) / hi;
        yoff = hi * (long) imy / pgy;
        if (nrpgs * (long) hi < imlen * (long) pgy / imy)
           nrpgs++;
        do
          {
          if (!mode &&
              (prdlginfo.Flags & PD_PAGENUMS) &&
               prdlginfo.nFromPage > pgnr)
           {
           /* Not at 1st selected page yet */
           pgnr++;
           y += yoff;
           continue;
           }
      /* Finished printing last selected page */
          if (!mode &&
             (prdlginfo.Flags & PD_PAGENUMS) &&
              prdlginfo.nToPage < pgnr)
```

```
        break;
/* Clear surface */
   PatBlt(temp, 0, 0, wid, hi, WHITENESS);
/* Scale VMAP bitmap to quasi-printer surface
 * (many printers don't support StrecthBlt) */
   if (mode)
      blterr = !StretchBlt(temp, 0, 0,
                  wid, hi, map->dc, 0, 0,
                  imwid, imlen, SRCCOPY);
   else
      blterr = !StretchBlt(temp, 0, 0,
                  min(wid, imwid * (long) pgx / imx),
                  min(hi, (imlen - y) * (long) pgy /
                      imy),
                  map->dc, 0, y, min(imwid,
                  (long) wid * imx / pgx),
                  min(yoff, imlen - y),
                  SRCCOPY);
   if (blterr)
      break;
/* Print surface to printer using Bands -- if you
 * don't use Bands for big bitmaps, Windows acts
 * erratically! */
/* don't use Escape(..NEWFRAME..) here! */
   StartPage(printer);
   Escape(printer, NEXTBAND, 0, NULL, (LPSTR) & band);
   while (!IsRectEmpty(&band))
     {
     (*abort) (printer, 0);
     if (!BitBlt(printer, band.left, band.top,
```

```
                    band.right - band.left,
                    band.bottom - band.top,
                    temp, band.left, band.top, SRCCOPY))
            {
            blterr = 2;
            break;
            }
          Escape(printer, NEXTBAND, 0, NULL, (LPSTR) &
                band);
          }
        y += yoff;      /* advance page */
        pgnr++;
        EndPage(printer);
        } while (!mode && !err && --nrpgs);

      DeleteObject(SelectObject(temp, orig));
      DeleteDC(temp);
      }
  else
    err = 1;

  if (!err)
    EndDoc(printer);
  if (!print_abort)
    DestroyWindow(print_dialog);
  if (err || blterr)
    MessageBox(NULL, "Can't Print", NULL,
              MB_OK | MB_ICONSTOP);
  else if (print_abort)
    MessageBox(NULL, "Printing Aborted", NULL,
```

```
                MB_OK | MB_ICONSTOP);
  FreeProcInstance(printproc);
  FreeProcInstance(abort);
dc_err:
  DeleteDC(printer);
  if (prdlginfo.hDevMode)
    GlobalFree(prdlginfo.hDevMode);
  if (prdlginfo.hDevNames)
    GlobalFree(prdlginfo.hDevNames);
  return err | print_abort | blterr;
  }

/* print a VMAP */
int Vprint_map(VMAP * map)
  {
  return print_map(map, 0);
  }

/* print a scaled VMAP */
int Vsprint_map(VMAP * map)
  {
  return print_map(map, 1);
  }


/* Save VWIN info (local use only) */
static void save_info(HWND w, MEMWINFO * info)
  {
  int i;
  for (i = 0; i < sizeof(MEMWINFO); i += 2)
```

```
    SetWindowWord(w, i, *(unsigned short *)
        (((unsigned char *) info) + i));
  }



/* Scroll handler (local) */
static void scrollit(HWND w, MEMWINFO * minfo, int type,
                WORD code, HWND sb, WORD pos)
  {

  unsigned int *offset;
  /* Store as long to avoid unsigned underflow */
  long newoffset;
  int step, page;
  RECT r;
  GetClientRect(w, &r);
  /* Set up offset and steps */
  if (type == SB_VERT)
      {
      offset = &minfo->yoff;
      step = minfo->map->ystep;
      page = minfo->map->ypage;
      }
  else
      {
      offset = &minfo->xoff;
      step = minfo->map->xstep;
      page = minfo->map->xpage;
      }
```

```
newoffset = *offset;

/* Process scroll command */
switch (code)
    {
  case SB_TOP:
    newoffset = 0;
    break;

  case SB_BOTTOM:
    if (type == SB_VERT)
      newoffset = minfo->height - (r.bottom - r.top);
    else
      newoffset = minfo->width - (r.right - r.left);
    break;

  case SB_LINEUP:
    step = -step;
    /* fall thru */
  case SB_LINEDOWN:
    newoffset += step;
    break;

  case SB_PAGEUP:
    page = -page;
    /* fall thru */
  case SB_PAGEDOWN:
    newoffset += page;
    break;
```

```
    case SB_THUMBPOSITION:
      newoffset = pos;
      break;
      /* I didn't process SB_THUMBTRACK since a big hires
       * VMAP can take too long to paint */
      }
  if (newoffset < 0)
    newoffset = 0;
  /* Update the offset */
  if (type == SB_VERT)
      {
      if (newoffset + (r.bottom - r.top) > minfo->height)
          newoffset = minfo->height - (r.bottom - r.top);
      }
  else
      {
      if (newoffset + (r.right - r.left) > minfo->width)
          newoffset = minfo->width - (r.right - r.left);
      }
  /* Update position */
  SetScrollPos(w, type, (unsigned) newoffset, TRUE);
  *offset = newoffset;
  save_info(w, minfo);
  Vcommit_draw(w);
  }

/* Set scroll parameters Currently you can't read them back
 * unless you call Vget_info(). If you must have a
 * Vget_scroll call you can write it! */
void Vset_scroll(VMAP * m, int xstep, int ystep,
```

```
                    int xpage, int ypage)
  {
  m->xstep = xstep;
  m->ystep = ystep;
  m->xpage = xpage;
  m->ypage = ypage;
  return;
  }

/* Process "scroll" keys */
static void key_scroll(HWND w, UINT key)
  {
  switch (key)
      {
    case VK_HOME:
      SendMessage(w, WM_VSCROLL, SB_TOP, 0L);
      break;

    case VK_END:
      SendMessage(w, WM_VSCROLL, SB_BOTTOM, 0L);
      break;

    case VK_PRIOR:
      SendMessage(w, WM_VSCROLL, SB_PAGEUP, 0L);
      break;

    case VK_NEXT:
      SendMessage(w, WM_VSCROLL, SB_PAGEDOWN, 0L);
      break;
```

```
    case VK_UP:
      SendMessage(w, WM_VSCROLL, SB_LINEUP, 0L);
      break;

    case VK_DOWN:
      SendMessage(w, WM_VSCROLL, SB_LINEDOWN, 0L);
      break;

    case VK_LEFT:
      SendMessage(w, WM_HSCROLL, SB_LINEUP, 0L);
      break;

    case VK_RIGHT:
      SendMessage(w, WM_HSCROLL, SB_LINEDOWN, 0L);
      break;
      }
  }


/* Local function to set scroll bars up */
static void set_sb(HWND w, MEMWINFO * minfo, UINT wid,
              UINT hi, int save)
  {
  RECT r;
  if (minfo->flags & V_INIT)
    return;
  if (minfo->flags & V_AUTOHSCROLL)
      {
      if (minfo->xoff && minfo->width <= wid)
        {
```

```
    /* If bitmap will fit in client area, make it do
     * so */
       minfo->xoff = 0;
       if (save)
         save_info(w, minfo);
       }
     /* Set up H bar */
     SetScrollPos(w, SB_HORZ, minfo->xoff, FALSE);
     SetScrollRange(w, SB_HORZ, 0,
          (unsigned)
           max(0L, (long) minfo->width - (long) wid)
          ,TRUE);
     /* Recompute size -- may have changed if scroll bar
      * enabled by above step */
     GetClientRect(w, &r);
     wid = r.right - r.left;
     hi = r.bottom - r.top;
     }
if (minfo->flags & V_AUTOVSCROLL)
     {
     if (minfo->yoff && minfo->height <= hi)
       {
    /* If bitmap will fit in client area, make it do
     * so */
       minfo->yoff = 0;
       if (save)
         save_info(w, minfo);
       }
     /* Set up V bar */
     SetScrollPos(w, SB_VERT, minfo->yoff, FALSE);
```

```
        SetScrollRange(w, SB_VERT, 0,
         (unsigned) max(0L, (long) minfo->height -
              (long) hi),
             TRUE);
        }
    }



/* Magic paint routine */
static void do_paint(HWND w)
    {
    HDC hdc;
    PAINTSTRUCT ps;
    MEMWINFO minfo;
    RECT r;
    int oldmode;
#ifndef WIN32
    DWORD oldworg, oldvorg;
#else
    POINT oldworg, oldvorg;
#endif
    hdc = BeginPaint(w, &ps);
    GetClientRect(w, &r);
    Vget_info(w, &minfo);
    if (minfo.map == NULL || minfo.map->dc == 0)
        {
        EndPaint(w, &ps);
        return;
```

```
        }
    /* Set up DC the way we like it */
    oldmode = SetMapMode(minfo.map->dc, MM_TEXT);
#ifndef WIN32
    oldworg = SetWindowOrg(minfo.map->dc, 0, 0);
    oldvorg = SetViewportOrg(minfo.map->dc, 0, 0);
#else
    SetWindowOrgEx(minfo.map->dc, 0, 0, &oldworg);
    SetViewportOrgEx(minfo.map->dc, 0, 0, &oldvorg);
#endif
    /* Do something different for scale window */
    if (minfo.flags & V_SCALE)
        {
        int oldmode;
        oldmode = SetStretchBltMode(hdc,
                    minfo.map->stretch_mode);
        StretchBlt(hdc, 0, 0, r.right - r.left,
                   r.bottom - r.top,
         minfo.map->dc, minfo.xoff, minfo.yoff,
         minfo.width, minfo.height, SRCCOPY);
        SetStretchBltMode(hdc, oldmode);
        }
    else
        {
/* if VMAP doesn't entirely cover window, clear first */
        if (r.right - r.left > minfo.width - minfo.xoff ||
            r.bottom - r.top > minfo.height - minfo.yoff)
          {
          HBRUSH brush;
          brush = CreateSolidBrush(GetBkColor(minfo.map->dc));
```

```
        brush = SelectObject(hdc, brush);
      /* erase "under" bitmap */
        PatBlt(hdc, 0, minfo.height,
           r.right - r.left, r.bottom - r.top, PATCOPY);
      /* erase to "right" of bitmap */
        PatBlt(hdc, minfo.width, 0,
           r.right - r.left, r.bottom - r.top, PATCOPY);
        DeleteObject(SelectObject(hdc, brush));
        }
       /* Draw it */
       BitBlt(hdc, 0, 0, minfo.width - minfo.xoff,
        minfo.height - minfo.yoff, minfo.map->dc,
        minfo.xoff, minfo.yoff, SRCCOPY);
       }
  SetMapMode(minfo.map->dc, oldmode);
#ifndef WIN32
  SetWindowOrg(minfo.map->dc, LOWORD(oldworg),
          HIWORD(oldworg));
  SetViewportOrg(minfo.map->dc, LOWORD(oldvorg),
       HIWORD(oldvorg));
#else
  SetWindowOrgEx(minfo.map->dc, oldworg.x, oldworg.y, NULL);
  SetViewportOrgEx(minfo.map->dc, oldvorg.x, oldvorg.y,
                   NULL);
#endif
  EndPaint(w, &ps);
  }
```

8

Things to Come

WHAT'S IN THIS CHAPTER

A look into the Windows crystal ball.

PREREQUISITES

None

Not so long ago, Windows was an oddity. There were only a few Windows applications, and most people only used Windows to run these applications. They would exit Windows to do most of their work. Today, things are quite different. Most PCs now come with Windows pre-installed. Many users never leave Windows, and there are numerous Windows applications available.

With the advent of Windows NT, Windows is no longer a thin shell over MSDOS. Windows NT is a complete operating system and can run on multiple platforms. Although it isn't very important now, in the future, many machines will take advantage of the ability of Windows NT to use multiple processors.

GUI systems, even Windows NT, are in their infancy. They will mature just as other operating software has matured over the years. Disk I/O is a good example of this.) Older operating systems for mainframes dealt with disk (or tape) data in a fixed block size (determined by the hardware). If the block size was 1K, for example, you were forced to read and write records in 1K chunks. If you needed 133 byte records, you had to write code to block and deblock these records into the 1K block. One day, someone realized that this blocking and deblocking process could be written once and put in the operating system. Very few programmers still write blocking and deblocking code.

As the amount of memory and processing power available to operating systems increases, techniques like VWIN (see Chapter 7) and TWIN (see Chapter 4) become more attractive. Eventually, most of the arcane code that

you write for GUI programs will reside in the GUI system—not in your program. A WM_PAINT handler will be as arcane as a disk deblocking routine is today.

Visual programming is another exciting frontier. Although products like Visual C++ purport to be visual, they are not as complete as other visual environments available on other platforms. However, you can expect more visual programming tools to appear soon.

Applications like Microsoft Word and Excel have started a trend that will certainly continue in the future. These programs provide powerful macro languages that are accessible via DDE (dynamic data exchange). You can also create compound documents with these programs using OLE (object linking and embedding). A compound document might contain a memo with an embedded spreadsheet and two graphic images. In the future, most major applications will support DDE and OLE. Future GUI operating systems may even support a universal macro language to control applications. As these trends continue, you'll find that your programs have to interact with these OLE and DDE applications more often. Although OLE and DDE are outside the scope of this book, you'll do well to become familiar with them. (See *Windows Programming for Mere Mortals* in the bibliography.)

GUI programming is here to stay. While today's GUI environments are somewhat difficult to work with, this won't always be the case. Commando techniques can help ease the transition, and they probably parallel the direction in which GUI systems are heading. Still, don't wait until tomorrow—start writing those programs today.

A

TWIN Calls

Note: many TWIN calls have aliases to standard library calls. For example, puts() maps to twin_puts().

```
void twin_create(TWIN_INFO *old,char *title,int wid,int hi);
```
Create TWIN window.

```
void twin_excreate(TWIN_INFO *old,char *title,DWORD style,
                   int x,int y,int wid,int hi,
                   HWND parent, HANDLE menu,int twid,int
thi);
```
Create TWIN window—arguments same as CreateWindow().

```
void twin_active(TWIN_INFO *old,TWIN_INFO *new);
```
Switch active window.

```
void twin_puts(char *s);
```
Write string to window.

```
void twin_putc(int c);
```
Write character to window.

```
void twin_goxy(int x,int y);
```
Set cursor position.

```
int twin_wherex(void);
```
Get cursor X position.

```
int twin_wherey(void);
```
Get cursor Y position.

```
void twin_show(void);
```
Update display (only useful if you draw with TF_HOLD set).

```
int twin_fflush(FILE *s);
```
Aliased to fflush(). If s is stdout, this maps to twin_show(). Otherwise, twin_fflush() calls fflush().

```
void twin__putc(int c);
```
Write character without showing it (used internally).

```
void twin_cls(void);
```
Clear window.

```
void twin_clreol(int x,int y);
```
Clear to end of line.

```
void twin_yield(void);
```
Yield time to Windows.

```
int twin_keyhit(void);
```
Check for keystrokes waiting.

```
int twin_getch(int *scan);
```
Get character and scan code.

```
int twin_getche(int *scan);
```
Get character and scan code and echo to current window.

```
int twin_setflag(int flagword);
```
Set TWIN flags.

```
int twin_gets(char *buf,unsigned int siz);
```
Get input string.

```
int twin_printf(char *fmt,...);
```
Printf-style output to window.

```
int twin_print(void);
```
Print current window to printer.

```
void twin_exit(int rv);
```
Exit TWIN program.

B

VWINL Call Reference

```
int Vcreate_window(char *title,DWORD style,int x,int
y,int width, int height,HWND parent,LPCSTR menu,long
(*callback)(),unsigned vflags,HDC *dc,HWND *win,int show)
```

The Vcreate_window function mostly mimics CreateWindow(). The menu parameter is actually a resource name or id. The vflags field is a VWINL flag. The window handle returns via the win pointer and the VMAP DC (if any) goes to the dc pointer (unless the dc pointer is NULL). The function returns zero upon success—any other value indicates failure.

```
VMAP *Vcreate_map(int width, int height)
```

Creates a VMAP of the specified width and height. This VMAP will match your current display, unless you have set the monochrome mode (see Vset_monomode()). The map will not display until you attach it to a window using Vselect_map().

```
VMAP *Vget_map(HWND w)
```

Returns a pointer to the VMAP associated with the window.

```
void Vdestroy_map(VMAP *map)
```

Releases a VMAP's resources. When a window closes, VWINL attempts to free its VMAP unless the V_NOFREEMAP flag is set.

```
VMAP *Vselect_map(HWND w, VMAP *new)
```

Changes the VMAP associated with a window. If the VMAP pointer is NULL, the window will have no VMAP. The function returns a pointer to the previously selected VMAP. By calling Vselect_map() repeatedly with different maps you can perform simple animations.

```
void Vcommit_draw(HWND w)
```

Force the contents of the window's VMAP to appear in the window. Until you call Vcommit_draw(), any output to the VMAP may or may not be visible. This call is actually a macro.

```
HDC Vget_mdc(VMAP *map)
```

Returns the DC associated with the specified VMAP. Actually a macro.

```
int Vget_stretchmode(VMAP *map)
```

Returns the stretch mode for the specified VMAP. For more about stretch modes, see the SetStretchBltMode() function in the Windows API reference. This function is actually a macro.

```
void Vget_info(HWND w,MEMWINFO *info)
```

Returns a read-only structure of information pertaining to the window.

```
unsigned long Vset_flags(HWND w,unsigned long flags,int
cmd)
```

You can use Vset_flags() to change a VWINL window's flags. You may need to call Vcommit_draw() after changing some flags. The cmd argument specifies how VWINL interprets the flags argument. If cmd is VF_STO, VWINL copies the flags to the window. VF_SET sets the specified flags leaving the other bits unchanged; VF_CLR clears them. The VF_TOG command causes the specified flags to change state. The return value is the previous flag value.

V_SCALE - Causes VWINL to scale the window's VMAP to fit the window's client area. If this flag is not set, VWINL clips the VMAP to the window. When clipping, VWINL can offset the VMAP (see Vset_offset()) or automatically manage scroll bars.

V_RESIZE - Causes the window's VMAP to automatically resize when the window resizes. This causes the VMAP's size to always match the window's size.

V_AUTOHSCROLL - When set, VWINL will automatically manage horizontal scroll bars for this window. When passed to Vcreate_window(), this flag forces the window to use the WS_HSCROLL style.

V_AUTOVSCROLL - When set, VWINL will automatically manage vertical scroll bars for this window. When passed to Vcreate_window(), this flag forces the window to use the WS_VSCROLL style.

V_NOMAP - Pass this flag to Vcreate_window to prevent VWINL from automatically creating a VMAP with the window. Presumably, you will use Vselect_map() to use a VMAP from another window or from Vcreate_map(). V_NOMAP is only meaningful during Vcreate_window().

V_NOQUIT - Ordinarily, closing a VWINL window will cause the entire application to terminate. If V_NOQUIT is set for a window, you may close it without disturbing your applications.

V_NOFREEMAP - This flag prevents VWINL from automatically freeing the window's VMAP when you close the window. You are responsible for calling Vdestroy_map() yourself. This is useful when more than one window shares a VMAP.

V_KSCROLL - Allow VWINL to intercept scrolling keys and translate them into scroll bar events. This is especially useful in conjunction with V_AUTOHSCROLL and V_AUTOVSCROLL.

V_ZEROSELECT - If this flag is set, a Vselect_map() call will also force the display offsets to zero. This causes the top left corner of the image to be visible. If you are animating with Vselect_map(), you don't want this flag set.

V_INIT - An internal flag used by VWINL. Don't set this flag at home.

Note: V_SCALE is incompatible with V_RESIZE, V_AUTOHSCROLL, or V_AUTOVSCROLL. The V_RESIZE flag is not compatible with V_AUTOHSCROLL, or V_AUTOVSCROLL.

```
void Vset_offset(HWND w,int x,int y)
```

Sets the offset of the specified window. When VWINL draws the VMAP to the window, it will use the offset as the VMAP's starting point (unless V_SCALE is set). The x and y parameters are in pixels.

```
void Vget_offset(HWND w,int *x,int *y)
```

This function returns the window's offset (see Vset_offset(), above).

```
HDC Vget_vdc(HWND w)
```

This function returns the VMAP DC associated with the given window. You use the DC to draw on the VMAP using Windows GDI calls.

```
int Vresize_winmap(HWND w,int width,int height)
```

Resize the VMAP associated with the specified window. This function automatically adjusts the window's scroll bars and handles other details.

```
int Vresize_map(VMAP *m,int wid,int hi)
```

Use Vresize_map() to change the size of a VMAP. If the VMAP is attached to a window, you will usually want to use Vresize_winmap() instead.

```
void Vset_scroll(VMAP *m,int xstep,int ystep,int
xpage,int ypage)
```

This function sets the scroll increments for a VMAP. By default the xstep and ystep variables equal 1 and the page variables equal 10. This causes smooth scrolling when you click the scroll bar arrows. When you scroll a page, 10 pixels go by.

```
void Vclear_map(VMAP *m)
```

Use Vclear_map() to erase the entire drawing surface of a VMAP using the background color.

```
void Vclear_win(HWND *w)
```

A macro that clears the VMAP associated with a window.

```
int Vset_stretchmode(VMAP *m,int mode)
```

Sets the VMAP's stretch mode (used when V_SCALE is set). For more about stretch modes, look up SetStretchBltMode() in the Windows API documentation. Returns the previous stretch mode.

```
void Vdont_quit(void)
```

During a WM_CLOSE message, you may call Vdont_quit() to prevent VWINL from terminating the application.

```
int Vset_monomode(int mode)
```

Sets or clears VWINL's monochrome mode. When monochrome mode is set, all Vcreate_window() and Vcreate_map() calls create monochrome bitmaps. These bitmaps may take up less space, but only support two colors.

```
HWND Vmodeless_dlg(HANDLE inst, LPSTR dlgname, HWND par-
ent, FARPROC fp)
```

This call works just like the standard CreateDialog() call except that it registers the modeless dialog with VWINL. Don't directly call CreateDialog(). Always destroy the dialogs you create with this call by using Vend_dlg().

```
int Vend_dlg(HWND w)
```

Use Vend_dlg() to terminate a modeless dialog created with Vmodeless_dlg().

```
Vuser_loop(int (*ul)())
```

You may install your own Windows event loop using this function. The event loop is exactly like an ordinary Windows event loop, and completely replaces VWINL's default loop. You must call this function in your main() routine, or not at all.

`Vprint_map(VMAP *)`

Invokes the standard print dialog to send the VMAP to the printer at its actual size. Vprint_map() prints multiple pages if required.

`Vsprint_map(VMAP *)`

Invokes the standard print dialog to send the VMAP to the printer scaled to fit on a page.

C

Annotated Bibliography

General Windows Programming

Microsoft Windows Guide to Programming, Redmond WA: Microsoft Press, 1992.

This book is a concise guide to programming Windows. Armed with what you learned in Chapter 2, this book may be all you need to start writing "real" Windows programs. Not surprisingly, this books seems to be Programming Windows with about half the pages removed (see below).

Charles Petzold, *Programming Windows,* Redmond WA: Microsoft Press, 1990.

The original (and best) Windows programming book. Sooner or later you will have to read it. Plan to spend quite some time digesting the 900+ pages. Still, it will be time well spent.

Commando Techniques

Woody Leonhard, *Windows 3.1 Programming for Mere Mortals*, Reading MA: Addison-Wesley, 1992.

If you are interested in non-C ways to write Windows programming, this irreverent book is a must. Even if, like me, you want to use C, Woody's explanations of OLE and DDE are excellent.

Andrew Schulman, et al., *Undocumented Windows*, Reading MA: Addison-Wesley, 1992.

Although this book covers the intimate details of Windows internal architecture, it contains some material relevant to commando programming. For example, it reveals simple ways to determine the amount of memory available to Windows. It also contains the WINIO library, which is similar to TWIN (see Chapter 4).

Al Williams, "A Quick Port With QuickWin," *Dr. Dobb's Journal*, August, 1993.

You can find an example of using Microsoft's QuickWin to port a large DOS-extended graphics program to Windows here.

Al Williams, "Simplified Windows User Interfaces," *Windows and DOS Developer's Journal*, July, 1993.

This article develops a program launcher using the same menu-only techniques that CDPLAYER illustrates (see Chapter 3).

Al Williams, "VWinL, A Virtual Window Library," *Dr. Dobb's Journal,* November, 1993 (forthcoming).

This version of VWINL doesn't do printing, but the technical details will be helpful if you want to modify the VWINL code in this book.

Al Williams, "A Quick BMP Viewer," *PC Techniques,* September/October 1993 (forthcoming).

Another QuickWin article, this one displays ordinary Windows BMP files by using the undocumented behavior of _putimage(). The viewer code sets the palette correctly—the only real tricky part of using QuickWin to manipulate bitmaps.

Index

D

E

F

G

H

I

P

Q

R

S

T

V

W

Windows Programming

COMMANDO WINDOWS™ PROGRAMMING

Fast and Easy Programming Solutions In C

Most Windows™ programming books talk about event loops, GDI, device contexts, and update routines–which is fine if you have plenty of time to learn the intricacies of Windows programming. But you live in the real world. You write real applications to access databases, fill in forms, and print reports. ***Commando Windows Programming*** shows you how to write these programs fast without having to learn every detail about Windows.

The book covers many techniques and tools to help you code for Windows and Windows NT, including:

- Writing dialog-only and menu-only programs
- Emulating text-based programs with edit controls
- Using libraries to simplify application creation
- Using Borland's OWL, Microsoft's Visual C++, and similar products.

Commando Windows Programming also includes two original libraries–TWIN and VWIN–that simplify many common programming tasks. TWIN simplifies the writing of text-based programs, and VWIN works for any type of program. These libraries also provide a starting point for developing your own tools.

Commando Windows Programming offers a quick return on your reading investment. Unlike other Windows books, you'll start writing practical programs right away.

Al Williams is author of *DOS and Windows Protected Mode* (Addison-Wesley, 1993) and *DOS 6 Developer's Guide*.

Addison-Wesley Publishing Company

52795

9 780201 624847

ISBN 0-201-62484-2

$27.95 US
$35.95 Canada