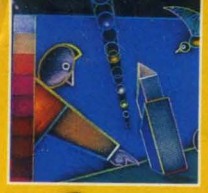


PANOV

SALOMON

PANOV

OS/2 Accredited

# The Art of OS/2® Warp Programming

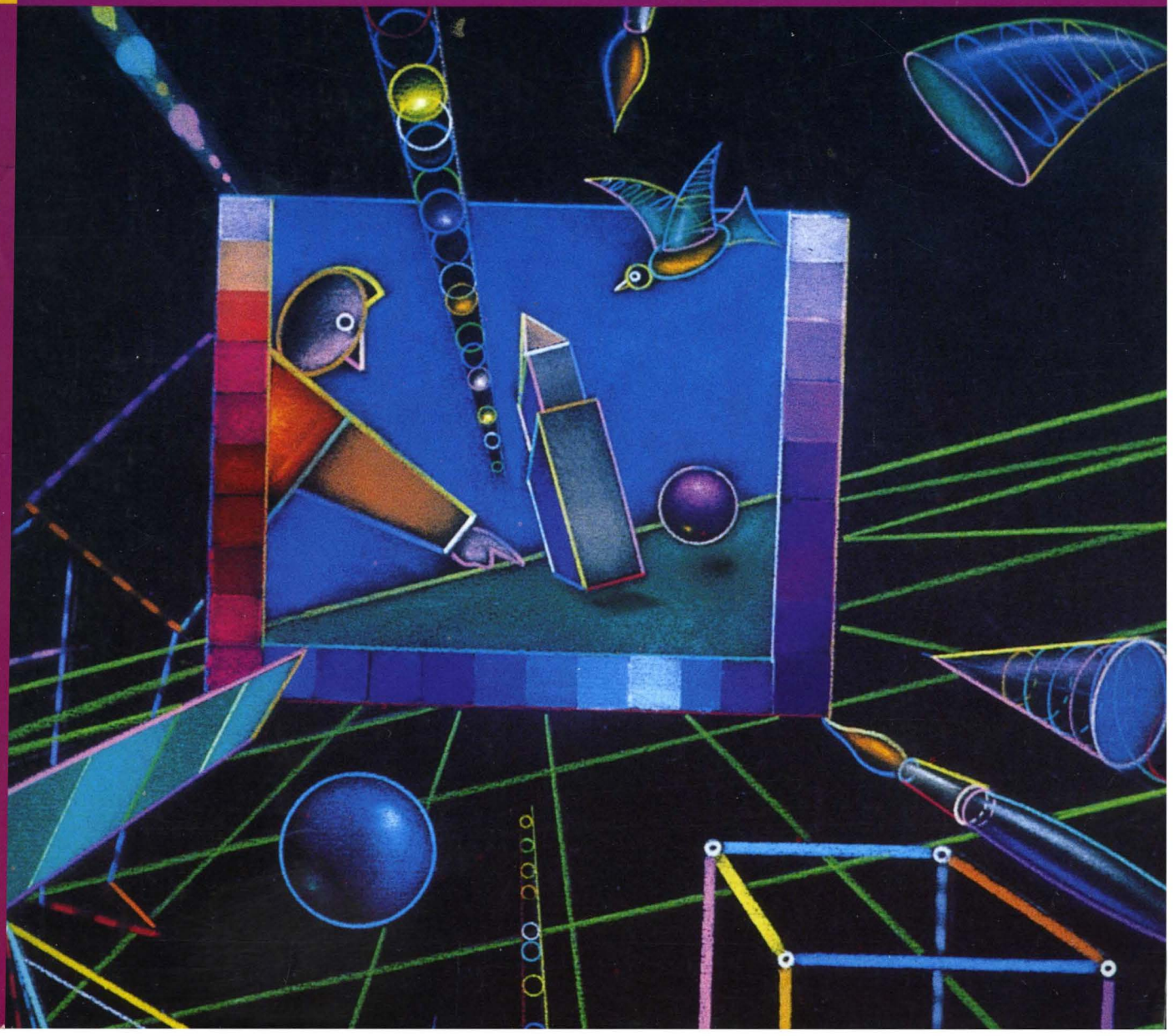
INCLUDES DISK

Kathleen Panov Larry Salomon, Jr. Arthur Panov

The Art of OS/2® Warp PROGRAMMING

INCLUDES DISK

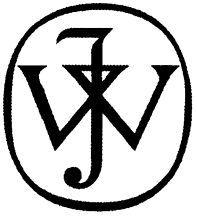
WILEY



*The Art of OS/2 Warp Programming*

# *The Art of OS/2 Warp Programming*

Kathleen Panov
Larry Salomon, Jr.
Arthur Panov



WILEY
John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

TRADEMARKS

OS/2, IBM, Presentation Manager, CUA, BookMaster, C Set/2, SCRIPT, THESEUS2, SPM/2, Common User Access are trademarks of IBM Corporation.
80286, 80386 are trademarks of Intel Corporation.
Macintosh, System/7 are trademarks of Apple Corporation.
Microsoft, Windows are trademarks of Microsoft Corporation.
SmartPics is a trademark of Lotus Development Corporation.
The clip-art images in this book were created using SmartPics from Lotus.

Publisher: Katherine Showalter
Editor: Theresa Hudson
Managing Editor: Maureen B. Drexel
Text Design & Composition: Kathleen Panov

This text is printed on acid-free paper.



**Library of Congress Cataloging-in-Publication Data**

Panov, Kathleen.
The art of OS/2 Warp programming / Kathleen Panov, Arthur Panov, Larry Salomon, Jr.
p. cm.
Includes index.
ISBN 0-471-08633-9
1. C (Computer program language) 2. OS/2 (Computer file)
I. Panov, Arthur. II. Salomon, Larry. III. Title.
QA76.73.C15P36 1993
005.4'469--dc20
93-25632
CIP

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

*This book is dedicated to Alexandra and Lisa.*

# Contents

# Figures

# Preface

## Notes From the Edge

OS/2 has come a long way since you last read the preface to this book. OS/2 2.1 made it to the public and it won accolades from the industry. OS/2 2.11 and OS/2 for Windows were subsequently released and were likewise praised by the industry pundits. Ironically, OS/2 was still the subject of criticism from the omnipresent cynics who sought to deride and belittle the operating system. However, when OS/2 Warp was released in the summer of 1994 and then won - for the third consecutive year - the "Product of the Year" award from Infoworld as well as many other awards, no one could deny it: the product that was "doomed to die" was here to stay after all.

It's been a long two years since The Art of OS/2 2.1 C Programming was released, but we've finally made it. The 1st edition, you said, was good. You liked the approach we took, analyzing the individual window classes instead of taking a task-oriented view of PM programming. You liked the "Gotchas" that indicated many of the things to watch out for when doing OS/2 development. However, there were also things you didn't like.

So, as OS/2 underwent its many mutations, so have we.

## What We Have Done

With this edition, you'll find all of the things that you said needed improving upon in the 1st edition. We've added 10 new chapters (50% more) to account for not only the essential areas we missed last time, but also the areas that "would have been nice to have." We've added more detail in the chapters that already existed as well as added more samples to them. We've added new sections to the existing chapters to allow the OS/2 developer to stay current with the new features of Warp.

## What We Expect of You

As with the last edition, we make some assumptions about your abilities. We assume that you have a good working knowledge of the C language. We do not assume that you have any prior development experience with a multitasking operating system, nor with a graphical user interface environment.

## What You Will Need

You will need the following software to compile the samples presented in the book:

OS/2 Warp
The Warp Programmer's Toolkit, or a compatible substitute
IBM C-Set++ (any version)

You may substitute any compiler for IBM C-Set++, but you should have a good knowledge of the compiler so that you can migrate the makefiles from IBM C-Set++. See Chapter 1 for a table of the more commonly used compiler switches and their meanings.

## Contacting the Authors

The authors look forward to your comments on this book, whether compliments, suggestions, or criticisms. Arthur and Kathleen Panov can be contacted by sending email to 71033,1721 (Compuserve) or 71033.1721@compuserve.com (Internet). Larry Salomon Jr. can be contacted by sending email to os2man@panix.com (Internet). All three authors follow the Internet newsgroup comp.os.os2.programmer.misc and Arthur and Kathleen also follow the OS/2 forums on Compuserve.

## Finally

We have worked hard to make sure that this book remains the book recommended by most people for doing OS/2 development. While we were not able to implement everything that you asked for in this edition of the book, we certainly tried. Enjoy.

# Acknowledgments

There are many people the authors would like to thank. Special thanks go to James Summers, Phil Doragh, Sam Detweiler, David Reich (author of *Designing OS/2 Applications* ), Tom Ingram, Bret and Brian Curran, Alan Warren, Jerry Cuomo, John Ponzo, Peter Haggar, Tanja Lindstrom, Marc Fiammante, and Mark Benge.

Lastly, we would like to thank Terri Hudson, Katherine Schowalter, and Maureen Drexel at John Wiley and Sons for making this book possible.

*The Art of OS/2 Warp Programming*

# Chapter 1

# Tools

All the examples in this book were compiled using the IBM C Set/2++ compiler and the IBM OS/2 Toolkit. There are other OS/2 compilers available including Watcom, Zortech, and the Borland C++ compiler. The include files and libraries necessary to access the system calls—memory management, multitasking, Presentation Manager, and so on—are found in the Developer's Toolkit. Although you can write a fully functional OS/2 program using only an OS/2 C compiler, you probably want to get the toolkit for any serious development work. Without it, you will need to delve into the minds of the OS/2 developers to find function prototypes, structure definitions, and the like. Suffice it to say, however, that doing so without the toolkit is an order of magnitude more difficult.

## Dialog Box Editor

The dialog box editor, DLGEDIT, is a very nice program to facilitate the creation of dialog boxes. The interface consists of a screen painter that lets you visually design the dialog boxes for your own applications. The editor will create a resource file (.RC), dialog file (.DLG), and a header file (.H). The dialog box editor is shipped with the Developer's Toolkit for OS/2 Warp.

## Resource Compiler

The resource compiler, RC, is a compiler that takes your application-defined resources—dialogs, menus, messages—and compiles them to a .RES file. This file can then be bound to your executable so that when the resources are needed, they are pulled into your program. The resource compiler is shipped with the Developer's Toolkit for OS/2 Warp and with the operating system.

## NMAKE

NMAKE is a newer version of the MAKE utility provided with most compilers. It is a program that sorts through all the tasks that need to be done to build an OS/2 executable and dispatches those tasks that should be done when a specific module has been changed. There are many different ways to build makefiles (.MAK). The IBM Workframe/2 environment will automate this process for you. However, the examples in this book contain .MAK files that were built by hand.

## IPFC

The program IPFC is the Information Presentation Facility Compiler. This will take a text-based file and create a .HLP or .INF file that can be used either with the help facility in Presentation Manager or using VIEW.EXE, which is shipped with OS/2 Warp. This program has been greatly expanded to give the programmer and the technical writer a lot of power over the online information displays.

# Libraries

The OS/2 Warp Developers Toolkit comes with two libraries, OS2286.LIB and OS2386.LIB. OS2386 contains the system call resolutions for all 32-bit entry points. OS2286 contains the 16-bit ones. You will need to explicitly link one of these in with your OS/2 Warp applications.

# Header (or INCLUDE) Files

The Developer's Toolkit for OS/2 contains many different header files, but only one, OS2.H, should be included in your program. However, you must use the #define INCL_xxx statements in order to include the function definitions, structures, data types, and the like necessary for your program. INCL_WIN will include all the necessary information for the Win... functions; INCL_DOS includes all the information for the Dos... functions; and INCL_GPI includes all the information for the Gpi... functions. These INCL_ statements can be broken down even further.

It is a very good idea for you to go snooping through the header files. They contain a lot of information, and also, in many cases the online and hard-copy documentation is just flat-out wrong. The header files are the final authority. One caveat here: The header files are not always complete. They will be adequate for development purposes 99 percent of the time; the other one percent of the time you will tear your hair out trying to find your mistake. Table 1.1 is a road map to the various header files.

Table1.1 Header Files

| Files | Description |
|---|---|
| OS2DEF.H | Includes the most common constants, data types, and structures. |
| PM*.H | Includes the necessary information for the Presentation Manager functions. |
| BSE*.H | Includes the necessary information for the base (Dos...) functions. |
| SOM*.H | Includes the System Object Model functions and information. |
| WP*.H | Includes all the information for the Workplace Object functions. |
| REXX*.H | Includes the REXX information and functions. |

# The Compiler Switches Used in This Book

All examples in this book include their own .MAK files. The compiler and linker switches for the IBM C Set/2++ compiler you may see are defined in Table 1.2 and Table 1.3. Check your compiler documentation for a full discussion of the compiler switches and the equivalents if you are not using the IBM C Set/2++ compiler.

Table 1.2 Compiler Switches

| Switch | Meaning | Default |
|---|---|---|
| C or C+ | Compile only, no linking | No |
| Gd- | Static linking | Yes |
| Ge+ | Build an .EXE file | Yes |
| Gm- | Single-threaded | Yes |
| Gm+ | Multithreaded | No |
| Kb+ | Basic diagnostic messages (check for function prototypes) | No |
| Ms- | Use system linkage | No |
| O- | No optimization | Yes |
| Re | Subsystem development enabled | Yes |
| S2 | SAA Level 2 | No |
| Sa | ANSI C | No |
| Spn | Structure packing along n byte boundaries | 4 byte boundaries |
| Ss+ | Allow use of // comments | No |
| W3 | Warning level | Yes |

Table 1.3 Linker Switches

| Linkage opts | Meaning |
|---|---|
| /MAP | Generate MAP file |
| /A:n | Align along n byte boundaries |
| /PM:VIO | Window-compatible application |

# Chapter 2

# Memory Management

In OS/2 1.3, the memory management scheme was designed to support the Intel segmented architecture. The 80286 could provide access to memory in segments that were limited in size to 64K. At times more than 64K was necessary. In those cases, the developer would have to create elaborate memory management schemes. This changed in OS/2 2.0. The amount of memory that developers can access is only limited by three items:

- The physical amount of RAM in the system
- The amount of disk space available on the drive pointed to by the SWAPPATH variable in config.sys
- The absolute limit of 512 MB

By dropping support for the 80286 and supporting only processors capable of supporting a 32-bit engine, OS/2 could have the flat, paged memory architecture of other non Intel-based chips. Both the Motorola 680x0 chips (base of the Apple Macintosh and other machines) and the RISC-base chips (base for IBM's RS-6000) use the flat, paged architecture. You can probably see where this is leading. Designing a memory model that is portable is the first step in designing a portable operating system. A 32-bit operating system will allow addresses of up to 0xFFFFFFFF, or 4GB. This also gives programmers the opportunity to allocate memory objects that are as large as the system memory allows.

OS/2 1.x used the 16-bit addressing scheme of the 80286. A location in memory was represented as a 16:16 pointer, in selector-offset fashion. The upper portion of the selector maps into a descriptor table. The entry in the descriptor table maps the absolute location of the memory address.

Thirty-two-bit OS/2 has only three segments that combine to make 4GB total. This means that memory addresses are represented as a 0:32 pointer. All memory resides in these three segments. A normal program will run in the segment that starts at address 0 and covers 480Mb. Protected dynamic link libraries (DLLs) see the same 480Mb region plus 32Mb above it. This 512Mb addressability limitation is due to compatibility with 16-bit OS/2 programs. The kernel functions see the full 4GB region. This is where the big performance boost comes in. Because all memory is in these three segments, when the operating system has to switch memory objects, the segment registers do not always have to be loaded. A flat memory management scheme has one more advantage: All pointers are near pointers, since all memory can be addressed using a 0:32 pointer. This means no more "FAR" jumps for the operating system. This also means memory models —small, medium, large, and huge—are now obsolete.

The basis of the 32-bit OS/2 memory management functions is *DosAllocMem*. This function allocates memory in 4,096-byte chunks called pages; however, a developer can allocate several contiguous pages in one call. While this means that you can allocate any amount of memory up to the process limit, it also

means that you can waste a considerable amount of memory if you're not careful. Consider the following code fragment:

```
for (i=0; i < 1000; i++)
        DosAllocMem(    &p[i],
                        1,
                        PAG_READ | PAG_WRITE | PAG_COMMIT );
```

The first parameter is a PPVOID, the second parameter is the number of bytes allocated, and the last parameter is the memory flags. We'll see this again soon.

What you see in the code fragment is 1,000 1-byte blocks being allocated. What you don't see is the 1,000 4,095-byte blocks that are not being used because *DosAllocMem* allocates memory as an integral number of pages.

## Committing Memory

OS/2 2.0 also introduced the concept of committing memory. A call to *DosAllocMem* will reserve an address range for the memory; however, physical memory is actually assigned to the range only if the PAG_COMMIT flag is specified. (A side note here: In 32-bit OS/2, a page is only assigned to an address *really* when the page is touched.) If you try to access uncommitted memory, otherwise known as sparse memory objects, TRAP-BOOM! If you choose to allocate memory without committing it, you have two ways of having it committed later—*DosSetMem* or *DosSubSetMem.* Also, in 32-bit OS/2, memory is guaranteed to be initialized to 0. This prevents the application from having to initialize the memory, thereby touching all the memory, thereby committing all the memory.

The following is a very simple program to allocate memory and to show a little about what happens to bad programs. Remember that we are seasoned professionals. Do not attempt this at home. Well, you may want to attempt it at home, but if you attempt this at work consistently, it may get you fired.

**BADMEM.C**

```
#define  INCL_DOSMEMMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
INT main(VOID)

{
   PBYTE            pbBuffer;
   APIRET           arReturn;
   USHORT           usIndex;

   arReturn = DosAllocMem((PPVOID)&pbBuffer,
                          3000,
                          PAG_READ|PAG_WRITE|PAG_COMMIT);

   if (arReturn == 0)
   {

      for (usIndex = 0; usIndex < 4097; usIndex++)
      {

         printf("\nNow Writing to %p ( index = %d ) ",
                &pbBuffer[usIndex],
                usIndex);
```

```
        pbBuffer[usIndex] = 1;

      }                                   /* endfor                */
   }                                      /* endif                 */
   return 0;
}
```

**BADMEM.MAK**

```
BADMEM.EXE:                    BADMEM.OBJ
        LINK386 @<<
BADMEM
BADMEM
BADMEM
OS2386
BADMEM
<<

BADMEM.OBJ:                    BADMEM.C
        ICC -C+ -Kb+ -Ss+ BADMEM.C
```

**BADMEM.DEF**

```
NAME BADMEM WINDOWCOMPAT
DESCRIPTION 'Memory example
             Copyright (c) 1992-1995 by Kathleen Panov.
              All rights reserved.'
```

Now, you may look at this code and say, "But, you're allocating only 3,000 bytes, and you're writing to 4,098." Okay, this is bad code; however, it illustrates that no matter how much you specify as bytes allocated, the operating system will return it to you in 4,096-byte pages, and you could use them all and never see a protection violation. You'd just end up stomping all over some data that you may need. However, notice that when you try to write to byte 4097, TRAP! This too can happen to you, so be very careful about writing to unallocated, uncommitted memory.

The flags used as the page attributes in the preceding example were PAG_READ | PAG_WRITE | PAG_COMMIT. Table 2.1 lists the possible page attributes.

Table 2.1 Page Attributes

| Flag | Description |
|---|---|
| PAG_READ | Read access is the only access allowed. A write to the memory location will generate a trap. |
| PAG_WRITE | Read, write, and execute access is allowed. |
| PAG_EXECUTE | Execute and read access to the memory is allowed. This flag will also provide compatibility for future versions of the operating system. |
| PAG_GUARD | Sets a guard page after the allocated memory object. If any attempt is made to write to that guard page, a guard page fault exception is raised, and the application is given a chance to allocate more memory as needed. (See Chapter 6—Exception Handling) |

| | |
|---|---|
| OBJ_TILE | All memory objects are put into the tiled, or compatibility, region in OS/2 2.x. All objects are aligned on 64K boundaries. Provides upward compatibility when applications will be allowed by future versions of the operating system to access regions above the 512 MB "16-bit compatibility" barrier. |

Often the example programs and manuals will reference the default page attribute, *fALLOC*; this is a #define for OBJ_TILE | PAG_COMMIT | PAG_EXECUTE | PAG_READ | PAG_WRITE.

## Suballocating Memory

*DosSubSetMem* and *DosSubAllocMem* provide a more efficient way for developers to access chunks of memory smaller than 4,096 bytes. An application can use *DosAllocMem* to allocate some number of bytes, called a memory object. *DosSubSetMem* is used to initialize or grow a heap within the memory object. This function has three parameters, PVOID *offset*, ULONG *flags*, and ULONG *size*. The flags parameter is used to provide more details about the heap. The following options are available for this parameter:

- DOSSUB_INIT – You *must* specify this option when first suballocating a memory object. If this bit is not set, the operating system will try to find shared memory from another process. If no shared memory is found, the return code ERROR_INVALID_PARAMETER (87) will result.

- DOSSUB_GROW – This option will grow the memory pool to the size specified by the last parameter. Note that this flag will increase just the amount of memory in the memory pool that will be suballocated. It will not increase the size of the memory pool itself.

- DOSSUB_SPARSE_OBJ – This option allows the operating system to commit and decommit pages as they are needed. Note that all pages in the memory object must be uncommitted.

- DOSSUB_SERIALIZE – Serializes the suballocation of shared memory by multiple processes. If you have two processes sharing memory and suballocating it, use this to make your life easier.

*DosSubSetMem* has access to all memory in the memory object. The application then calls *DosSubAllocMem* to allocate a smaller chunk of the heap. *DosSubAllocMem* can allocate all but 64 bytes of the heap. The 64 bytes is called a memory pool header. The operating system uses it to manage the suballocated portion. *DosSubAllocMem* has three parameters, PVOID *Offset*, PPVOID *SmallBlock*, and ULONG *Size*. The amount actually allocated is a multiple of 8 bytes, rounded *up* if not a multiple of 8.

The following program shows you how to handle suballocation of memory:

**SUBMEM.C**

```
#define  INCL_DOSMEMMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
INT main(VOID)

{
   PBYTE             pbHeap;
   PBYTE             pbPtr1;
   PBYTE             pbPtr2;
   PBYTE             pbPtr3;
   PBYTE             pbPtr4;
   APIRET            arReturn;
```

```
arReturn = DosAllocMem((PPVOID)&pbHeap,
                       4096,
                       PAG_READ|PAG_WRITE);

printf("\nDosAllocMem ( ) returns %d",
       arReturn);

arReturn = DosSubSetMem((PVOID)pbHeap,
                        DOSSUB_SPARSE_OBJ|DOSSUB_INIT,
                        4096);

printf("\nDosSubSetMem ( ) returns %d",
       arReturn);

arReturn = DosSubAllocMem(pbHeap,
                          (PPVOID)&pbPtr1,
                          20);

printf("\nDosSubAlloc ( ) returns %ld "
   "pbPtr1 size requested = 20",
       arReturn);

arReturn = DosSubAllocMem(pbHeap,
                          (PVOID)&pbPtr2,
                          15);

printf("\nDosSubAlloc ( ) returns %ld "
   "pbPtr2 size requested = 15",
       arReturn);

arReturn = DosSubAllocMem(pbHeap,
                          (PVOID)&pbPtr3,
                          45);

printf("\nDosSubAlloc ( ) returns %ld "
   "pbPtr3 size requested = 45",
       arReturn);

arReturn = DosSubAllocMem(pbHeap,
                          (PVOID)&pbPtr4,
                          8);

printf("\nDosSubAlloc ( ) returns %ld "
   "pbPtr4 size requested = 8",
       arReturn);

printf("\n\nHeader size = %d",
       pbPtr1-pbHeap);
printf("\nSize of pbPtr1 ptr = %d",
       pbPtr2-pbPtr1);
printf("\nSize of pbPtr2 ptr = %d",
       pbPtr3-pbPtr2);
printf("\nSize of pbPtr3 ptr = %d",
       pbPtr4-pbPtr3);
printf("\nSize of pbPtr4 undeterminable");

DosSubFreeMem(pbHeap,
              pbPtr1,
              20);
DosSubFreeMem(pbHeap,
              pbPtr2,
              15);
DosSubFreeMem(pbHeap,
              pbPtr3,
              45);
DosSubFreeMem(pbHeap,
              pbPtr4,
              8);
```

```
   arReturn = DosFreeMem(pbHeap);

   printf("\nDosFreeMem ( ) returns %d",
          arReturn);

   return 0;
}
```

**SUBMEM.MAK**

```
SUBMEM.EXE:                     SUBMEM.OBJ
        LINK386 @<<
SUBMEM
SUBMEM
SUBMEM
OS2386
SUBMEM
<<

SUBMEM.OBJ:                     SUBMEM.C
        ICC -C+ -Kb+ -Ss+ SUBMEM.C
```

**SUBMEM.DEF**

```
NAME SUBMEM WINDOWCOMPAT
DESCRIPTION 'Memory suballocation example
             Copyright (c) 1992-1995 by Kathleen Panov.
             All rights reserved.'
```

You'll notice when you run this program that all your pointer sizes are rounded up in increments of 8 and that *DosSubAllocMem* starts allocating at the 65th byte of the memory object.

# Shared Memory

Shared memory is the fastest method of interprocess communication. There are two types of shared memory, named and unnamed. Shared memory is created by a call to *DosAllocSharedMem*. If creating shared memory, the second parameter to *DosAllocSharedMem* is the name for the memory, in the form of \SHAREMEM\MemName. If using unnamed memory, a NULL is specified. There is one other difference between shared and unnamed memory—the process that allocates an unnamed memory object must declare it as giveable by using *DosGiveSharedMem*, and the process accessing the memory object must call *DosGetSharedMem*. Shared memory can be committed and decommitted just like private memory. Also, when suballocating memory from a shared memory pool, both *DosSubSetMem* must use the same size parameter in both processes, or an error will result.

**Gotcha!**

All the processes involved with the shared memory (both the getting and giving) must free the shared memory before it is available for reuse. If only one process frees the memory, you may begin to notice an increase in your program's memory consumption over time. The system maintains a usage count of shared memory that enables it to keep track of all the processes that have access to the shared memory. The IBM products THESEUS2 and SPM/2 are the only tools available to detect memory leakage. They are two excellent tools to monitor the system performance.

The following programs are examples of allocating a named shared memory object. Notice that the memory is being allocated in a downward fashion; private memory is allocated upward from the bottom of the available space.

**BATMAN.C**

```
#define  INCL_DOSMEMMGR
#include <os2.h>
#include <stdio.h>
#include "dynduo.h"
INT main(VOID)

{
   PBYTE            pchShare;
   APIRET           arReturn;

   arReturn = DosGetNamedSharedMem((PPVOID)&pchShare,
                                   SHAREMEM_NAME,
                                   PAG_READ|PAG_WRITE);

   if (arReturn == 0)
   {
      printf("\nString read is: \"%s\"\n",
             pchShare);
   }                                   /* endif                */
   DosFreeMem(pchShare);
   return 0;
}
```

**BATMAN.DEF**

```
NAME BATMAN WINDOWCOMPAT

DESCRIPTION 'Shared memory example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

**ROBIN.C**

```
#define  INCL_DOSMEMMGR
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include "dynduo.h"
INT main(VOID)

{
   PCHAR            pchShare;
   APIRET           arReturn;

   arReturn = DosAllocSharedMem((PVOID)&pchShare,
                                SHAREMEM_NAME,
                                1024,
                                PAG_READ|PAG_WRITE|PAG_COMMIT);

   if (arReturn == 0)
   {
      strcpy(pchShare,
             "Holy Toledo, Batman");
      getchar();
      DosFreeMem(pchShare);
```

```
        getchar();
        DosFreeMem(pchShare);
    }                                   /* endif                */
    return 0;
}
```

**ROBIN.DEF**

```
NAME ROBIN WINDOWCOMPAT

DESCRIPTION 'Shared memory example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

**DYNDUO.H**

```
#define SHAREMEM_NAME              "\\SHAREMEM\\BATMAN"
```

**DYNDUO.MAK**

```
ALL:                                BATMAN.EXE \
                                    ROBIN.EXE

BATMAN.EXE:                         BATMAN.OBJ
        LINK386 @<<
BATMAN
BATMAN
BATMAN
OS2386
BATMAN
<<

BATMAN.OBJ:                         BATMAN.C \
                                    DYNDUO.H
        ICC -C+ -Kb+ -Ss+ BATMAN.C

ROBIN.EXE:                          ROBIN.OBJ
        LINK386 @<<
ROBIN
ROBIN
ROBIN
OS2386
ROBIN
<<

ROBIN.OBJ:                          ROBIN.C \
                                    DYNDUO.H
        ICC -C+ -Kb+ -Ss+ ROBIN.C
```

## *DosAllocMem* or *malloc*?

*DosAllocMem*, *DosSubSetMem*, and *DosSubAllocMem* might seem like a bit of overkill if you would like to have only 20 bytes for a string every now and then. And they are. These functions are most useful for large programs that allocate large quantities of memory at one time, allocate shared memory, or have special memory needs. For most smaller applications, *malloc* from an ANSI C compiler will be just fine. Also, you probably will find that *malloc* is much more portable to other versions of OS/2 running on top of the Power PC. The C Set++ version of *malloc* is the only compiler version of *malloc* that will be compared to *DosAllocMem* and company. In most cases *malloc* will provide memory to the program just as fast as *DosAllocMem*. The C Set++ compiler uses a special algorithm, designed to provide the expected amount of memory in the fastest time. The following program uses *malloc* to allocate memory and then displays the

displays the amount of memory allocated plus the location of the pointer in memory. You probably will start to notice a pattern emerging, and there is one.

**SPEED.C**

```
#define  INCL_DOSMEMMGR
#define  INCL_DOSMISC
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
INT main(VOID)

{
   PBYTE            apbBuf[1500];
   USHORT           usIndex;

   for (usIndex = 0; usIndex < 1500; usIndex++)
   {
      apbBuf[usIndex] = malloc(usIndex);

      if (usIndex > 0)
      {
         printf("\napbBuf [%d] = %p delta = %ld",
                usIndex,
                apbBuf[usIndex],
                (PBYTE)apbBuf[usIndex]-(PBYTE)apbBuf[usIndex-1]);
      }                                  /* endif                */
      if (((usIndex%25) == 0) && (usIndex != 0))
      {
         printf("\nPress ENTER to continue...");
         fflush(stdout);
         getchar();
      }                                  /* endif                */
   }                                     /* endfor               */
   for (usIndex = 0; usIndex < 1500; usIndex++)
   {
      free(apbBuf[usIndex]);
   }                                     /* endfor               */
   return 0;
}
```

**SPEED.MAK**

```
SPEED.EXE:                          SPEED.OBJ
        LINK386 @<<
SPEED
SPEED
SPEED
OS2386
SPEED
<<

SPEED.OBJ:                          SPEED.C
        ICC -C+ -Kb+ -Ss+ SPEED.C
```

**SPEED.DEF**

```
NAME SPEED WINDOWCOMPAT

DESCRIPTION 'malloc() example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

By looking at the program's otput, you'll notice that memory allocation starts by using 32 for values between 1 and 16. It uses 64 for values between 17 and 32, 128, 256, and finally 512. You may notice a few "bumps" in the algorithm. They occur when the C runtime is using some of the memory for its own purposes.

# Chapter 3

# Multitasking

The session and task management facilities in OS/2 give the programmer an exceptional opportunity to fully exploit the multitasking features in the operating system. Threads or processes can provide applications with a tremendous performance boost. OS/2 provides a special brand of multitasking, preemptive multitasking, which is different from the multitasking found in either Windows or the Macintosh System 7. Preemptive multitasking is controlled by the operating system. Each process is interrupted when its time to run is over, and the process will never realize it has been interrupted the next time it is running. In other words, OS/2 lets your computer walk and chew gum at the same time. With either the Mac or Windows, your computer takes a step, chews the gum, takes a step, chews the gum, and so on.

The task management of OS/2 is divided into three separate entities:

- Threads
- Processes
- Sessions

A thread is the only unit to get its own time slice of the CPU. All threads belonging to a process are contained within that process, and each thread has its own stack and registers. There is a systemwide limit of 4,096 threads; however, CONFIG.SYS contains a THREADS parameter that is usually set at a significantly smaller number—256 is the default. The base operating system uses approximately 40 threads, so most applications are limited to 216 threads unless the THREADS parameter is changed. Typically, a thread should have one distinct function; for example, file I/O, asynch communications, or heavy number crunching. Each thread has a thread identifier—a TID. Each thread also has a priority. The higher the priority, the more CPU time slices are given to the thread. A thread is much quicker to create than a process or session and has less system overhead. All threads within a process run in the same virtual address space; therefore, global resources, such as file handles, and global variables are accessible from all threads in the process. Threads are created using *DosCreateThread*, with the first thread created automatically by the operating system. When a thread is created it is assigned the same priority class as the thread that created it.

A process is a collection of threads and resources that are owned by those threads. Each process occupies a unique address space in memory that cannot be accessed by other processes in the system. Two processes can access the same area in memory only by using shared memory. A process also contains file handles, semaphores, and other resources. All processes contain at least one thread, the main thread. A process also contains a unique identifier—a PID. A process contains its own set of memory pages that can be swapped in and out as the kernel switches from one process to the other. A process can create other processes;

however, these must be of the same session type. For instance, a full-screen process can only create other full-screen processes. The five types of processes are OS/2 Full Screen, OS/2 windowed, DOS Full Screen, DOS windowed, and Presentation Manager.

A session is similar to a process except a session also contains ownership of the mouse, keyboard, and video. A session can contain either one process or multiple processes. The task list (accessed by Ctrl-Esc) contains a list of all running sessions. When a process or session creates a new session using *DosStartSession*, the keyboard, screen, and mouse are responsive only to the session in the foreground. The session chosen as the background can gain control of the three resources only by switching to the foreground.

## The Scheduler

The OS/2 Scheduler runs on a round-robin type of disbursement of CPU time. The Scheduler deals only with threads, not processes or sessions. Threads have four different priority levels: time-critical, server class or fixed high, regular, and idle time. The first threads to run are the time-critical threads. All time-critical threads will run until there are no more time-critical threads waiting to be run. After all time-critical threads are finished, the server-class threads are run. After server-class, the regular class of threads are run. After the regular class of threads are run, idle-time threads are run. Within each class of priorities are 32 sublevels. A thread that is not running is called a "blocked" thread.

The OS/2 Scheduler does a lot of monkeying around with thread priorities. Threads are given "boosts" by the scheduler to make OS/2's multitasking smarter. Three types of artificial priority boosts are given to threads :

- Foreground boost
- I/O boost
- Starvation boost

The foreground boost is given to the user interface thread of the process that is in the foreground. This is usually the main thread. The foreground process is the process with which the user is currently interacting. This makes the system respond quickly when the user clicks a mouse button or types in characters at a keyboard. This boost is a full boost in priority. Also, a Presentation Manager thread has a boost applied to it while it is processing a message.

We'll take this opportunity to get up on our soapbox. Do not throw away all the work the operating system does to provide the end user with a crisp response time. Any operation that takes any amount of time should be in its own thread. A well-written, multithreaded program running on a 20 MHz 386SX will be blazingly fast to an end user used to a single-threaded program running on a 486 DX2. Well, maybe that's a little bit of an exaggeration, but you get the idea. Any time you see an hourglass on the screen for more than a second or two, and the user cannot size a window or select a menu item, that program should be put through a serious design review. Okay, off the soapbox, and on to our regularly scheduled programming.

An I/O boost is given after an I/O operation is completed. An I/O boost does not change a thread's priority but will bump it up to level 31 (the highest level) within its own priority class.

A starvation boost is given to a thread in the regular class that has not been able to run. The MAXWAIT parameter in CONFIG.SYS is used to define how long, in seconds, a thread must *not* run before it is given a starvation boost. The default value is 3 seconds.

The time slices for threads that are given a starvation boost or an I/O boost are different from a normal time slice. Because of the tinkering the scheduler does with their priorities, they do not get to run as long as a nonadjusted thread would run. The length of time for the "short" and normal time slices is controlled by the TIMESLICE parameter in CONFIG.SYS. The first value represents the "short" time slice length; the default amount of time is set to 32 ms. The second value represents the normal time slice length; the default amount of time is set to 65536 ms.

A programmer can refine the way the threads in a program are run in four ways:

- *DosSetPriority*
- *DosSuspendThread/DosResumeThread*
- *DosEnterCritSec / DosExitCritSec*
- *DosSleep*

```
APIRET APIENTRY  DosSetPriority(ULONG scope, ULONG ulClass,
                                LONG delta, ULONG PorTid)
```

*DosSetPriority* has four parameters. The first indicates to what extent the priority is to be changed. The priority can be changed at the process or thread level. The *ulClass* parameter indicates at what class to set the priority. The *delta* parameter indicates at what level within the class to set the priority. The last parameter is the process ID of the process to be affected. A value of 0 indicates the current process. Note that a process can change just the priority of a child process. *DosSetPriority* can be called anytime in a thread's lifetime. It is used to adjust the class and/or the priority level within that class. *DosSetPriority* should be used to adjust threads whose tasks need special timing considerations. For instance, a thread handling communications would probably want to run at a server class. A thread that backs up files in the background should be set at idle-time priority, so that it would run when no other tasks were running. You can change the priority of threads in another process, but only if they were not changed explicitly from the regular class.

```
APIRET DosResumeThread( TID tid )
APIRET DosSuspendThread( TID tid )
```

The only parameter to each of these functions is the thread ID of the thread. *DosResumeThread* and *DosSuspendThread* are used to change a thread's locked state. *DosSuspendThread* will cause a thread to be set to a blocked state. *DosResumeThread* is used to cause a suspended thread to be put back in the list of ready-to-run threads.

*DosEnterCritSec* is used to suspend all other threads in a process. This function should be used when it is vitally important that the running thread not be interrupted until it is good and ready. *DosExitCritSec* will cause all the suspended threads to be put back in a ready-to-run state. A program can nest critical sections within critical sections. A counter is incremented by *DosEnterCritSec* calls and decremented by *DosExitCritSec* calls. Only when this counter is 0 will the critical section exit. You probably should avoid nesting critical sections unless you absolutely need this functionality. One final note on critical sections: If a thread exits while in a critical section, the critical section automatically ends.

**Gotcha!**

*DosEnterCritSec* can be a very dangerous function. If for any reason the single thread running is put in a blocked state and needs some other thread to cause it to be unblocked, your program will go out to lunch and will not return. For example, *DosWait...Sem* are major no-nos in a critical section, because the required *DosPost...Sem* calls probably will exist in a thread that will be put in a suspended state. Also, be very careful calling a function that resides in a .DLL when inside a critical section. The function may use semaphores to manage resources, and it may be put in a suspended state while waiting for those resources to be freed.

*DosSleep* is the most practical function of the group. Using this function you can put a thread in a suspended state until a specified amount of time has passed. *DosSleep* has only one argument, the amount of time to "sleep." This value is specified in milliseconds. A thread cannot suspend other threads using *DosSleep*, only itself. When *DosSleep* is called with an argument of 0, the thread gives up the rest of its time slice. This does not change the thread's priorities or affect its position in the list of ready-to-run threads.

## The Subtleties of Creating a Thread

*DosCreateThread* is used to create a thread. The following code illustrates this:

```
DosCreateThread( &tidThread, /* thread TID */
                 pfnThreadFunction, /* pointer to fn */
                 ulThreadParameter, /* parameter passed */
                 ulThreadState, /* 0 to run, 1 to suspend */
                 ulStackSize ); /* 4096 at a minimum */
```

The first parameter contains the address of the threads TID, or Thread ID. The next parameter is a pointer to the function that the operating system will call when the thread is running. When using *DosCreateThread*, a typical function prototype of a thread function looks something like this:

```
VOID APIENTRY fnThread( ULONG ulThreadArgs );
```

Notice the APIENTRY keyword. This is used to indicate that this is a function that will be called by the operating system. The *ulThreadParameter* is 4 bytes of data, in the form of a ULONG, that are passed as an argument to the thread function. If you need to pass more than one value, you need to create a structure that contains all the values you want to pass. The first bytes of the structure should contain the size of the structure that is being passed. Also, if you use a structure, make sure you pass the address of the structure as the data. The *ulThreadState* parameter indicates whether the thread is started in a running state (with a value of 0) or in a suspended state (with a value of 1). If the thread is started suspended, somebody needs to call *DosResumeThread* to get the thread going. The last parameter is the stack size. The thread's stack is located in memory when the thread is blocked and is loaded into registers when the thread becomes ready to run. In OS/2 2.0, the programmer no longer needs to mess with allocating and freeing the memory for the stack. However, the programmer does need to know the maximum amount of memory that the stack will use. This is the value passed as the last parameter. This memory is not committed until it is absolutely necessary. The thread stack uses guard pages to commit a new page as necessary. Also, you may notice that a thread stack grows *downward* rather than upward as normal memory grows.

## Threads and the C Runtime

The C runtime library can cause problems when used within a thread other than the main thread. Because the C runtime uses many internal variables, multiple threads using the C runtime can cause problems unless

the runtime library is notified of the other threads. C-Set/++ has provided a separate function, *_beginthread,* to fix this situation. This function should be used to create threads in which you want to use the C library. The parameters for *_beginthread* are very similar to the parameters for *DosCreateThread.*

```
_beginthread(  pfnThreadFunction,
        /* void pointer to thread function */
               pNull,
        /* This is a NOP parameter,  used for migration */
               ulStackSize,
        /* stack size */
               pArgList );
        /* void pointer to argument list */
```

The prototype for a thread function changes a little here. The typical thread function prototype looks something like this:

```
        void fnThread(  void *pArgList );
```

**Gotcha!**

When using the C Set/++ compiler, make sure you specify the multithreaded option, Gm+. Also, either let the compiler link in the proper library for you, or make sure you specify DDE4M*.LIB.

## A Thread Example

The following example creates threads with different priorities. Each thread writes its priority to the screen. In this example, we avoided using *_beginthread* and *printf* but instead used *DosCreateThread* and *DosWrite*. This gives us the opportunity to start the threads in a suspended state.

**THREAD.C**

```
#define  INCL_DOS
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#define  THREAD_SUSPEND 1L
#define  STDOUT         (HFILE) 1
VOID APIENTRY MyThread(ULONG ulThreadArgs);

INT main(VOID)
{
   APIRET           arReturn;
   TID              tidThreadID[5];
   USHORT           usIndex;
   ULONG            ulThreadPriorities[] =
   {
      PRTYC_FOREGROUNDSERVER,PRTYC_TIMECRITICAL,PRTYC_REGULAR,
      PRTYC_NOCHANGE,PRTYC_IDLETIME
   }
   ;

   for (usIndex = 0; usIndex < 5; usIndex++)
   {
      arReturn = DosCreateThread(&tidThreadID[usIndex],
                                 MyThread,
                                 ulThreadPriorities[usIndex],
                                 THREAD_SUSPEND,
                                 4096);
```

```
      arReturn = DosSetPriority(PRTYS_THREAD,
                                ulThreadPriorities[usIndex],
                                (LONG)0,
                                tidThreadID[usIndex]);

      if (arReturn)
      {
         printf("\narReturn = %d",
                arReturn);
      }                                  /* endif                */
      DosResumeThread(tidThreadID[usIndex]);
   }                                     /* endfor               */
   DosSleep(2000);
   return 0;
}

VOID APIENTRY MyThread(ULONG ulThreadArgs)
{
   USHORT           usIndex;
   CHAR             cChar;
   ULONG            ulBytesWritten;

   for (usIndex = 0; usIndex < 200; usIndex++)
   {
      cChar = (CHAR)ulThreadArgs+'0';

      DosWrite(STDOUT,
               (PVOID)&cChar,
               1,
               &ulBytesWritten);
   }                                     /* endfor               */
   return ;
}
```

**THREAD.MAK**

```
THREAD.EXE:                     THREAD.OBJ
        LINK386 @<<
THREAD
THREAD
THREAD
OS2386
THREAD
<<

THREAD.OBJ:                     THREAD.C
        ICC -C+ -Gm+ -Kb+ -Ss+ THREAD.C
```

**THREAD.DEF**

```
NAME THREAD WINDOWCOMPAT

DESCRIPTION 'Multithread example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

The first part of the program is the actual creation of the threads. We'll create five almost identical threads. Each thread is started in suspended state by specifying 1 (THREAD_SUSPEND) as *ulThreadFlags*. The

thread function, *MyThread*, is assigned to *pfnThreadFunction*. Since the thread function itself is fairly small, the minimum stack size of 4,096 is specified.

The one difference between the five threads is their priority. Each thread priority is passed to *MyThread* in the *ulThreadArgs* variable. An array, *ulThreadPriorities[]*, holds all the possible thread priority classes.

*DosSetPriority* is used actually to change the priority of the threads from regular priority to the respective priority in the *ulThreadPriorities[]* array. The first parameter, PRTYS_THREAD, specifies that only one thread, not all the threads in the process, will have its priority affected. The second parameter is the priority class to use. The third parameter is the delta of the priority level. Within each class are 32 levels that can be used to refine a thread's priority even further. Threads at level 31 of a class will execute before threads at level 0 of the same class. This parameter specifies the change to make to the current level, *not the absolute level value itself.* Values are from –31 to +31. A value of 0 indicates no change, and this is what we use in this example. The last parameter, *tidThreadID[]*, is the thread ID of the thread whose priority is to be changed.

Once the thread is created and its priority has been changed, *DosResumeThread* is called to wake the thread up and have it begin running.

These steps are repeated for all five threads in a FOR loop. *DosSleep* is used to delay the main thread from ending for 2 seconds. This gives all the threads a chance to complete.

## The Thread Output

Each thread will print out its priority 200 times. Although this example is an elementary program, it will give you some insight into how threads are scheduled. The screen output you see should show the "3" thread (PRTYC_TIMECRITICAL) running first, followed by the "4" thread (PRTYC_FOREGROUNDSERVER). The "2" thread (PRTYC_REGULAR) and the "0" thread (PRTYC_NOCHANGE) actually are running at the same priority and should appear somewhat intermingled. A 0 in the priority class means no change from the existing class. The "1" thread (PRTYC_IDLETIME) should always run after the other priority threads.

## Executing a Program

The function *DosExecPgm* is used to execute a child process from within a parent process. A child process is a very special kind of process. Normally all resources are private to each process; however, because of the parent/child relationship, a child can inherit some of the resources owned by the parent. Most handles can be inherited; however, memory cannot, unless it is shared memory. This protects one process (even if it is a child process) from destroying another process.

The following examples uses *DosExecPgm* to create a new command process session. The command process executes a "dir *.*".

**PROG.C**

```
#define  INCL_DOSPROCESS
#include <os2.h>
#include <stdio.h>
#define  BUFFER_SIZE   200
INT main(VOID)

{
```

```
   APIRET           arReturn;
   CHAR             achFail[BUFFER_SIZE];
   RESULTCODES      rcResult;

   arReturn = DosExecPgm(achFail,
                         BUFFER_SIZE,
                         EXEC_ASYNC,
                         "CMD.EXE\0 /C dir *.* \0",
                         (PSZ)NULL,
                         &rcResult,
                         "CMD.EXE");

   if (arReturn)
   {
      printf("\narReturn = %d",
             arReturn);
   }                                  /* endif                */
   printf("\nrcResult = %ld",
          rcResult.codeResult);
   return 0;
}
```

**PROG.MAK**

```
PROG.EXE:                          PROG.OBJ
        LINK386 @<<
PROG
PROG
PROG
OS2386
PROG
<<

PROG.OBJ:                          PROG.C
        ICC -C+ -Kb+ -Ss+ PROG.C
```

**PROG.DEF**

```
NAME PROG WINDOWCOMPAT

DESCRIPTION 'DosExecPgm example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

The first parameter of *DosExecPgm* is a buffer that is used to store information if the application being started fails. The size of the buffer is the next parameter.

The third parameter indicates how you want to the child process to run. A child process can run simultaneously with the parent process (EXEC_ASYNC), or the parent can wait to run until the child has finished (EXEC_SYNC). There are other options, but these are the two most commonly used.

**Gotcha!**

The parameter string conforms to regular C parameter conventions, where argv[0] is the name of the executing program. After the program name, *you must insert one null character.* Following the null is the regular string of program arguments. These arguments must be terminated by *two null characters.* This is accomplished easily by manually inserting one null at the end of the argument string and letting the normal C string null termination insert the other.

The argument string for this example is:

```
"CMD.EXE\0 /C dir *.*\0";
```

CMD.EXE will execute a new command processor session. The "\0" is the first null character. The argument string "/C dir *.* \0" indicates the session will be closed when it finishes executing the dir *.* command. The "\0" at the end is the first of the last two nulls. The second null is inserted automatically at the end of the string.

The fifth parameter is the environment string to pass to the new program. This is formatted:

```
variable = text \0 variable = text \0\0
```

Each environment variable you want to set must be ended with a null character. The end of the string must be terminated with *two null characters.* A null value in the environment string variable indicates that the child process will inherit its parent's environment.

The next parameter is a RESULTCODES structure. This structure contains two values, a termination code and a result code. The operating system provides a termination code to indicate whether the program ended normally or whether some error, for example, a trap, ended the program abruptly. The result code is what is returned by the program itself, either through *DosExitProcess* or through *return.*

The last parameter is the actual name of the program to be executed. A fully qualified pathname is necessary only if the executable file is not found in the current directory or in any of the directories specified in the path.

There are several ways to tell whether a child process has terminated, but the easiest by far is *DosCwait.* This function either will wait indefinitely until a child process has ended, or will return immediately with an error, ERROR_CHILD_NOT_COMPLETE.

## Sessions

A session is a process with its own input/output devices (i.e., Presentation Manager/non-Presentation Manager output, keyboard, and mouse). There are several different types of sessions:

- OS/2 window
- OS/2 full screen
- DOS window
- DOS full screen
- Presentation Manager (PM)

All are started the same way, using *DosStartSession.*

**Gotcha!**

There is a little bit of a trick to determine whether to use *DosExecPgm* or *DosStartSession.* The difference lies in whether the newly created process is going to perform *any* input or output. Table 3.1 outlines the guidelines. If you need to determine the type of an application (or .DLL), *DosQueryAppType* can be used.

Table 3.1 Starting Session Guidelines

| Parent Type | Child Type | Child does I/O ? | Use |
|---|---|---|---|
| PM | PM | — | *DosExecPgm* or *DosStartSession* |
| Non-PM | PM | — | *DosStartSession* |
| PM | Non-PM | yes | *DosStartSession* |
| PM | Non-PM | no | *DosExecPgm* or *DosStartSession* |

The following example program starts a seamless Windows session using *DosStartSession.*

**STARTWIN.C**

```
#define  INCL_DOS
#define  INCL_WINPROGRAMLIST
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
INT main(VOID)

{

   STARTDATA        sd;
   PID              procID;
   ULONG            sessID;
   APIRET           rc;

/* initialize everything to 0                                    */

   procID = 0;
   sessID = 0;
   memset(&sd,
          0,
          sizeof(STARTDATA));

/* for Warp or OS/2 for Windows, start real Windows              */

   sd.PgmName = "C:\\WIN\\WIN.COM";

/* insert path to some Windows program here                      */

   sd.PgmInputs = "C:\\WIN\\WEP\\GOLF.EXE";
   sd.SessionType = PROG_31_STDSEAMLESSCOMMON;
   sd.Length = sizeof(STARTDATA);

/* in this case start as independent                             */

   sd.Related = SSF_RELATED_INDEPENDENT;
   sd.FgBg = SSF_FGBG_FORE;
```

```
   sd.TraceOpt = SSF_TRACEOPT_NONE;
   sd.TermQ = 0;
   sd.Environment = NULL;
   sd.PgmControl = SSF_CONTROL_VISIBLE|SSF_CONTROL_SETPOS;
   sd.InitXPos = 50;
   sd.InitYPos = 50;
   sd.InitXSize = 400;
   sd.InitYSize = 600;
   sd.Reserved = 0;
   sd.ObjectBuffer = NULL;
   sd.ObjectBuffLen = 0;

   rc = DosStartSession(&sd,
                        &sessID,
                        &procID);
   printf("\nReturn code rc from DosStartSession = %d",
          rc);
   fflush(stdout);
   return (0);

}
```

**STARTWIN.MAK**

```
STARTWIN.EXE:                          STARTWIN.OBJ
        LINK386 @<<
STARTWIN
STARTWIN
STARTWIN
OS2386
STARTWIN
<<

STARTWIN.OBJ:                          STARTWIN.C
        ICC -C+ -Kb+ -Ss+ STARTWIN.C
```

**STARTWIN.DEF**

```
NAME STARTWIN WINDOWCOMPAT

DESCRIPTION 'Simple example to start a seamless Windows session
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

The *DosStartSession* function itself is actually very small. Most of the preparatory work is done by setting up the STARTDATA structure. The structure looks like this:

```
typedef struct _STARTDATA     /* stdata */
        {
        USHORT  Length;
        USHORT  Related;
        USHORT  FgBg;
        USHORT  TraceOpt;
        PSZ     PgmTitle;
        PSZ     PgmName;
        PBYTE   PgmInputs;
        PBYTE   TermQ;
        PBYTE   Environment;
        USHORT  InheritOpt;
        USHORT  SessionType;
        PSZ     IconFile;
        ULONG   PgmHandle;
        USHORT  PgmControl;
        USHORT  InitXPos;
        USHORT  InitYPos;
        USHORT  InitXSize;
        USHORT  InitYSize;
        USHORT  Reserved;
        PSZ     ObjectBuffer;
        ULONG   ObjectBuffLen;
        } STARTDATA;
typedef STARTDATA *PSTARTDATA;
```

*Length* is the length of the structure in bytes.

*Related* specifies whether the new session will be a child session (field is TRUE) or an independent session (field is FALSE ).

*FgBg* defines whether the session is to be started in the foreground (field is FALSE) or in the background (field is TRUE ).

*TraceOpt* specifies whether there is to be any debugging (tracing) of the new session. TRUE indicates debug on; FALSE indicates debug off.

*PgmTitle* is the name that the program is to be called. This is *not* the name of the executable, only the title for any windows or task list. If a NULL is used, the executable name is used for the title.

*PgmName* is the fully qualified pathname of the program to load.

*PgmInputs* is a pointer to a string of program arguments (see page 23 for argument formatting.)

*TermQ* is a pointer to a string that specifies the name of a system queue that will be notified when the session terminates.

*Environment* is a pointer to a string of environment variables (see page 23 for environment variable formatting.)

*InheritOpt* indicates whether the new session will inherit open file handles and an environment from the calling process. TRUE in this field will cause the session to inherit the parent's environment; FALSE will cause the session to inherit the shell's environment.

*SessionType* specifies the type of session to start. Possible values are listed in Table 3.2.

Table 3.2 Description of Session Types

| Value | Description |
|---|---|
| SSF_TYPE_DEFAULT | Uses the program's type as the session type |
| SSF_TYPE_FULLSCREEN | OS/2 full screen |
| SSF_TYPE_WINDOWABLEVIO | OS/2 window |
| SSF_TYPE_PM | Presentation Manager program |
| SSF_TYPE_VDM | DOS full screen |
| SSF_TYPE_WINDOWEDVDM | DOS window |

In addition, Table 3.3 lists the values that are also valid for Windows programs.

Table 3.3 Valid Windows Session Types

| Value | Description |
|---|---|
| PROG_31_STDSEAMLESSVDM | Windows 3.1 program that will execute in its own windowed session. |
| PROG_31_STDSEAMLESSCOMMON | Windows 3.1 program that will execute in a common windowed session. |
| PROG_31_ENHSEAMLESSVDM | Windows 3.1 program that will execute in enhanced compatibility mode in its own windowed session. |
| PROG_31_ENHSEAMLESSCOMMON | Windows 3.1 program that will execute in enhanced compatibility mode in a common windowed session. |
| PROG_31_ENH | Windows 3.1 program that will execute in enhanced compatibility mode in a full screen session. |
| PROG_31_STD | Windows 3.1 program that will execute in a full screen session. |

*IconFile* is a pointer to a fully qualified pathname of an .ICO file to associate with the new session.

*PgmName* is a program handle that is returned from either *WinAddProgram* or *WinQueryProgramHandle*. A 0 can be used if these functions are not used.

*PgmControl* specifies the initial attributes for either the OS/2 window or DOS window sessions. The following values can be used:

SSF_CONTROL_VISIBLE
SSF_CONTROL_INVISIBLE
SSF_CONTROL_MAXIMIZE
SSF_CONTROL_MINIMIZE
SSF_CONTROL_NOAUTOCLOSE
SSF_CONTROL_SETPOS

Except for SSF_CONTROL_NOAUTOCLOSE and SSF_CONTROL_SETPOS, the values are pretty self-explanatory. SSF_CONTROL_NOAUTOCLOSE is used only for the OS/2 windowed sessions and will keep the sessions open after the program has completed. The SSF_CONTROL_SETPOS value indicates that the operating system will use the InitXPos, InitYPos, InitXSize, and InitYSize for the size and placement of the windowed sessions.

The second parameter to *DosStartSession* is the address of a ULONG that will contain the session ID after the function has completed. The last parameter is the address of a PID (process ID) that will contain the new process's PID after the session has started.

# Chapter 4

# File I/O and Extended Attributes

File I/O is one of the most important aspects of any operating system. OS/2 makes the file system programming very easy to understand and master, yet it still provides the programmer with many flexible and powerful features. OS/2 has introduced to DOS developers the new concept of Installable File Systems, which allows various file systems to be installed like device drivers. OS/2 introduces the new High Performance File System (HPFS), which allows greater throughput and security features for servers, workstations, and local area network (LAN) administrators. The File Allocation Table (FAT) compatibility is preserved, so DOS users can manipulate their files without any constraints.

## Extended Attributes

The following examples demonstrate some straightforward file manipulation, yet provide the user with some useful concepts. It is also necessary to introduce the concept of Extended Attributes (EAs), which is the lesser-known OS/2 file system feature. One of the examples shows a way to gain access to the various types of EAs. EAs appeared in OS/2 1.2 and have remained there through the 16- to 32-bit migration; they are nothing more than additional data that is associated with the file. The user does not see this extra data. It is there only for the use of the application and operating system. The designers had to be creative in order to implement EA support under FAT due to the fact that DOS, which is the grandfather of FAT, never had support for EAs. The HPFS does not require the same creativity in implementation, thus the FAT implementation is the one that deserves a short explanation. The FAT directory entries take up 32 bytes (20 hex) and are represented by Table 4.1.

Table 4.1 FAT Directory Entries

| Entry | Location |
|---|---|
| Filename: | 00–07 |
| Extension: | 08–0A |
| Attribute: | 0B |
| Reserved: | 0C–15 |
| Time: | 16–17 |
| Date: | 18–19 |
| FAT cluster : | 1A–1B |
| Size: | 1C–1F |

Most DOS files will have the reserved bits 0C to 15 set to zero. This is the area that is utilized to attach the Extended Attributes to the files in OS/2. The EA allocation clusters use the 14h and 15h bytes, and thus may appear illegal to some DOS applications. In order to avoid DOS compatibility problems, another file entry is maintained called EA DATA. SF; this file "pretends" to own all of the loose EA clusters on the hard disk, thus eliminating "lost clusters" messages from chkdsk.exe and similar messages from other disk

managing utilities. Two references to all EA clusters exist: one that is maintained with the 14h- and 15h-byte directory entries, and one that is "assigned" to the EA DATA. SF. This implementation creates a source of confusion for users who are not familiar with EAs. For example, when using EA unaware backup utilities or when copying files from an OS/2 partition under DOS, most users do not know what to do with the EA DATA. SF file. Users must realize that the EA clusters referenced by that file belong to several different applications. In order to maintain the EAs properly, it is best to use the OS/2 EAUTIL program to separate EAs from their owners, then copy them as separate files and later reunite them for a happy ending. Generally the EAs take up a substantial amount of disk space; if space is at a premium, EAS not associated with a critical attribute can usually be deleted. In such cases, the presence of the EA is not critical to the application's correct execution and thus it can be removed. Users must take care in determining which EAs can be removed, as some applications will not work correctly afterward.

A more thorough discussion of EA APIs and a detailed discussion of the API structures for the FAT and HPFS can be found in the *OS/2 Programming Guide* and various other IBM technical publications. The short description offered here is merely for the benefit of the programming examples and to help the programmer understand the API syntax used to attain the EA information. Extended attributes will appear foreign to DOS users and programmers, and their usefulness generally is questioned almost immediately. Only upon closer inspection does it become evident that EAs are quite important and really constitute a must-have feature, especially in high-end operating systems such as OS/2. Basically the Extended Attributes are nothing more than a storage area of information no more than 64K in size that are available for applications to use as they please. OS/2 defines several standard types of EAs that are available for general use. Also, the programmer can define application-specific extended attributes. The only restriction is that the total EA size cannot exceed 64K. Standard EAs are called SEAs, and by convention start with a period [.]. They include:

- .ASSOCTABLE
- .CODEPAGE
- .COMMENTS
- .HISTORY
- .ICON
- .KEYPHRASES
- .LONGNAME
- .SUBJECT
- .TYPE
- .VERSION

It is a good idea to not use the preceding [.] character in your own applications. The operating system reserves the right to use [.] as the first character of the EA name types. Nothing prevents users from implementing the same convention, but if OS/2 designers decide to add another standard type that happens to use your EA name, some unpredictable behavior may result. The type of data that is stored within an SEA is representative of the SEA name. For example, the .ICON SEA will contain the icon data, while the .TYPE SEA will contain the file object's type. This type can represent an executable, data, metafile, C code, bitmap, icon, resource file, object code, DOS binary, and so on. As you might have guessed, the .TYPE SEA is one of the more frequently used attributes of a file object. Note that extended attributes are associated not only with files but also with subdirectories. In fact, the subdirectory containing the Workplace Shell desktop information contains subdirectories that have many, many EAs.

# EAs—Fragile: Handle with Care

A programmer must take certain steps while using EAs. First, if the file objects are being moved or copied to a system that does not support EAs (such as a DOS-FAT combination), the programmer must take care not to lose the EAs that may be associated with the particular file object. Consider the case of uploading a file with EAs to a UNIX machine and then downloading the same file back. Doing so may result in EAs being lost or misplaced because most UNIX machines do not support EAs. Another good example is trying to copy a file that has a long name from an HPFS partition to a FAT partition. Since FAT supports the 8.3 naming convention only, the file name may be truncated, but that is not a problem since the correct HPFS name may be stored in the .LONGNAME EA. An application that manipulates files must be EA- and HPFS-aware in order to perform proper file management in an OS/2 environment.

# The LIBPATH.C Example

The first example we discuss attempts to find out the value of the LIBPATH environment variable. In OS/2 Warp, an extended LIBPATH variable was created. This special variable can be set or queried from the command line or from an API, *DosSetExtLIBPATH* and *DosQueryExtLIBPATH*. This variable can be changed dynamically and either prepended or appended to the system LIBPATH variable. The system LIBPATH itself cannot be returned from the regular environment SET command or a DosQuery... API. Occasionally the system LIBPATH variable is a handy thing to know. So, a not-so-clean solution is to find the value of the boot drive, find the CONFIG.SYS file, and attempt to extract the LIBPATH string from that file. This will work only when there have been no previous changes to the CONFIG.SYS file since the system has been booted and specifically no direct manipulations of the system LIBPATH value. Although this example is a crude kluge, the method actually can be useful on a number of occasions.

**LIBPATH.C**

```
#define  INCL_DOSFILEMGR
#define  INCL_DOSMISC
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define  CONFIG_FILE    "?:\\CONFIG.SYS"
#define  LIBPATH        "LIBPATH"
#define  CR             13
#define  LF             10

INT main(VOID)
{
   APIRET           arReturn;
   ULONG            ulDrive;
   PCHAR            pchFile;
   HFILE            hfFile;
   ULONG            ulAction;
   ULONG            ulBytesRead;
   PCHAR            pchBuffer;
   PCHAR            pchLibpath;
   USHORT           usIndex;
   FILESTATUS3      fsStatus;

   pchBuffer = NULL;

   /*********************************************************/
   /* find the boot drive                                   */
   /*********************************************************/

   arReturn = DosQuerySysInfo(QSV_BOOT_DRIVE,
                              QSV_BOOT_DRIVE,
```

```
                              &ulDrive,
                              sizeof(ulDrive));

   pchFile = CONFIG_FILE;
   pchFile[0] = ulDrive+'A'-1;


   /*********************************************************/
   /* get the size of the CONFIG.SYS                        */
   /*********************************************************/

   arReturn = DosQueryPathInfo(pchFile,
                               FIL_STANDARD,
                               &fsStatus,
                               sizeof(fsStatus));

   /*********************************************************/
   /* allocate buffer size + 1 for NULL                     */
   /*********************************************************/

   pchBuffer = malloc(fsStatus.cbFile+1);

   arReturn = DosOpen(pchFile,
                      &hfFile,
                      &ulAction,
                      0,
                      FILE_NORMAL,
                      FILE_OPEN,
                      OPEN_FLAGS_FAIL_ON_ERROR|
                         OPEN_FLAGS_SEQUENTIAL|
                         OPEN_SHARE_DENYNONE|OPEN_ACCESS_READONLY
                         ,
                      NULL);

   arReturn = DosRead(hfFile,
                      pchBuffer,
                      fsStatus.cbFile,
                      &ulBytesRead);

   arReturn = DosClose(hfFile);

   pchBuffer[fsStatus.cbFile] = 0;

   /*********************************************************/
   /* seach buffer for LIBPATH variable                     */
   /*********************************************************/

   pchLibpath = strstr(pchBuffer,
                       LIBPATH);

   if (pchLibpath == NULL)
   {

      /*******************************************************/
      /* will only execute this section of code if LIBPATH is  */
      /* NOT all caps                                        */
      /*******************************************************/

      for (usIndex = 0; usIndex < strlen(pchBuffer); usIndex++)
      {
         if (toupper(pchBuffer[usIndex]) == 'L')
            if (toupper(pchBuffer[usIndex+1]) == 'I' && toupper
               (pchBuffer[usIndex+2]) == 'B' && toupper(pchBuffer
               [usIndex+3]) == 'P' && toupper(pchBuffer[usIndex+4
               ]) == 'A' && toupper(pchBuffer[usIndex+5]) == 'T'
               && toupper(pchBuffer[usIndex+6]) == 'H')
            {
               pchLibpath = pchBuffer+usIndex;
               break;
```

```
            }
        }
    }

    /*************************************************************/
    /* read to the line feed                                     */
    /*************************************************************/


    for (usIndex = 0; usIndex < CCHMAXPATHCOMP; usIndex++)
    {
        if (pchLibpath[usIndex] == CR)
        {
            if (pchLibpath[usIndex+1] == LF)
                break;
        }                                   /* endif             */
    }                                       /* endfor            */
    pchLibpath[usIndex] = 0;

    /*************************************************************/
    /* print out the LIBPATH                                     */
    /*************************************************************/

    printf("\n%s\n",
           pchLibpath);
    free(pchBuffer);
    return  arReturn;
}
```

**LIBPATH.MAK**

```
LIBPATH.EXE:                    LIBPATH.OBJ
        LINK386 @<<
LIBPATH
LIBPATH
LIBPATH
OS2386
LIBPATH
<<

LIBPATH.OBJ:                    LIBPATH.C
        ICC -C+ -Kb+ -Ss+ LIBPATH.C
```

**LIBPATH.DEF**

```
NAME LIBPATH WINDOWCOMPAT NEWFILES
DESCRIPTION 'LIBPATH Example
             Copyright (c) 1993-1995 by Arthur Panov
             All rights reserved.'
PROTMODE
```

The first step is to find the system boot drive. In order to do this, use *DosQuerySysInfo* and specify the arguments corresponding to the boot drive information. *DosQuerySysInfo* takes three input parameters and one output parameter, and returns the values of the system's static variables:

```
APIRET = DosQuerySysInfo (    ULONG ulStartIndex,
                              ULONG ulLastIndex,
                              PVOID pDataBuf,
                              ULONG ulDataBufLen);
```

This call can return a single value or a range of values, depending on the *ulStartIndex*, *ulLastIndex*. As is evident by the example, in order to obtain a single value, the *ulStartIndex* and *ulLastIndex* are set to the same input value:

```
arReturn = DosQuerySysInfo (  QSV_BOOT_DRIVE,
                              QSV_BOOT_DRIVE,
                              &ulDrive,
                              sizeof ( ulDrive )) ;
```

The QSV_BOOT_DRIVE constant is defined by the BSEDOS.H header file, which is part of the set of standard header files provided by the Programmer's Toolkit. Table 4.1 defines the additional values. The third parameter is the data buffer that *DosQuerySysInfo* uses to place the returned values into. The last parameter is the size of the data buffer.

Table 4.2 System Constants

| Description | Value | Meaning |
|---|---|---|
| QSV_MAX_PATH_LENGTH | 1 | Maximum path name length in bytes |
| QSV_MAX_TEXT_SESSIONS | 2 | Maximum number of text sessions |
| QSV_MAX_PM_SESSIONS | 3 | Maximum number of PM sessions |
| QSV_MAX_VDM_SESSIONS | 4 | Maximum number of virtual DOS machine (VDM) sessions |
| QSV_BOOT_DRIVE | 5 | Boot drive value (0=A:, 1=B:, etc.) |
| QSV_DYN_PRI_VARIATION | 6 | Dynamic/Absolute priority (0=absolute) |
| QSV_MAX_WAIT | 7 | Maximum wait time in seconds |
| QSV_MIN_SLICE | 8 | Minimum time slice allowed in milliseconds |
| QSV_MAX_SLICE | 9 | Maximum time slice allowed in milliseconds |
| QSV_PAGE_SIZE | 10 | Default page size (4K) |
| QSV_VERSION_MAJOR | 11 | Major version number |
| QSV_VERSION_MINOR | 12 | Minor version number |
| QSV_VERSION_REVISION | 13 | Revision version letter |
| QSV_MS_COUNT | 14 | The value of a free-running 32-bit counter in milliseconds (value=0 at boot time) |
| QSV_TIME_LOW | 15 | Lower 32 bits of time since 01-01-1980 in seconds |
| QSV_TIME_HIGH | 16 | Upper 32 bits of time since 01-01-1980 in seconds |
| QSV_TOTPHYSMEM | 17 | Total number of pages of physical memory (4K each) |
| QSV_TOTRESMEM | 18 | Total number of system-resident memory |
| QSV_TOTAVAILMEM | 19 | Total number of pages available for allocation to the system at the instance of the call |
| QSV_MAXPRMEM | 20 | Total number of pages available for allocation to the process at the instance of the call |
| QSV_MAXSHMEM | 21 | Total number of shareable pages available to the caller in the shared area |
| QSV_TIMER_INTERVAL | 22 | Timer interval in 1/10 millisecond |
| QSV_MAX_COMP_LENGTH | 23 | Maximum length of a component's pathname in bytes |

**Gotcha!**
An application that is intended to be used in the HPFS/FAT environment should make the *DosQuerySysInfo* call and determine the maximum value of the legal file name length: QSV_MAX_COMP_LENGTH. For HPFS, this value is much greater than FAT (255). The application should issue this call in its initialization section and remember the pertinent values for future *DosFindFirst*, *DosFindNext* buffer size allocation values.

Once the boot drive is located, the string containing the full path to CONFIG.SYS is created.

## Getting the File Size

```
arReturn = DosQueryPathInfo ( pchFile,
                              FIL_STANDARD,
                              &fsStatus,
                              sizeof ( fsStatus )) ;

pchBuffer = malloc ( fsStatus.cbFileAlloc +1 ) ;
```

*DosQueryPathInfo* is used to determine the size of CONFIG.SYS. The function is designed to get file information for a file or subdirectory. The first parameter, *pchFile*, is the fully qualified path for the file. The second parameter is the level of information required. All we need for this example is standard file information, FIL_STANDARD. The information level determines the third parameter. If FIL_STANDARD is specified, a pointer to a FILESTATUS3 structure is used. The structure looks like this:

```
typedef struct _FILESTATUS3
{       FDATE  fdateCreation;
        FTIME  ftimeCreation;
        FDATE  fdateLastAccess;
        FTIME  ftimeLastAccess;
        FDATE  fdateLastWrite;
        FTIME  ftimeLastWrite;
        ULONG  cbFile;
        ULONG  cbFileAlloc;
        ULONG  attrFile;
} FILESTATUS3;
```

The FILESTATUS3 structure contains two fields of interest: *cbFile* and *cbFileAlloc*. The *cbFile* element contains the actual size of the file, start to finish, in bytes. The *cbFileAlloc*, on the other hand, contains the file system allocation unit (AU) size, whose value can be a multiple of 512, 2K, 4K, 8K, 16K, 32K, and so on, depending on the type of magnetic media used. HPFS and diskettes use 512-byte AUs, while the FAT AU size depends on the volume size. *cbFileAlloc* is of minimal value in applications, and the *cbFile* value should be used to allocate the required storage. Thus, *cbFile* size value is used in the next call to allocate the memory buffer needed to read the whole CONFIG.SYS at once, plus an extra byte for a NULL character.

This memory allocation does not have to be performed. It is possible to read one character at a time and parse the output using a 1-byte storage area. The method used in CHKEA was used for ease of implementation as well as performance reasons. Reading the whole file is much quicker. Since the CONFIG.SYS is generally smaller than 4K in size, it should easily fit into a single page of memory, which is the smallest allocation allowed in 32-bit OS/2. The parsing can be achieved more rapidly as well. Memory operations are much quicker than storage disk I/O.

## Opening a File

Having found the file size, the next step is to attempt to open the CONFIG.SYS file. The *DosOpen* API call is a good example of the flexibility and power of the OS/2 file system interface. Several flags are available to the programmer, and almost any combination of them can be defined in order to provide for maximum systemwide cooperation. In this case, the file is opened in read-only mode and with full sharing enabled. This means that if another application decided to open and read CONFIG.SYS at the same time, it would be able to do so. Allowing other applications full sharing rights also presents a problem of the file data being changed while we are attempting to read it. Although this is a remote possibility, the risk is still there; using the OPEN_SHARE_DENYWRITE flag instead of OPEN_SHARE_DENYNONE easily prevents it. The OPEN_FLAGS_SEQUENTIAL flag is used to define how we will be reading the file. Last, we examine the file in read-only mode by specifying the flag OPEN_ACCESS_READONLY. *DosOpen* is a fairly involved API. We'll go into some more details in just a moment.

```
arReturn = DosOpen ( pchFile,
                     &hfFile,
                     &ulAction,
                     0,
                     FILE_NORMAL,
                     FILE_OPEN,
                     OPEN_FLAGS_FAIL_ON_ERROR |
                     OPEN_FLAGS_SEQUENTIAL |
                     OPEN_SHARE_DENYNONE |
                     OPEN_ACCESS_READONLY,
                     NULL ) ;
```

## Reading a File

```
arReturn = DosRead ( hfFile,
                     pchBuffer,
                     fsStatus.cbFile,
                     &ulBytesRead ) ;
```

*DosRead* is the function to read not only files but any devices. The first parameter, *hfile,* is the file handle returned from *DosOpen.* The buffer, *pchBuffer*, is the second parameter. The third parameter is the number of bytes to read. In our case, the entire file size is used. The last parameter is a pointer to a ULONG. The number of bytes actually placed into the buffer is returned in a variable, *ulBytesRead.*

Note: In DOS and OS/2, it is possible to get a good return code (*arReturn*=0) and not have the *DosRead/DosWrite* API complete as expected. It is a good idea to check for the return code first, then check for the *BytesRead* value and compare it with the expected number.

Once in memory, the last character of the CONFIG.SYS file is set to NULL. This is done so that string operations can be performed more easily. The last step is parsing the file in order to find the LIBPATH information. Once the LIBPATH, is found it is displayed with a straightforward *printf.* The cleanup is accomplished by freeing the memory and using *DosClose* to close the file.

```
arReturn = DosClose ( hfFile ) ;
printf ( "\n%s\n", pchLibpath ) ;
free ( pchBuffer ) ;
```

## More on *DosOpen*

Before we continue with the EA example, it might be beneficial to cover the *DosOpen* API in greater detail.

```
APIRET DosOpen (          PSZ         pszFileName,
                          PHFILE      ppFileHandle,
                          PULONG      pActionTaken,
                          ULONG       ulFileSize,
                          ULONG       ulFileAttribute,
                          ULONG       ulOpenFlag,
                          ULONG       ulOpenMode,
                          PEAOP2      ppEABuf);
```

The first three arguments are clearly identified.

- *pszFileName* Input address of a string containing the file name
- *ppFileHandle* Output address of a returned file handle
- *pActionTaken* Output address of a specified action variable

The action variable on output will have the following useful values:

Table 4.3 Values of the Action Variable

| #define | Value | Meaning |
|---|---|---|
| FILE_EXISTED | 1 | File existed prior to call |
| FILE_CREATED | 2 | File was created as the result of the call |
| FILE_TRUNCATED | 3 | Existing file was changed by the call |

The next three input arguments can create the most confusion.

- *ulFileAttribute* Double word containing the files attributes
- *ulOpenFlag* Double word containing the desired open conditions
- *ulOpenMode* Double word containing the mode/sharing conditions

They create confusion because the same *DosOpen* call can be used to open files, disk volumes, pipes, and other devices. For example, if a user wanted to open a named pipe, some of the sharing flags and the *ulFileSize* value are ignored by the operating system because the pipe's buffer sizes are specified by the *DosCreateNPipe* API. Also, the *ulFileSize* may not make sense if the user is opening a disk volume for direct access. Sometimes device drivers allow *DosOpen* calls with a device name specified in place of the *pszFileName*. It is still a null-terminated string, but in the case of a device driver the string contains the device name, such as "DEVICE$". Specifying *ulFileSize* or other *ulFileAttribute* flags makes no sense, and thus some of the input parameters are ignored. All three input flag parameters are bit-encoded, meaning each bit that is set represents a new or unique flag condition. Most of the bits can be set in combination. All of the flags are 32 bits wide, but not all of the 32 bits are used at this time. Some are reserved for future use and must be set to zero. For example, the *ulFileAttribute* bit values are shown in Figure 4.1.

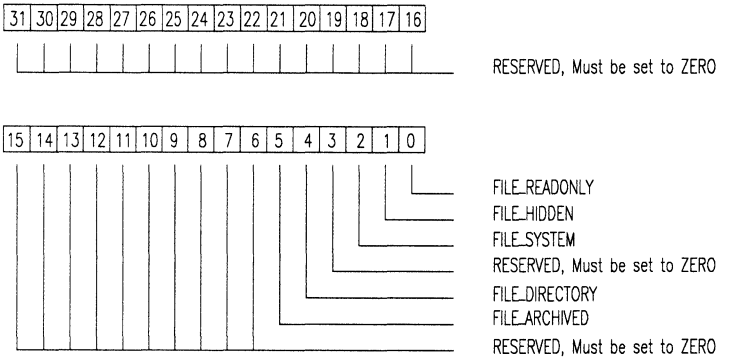



Figure 4.1 File attribute bit flags.

Table 4.4 describes the file attribute bit flags.

Table 4.4 File Attribute Bit Flag Descriptions

| Value | Description |
|---|---|
| FILE_READONLY | File can be read but not written to |
| FILE_NORMAL | File can be read and written to |
| FILE_HIDDEN | File is a hidden file |
| FILE_SYSTEM | File is a system file |
| FILE_DIRECTORY | File is a subdirectory |
| FILE_ARCHIVED | File has archive bit set |

To allow the file read-only access and to declare the file to be of the system type, the following combination is used.

```
ulFileAttribute =      FILE_READONLY | FILE_SYSTEM;
```

The *ulOpenFlag* describes the actions that the *DosOpen* will perform based on the bit encoding specified by the programmer. These actions deal with conditions of file existence, replacement, and creation. A user may want to allow the *DosOpen* API to fail, if the file already exists. If so, specify:

```
ulOpenFlag =   OPEN_ACTION_FAIL_IF_EXISTS;
```

If the user wants the *DosOpen* call to open the file if it exists, and fail if it does not exist, the following should be specified :

```
ulOpenFlag =   OPEN_ACTION_FAIL_IF_NEW |
               OPEN_ACTION_OPEN_IF_EXISTS;
```

Figure 4.2 depicts additional file open action flags.

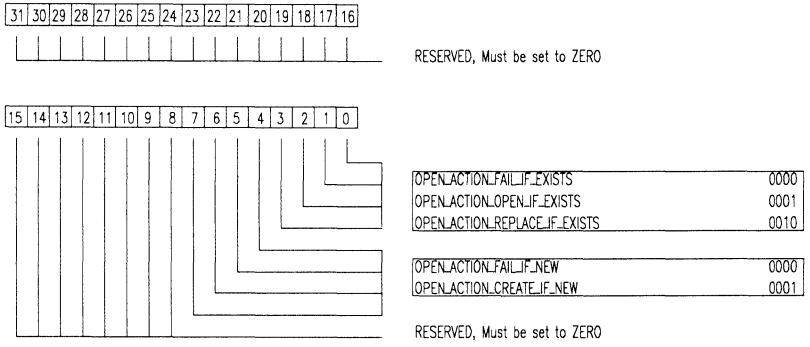



Figure 4.2 File open action flags.

Table 4.5 describes the file open action flags that are available.

Table 4.5 File Open Action Flags

| Value | Description |
|---|---|
| OPEN_ACTION_FAIL_IF_NEW | *DosOpen* will fail if file does not exist; file is opened if it does exist |
| OPEN_ACTION_CREATE_IF_NEW | File is created if it does not exist |
| OPEN_ACTION_FAIL_IF_EXISTS | *DosOpen* will fail if the file already exists; file is created |
| OPEN_ACTION_OPEN_IF_EXISTS | File is opened if it already exists |
| OPEN_ACTION_REPLACE_IF_EXISTS | File is replaced if it already exists |

The *ulOpenMode* describes the mode that the open call will set for the file object. This flag will tell the system how to behave when other users request access to the file that is currently in use by someone else. It is here that the system write-through buffering is specified and the error reporting is decided. For example, the user may want to allow the system to use its cache to transfer the data between the application and the file object, but the actual write must complete prior to the return of the call. Also, the user may want to have all of the errors reported directly to his or her application and not through the system critical-error handler. On top of that, a programmer may want to open this file in read-only mode and not allow anyone else write access to the file while in use. Wow! Well, for a combination of conditions like that, use the following flags:

```
ulOpenMode =    OPEN_FLAGS_WRITE_THROUGH
                |OPEN_FLAGS_FAIL_ON_ERROR
                |OPEN_SHARE_DENY_WRITE
                |OPEN_ACCESS_READONLY;
```

Thus, a number of conditions can be specified, and file management becomes a tedious and time-consuming task for the programmer and the operating system. Figure 4.3 depicts the available open mode flag.

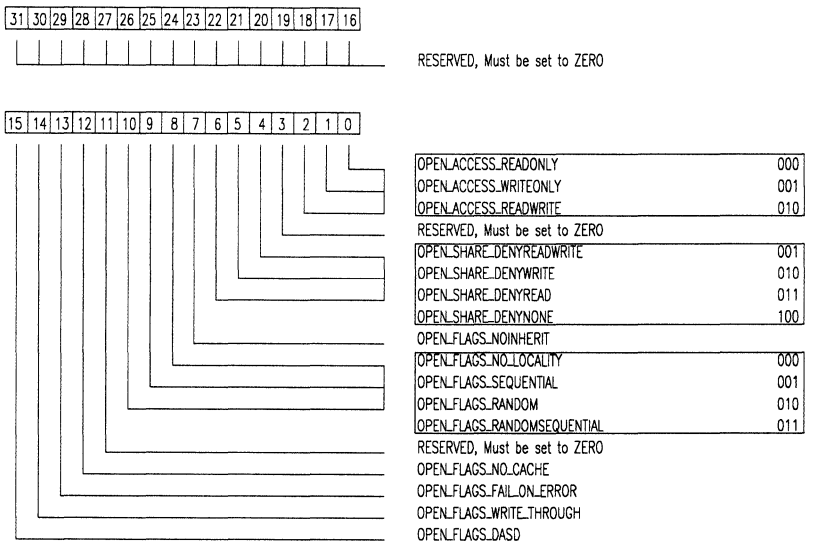



Figure 4.3 Open mode flags.

Table 4.6 describes the open mode flags avilable.

Table 4.6 Open Mode Flag Descriptions

| Value | Description |
|---|---|
| OPEN_ACCESS_READWRITE | File is given read/write access |
| OPEN_ACCESS_WRITEONLY | File is given only write access |
| OPEN_ACCESS_READONLY | File is given only read access |
| OPEN_SHARE_DENYNONE | Other processes can have read and write access to file |
| OPEN_SHARE_DENYREAD | Other processes cannot be given read access |
| OPEN_SHARE_DENYWRITE | Other processes cannot be given write access |
| OPEN_SHARE_DENYREADWRITE | Other processes cannot be given read or write access |
| OPEN_FLAGS_NOINHERIT | File handle is not inherited to spawned processes |
| OPEN_FLAG_RANDOMSEQUENTIAL | File is opened for both random and sequential access |
| OPEN_FLAG_RANDOM | File is opened for mainly random access |
| OPEN_FLAG_SEQUENTIAL | File is opened for mainly sequential access |
| OPEN_FLAG_NO_LOCALITY | File locality is not known |
| OPEN_FLAGS_NO_CACHE | No file data is placed in cache |
| OPEN_FLAGS_FAIL_ON_ERROR | Media I/O errors are reported by return code rather than through the system error handler |
| OPEN_FLAGS_WRITE_THROUGH | File writes may go through cache but will be completed before the write call returns |
| OPEN_FLAGS_DASD | File is a drive to be opened for direct access |

# An Extended Attribute Example: CHKEA.C

The next example, CHKEA.C, shows a way to find out if the file object has Extended Attributes associated with it. If so, then the query is made as to the size of all of the Extended Attributes that are attached. Last, the names of the types of the Extended Attributes are displayed, and the extended attribute data is dumped.

**CHKEA.C**

```
#define  INCL_DOSERRORS
#define  INCL_DOSFILEMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define  MAXEATYPE     3
CHAR *pszKeyEAName[] =
{

   ".LONGNAME",
   ".ICONPOS",
   ".TYPE"
} ;

typedef struct   _EAINFO
{
   USHORT           usEAType;

   USHORT           usEALength;
   BYTE             bEAData[1];
} EAINFO,*PEAINFO;

INT DumpEA(CHAR *pszFile,PFEA2 pszAttributeName);

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{

   CHAR             achPath[CCHMAXPATHCOMP];
   PCHAR            pchPath;
   ULONG            ulCount;
   HDIR             hdFile;
   APIRET           arReturn;
   FILEFINDBUF4     ffbFile;
   CHAR             achFile[CCHMAXPATHCOMP];
   PBYTE            pbBuffer;
   PFEA2            pdAttribute;

   if (usNumArgs != 2)
   {

      puts("Syntax:  CHKEA [filename]");
      puts("");
      puts("where \'filename\' is the name of a ");
      puts("file/directory and can contain wildcards.");
      return 1;

   }                                   /* endif                */

   /************************************************************/
   /* get full path name                                       */
   /************************************************************/

   DosQueryPathInfo(apchArgs[1],
                    FIL_QUERYFULLNAME,
                    achPath,
                    CCHMAXPATHCOMP);

   pchPath = strrchr(achPath,
                     '\\');

   if (pchPath != NULL)
   {

      pchPath++;
      *pchPath = 0;
```

```
   }                                    /* endif                */
   ulCount = 1;
   hdFile = HDIR_SYSTEM;

   arReturn = DosFindFirst(apchArgs[1],
                           &hdFile,
                           FILE_DIRECTORY,
                           &ffbFile,
                           sizeof(FILEFINDBUF4),
                           &ulCount,
                           FIL_QUERYEASIZE);

   while (arReturn == 0)
   {

      /*********************************************************/
      /* print out full path name                              */
      /*********************************************************/

      sprintf(achFile,
              "%s%s",
              achPath,
              ffbFile.achName);

      printf("\nFile name: %s\n",
             achFile);
      printf("\nTotal bytes allocated for EAs: %ld bytes.",
             ffbFile.cbList);

      /*********************************************************/
      /* allocate memory for ea buffer                         */
      /*********************************************************/

      pbBuffer = malloc(ffbFile.cbList);

      ulCount = -1;

      arReturn = DosEnumAttribute(ENUMEA_REFTYPE_PATH,
                                  achFile,
                                  1,
                                  pbBuffer,
                                  ffbFile.cbList,
                                  &ulCount,
                                  ENUMEA_LEVEL_NO_VALUE);

      printf("\nThis object contains %ld EAs.",
             ulCount);

      pdAttribute = (PFEA2)pbBuffer;

      while (ulCount != 0)
      {

         printf("\nFound EA with name \"%s\"",
                pdAttribute->szName);

         DumpEA(achFile,
                pdAttribute);

         ulCount--;
         pdAttribute = (PFEA2)(((PBYTE)pdAttribute)+
            pdAttribute->oNextEntryOffset);

      }                                 /* endwhile             */
      free(pbBuffer);

      ulCount = 1;
      arReturn = DosFindNext(hdFile,
```

```
                              &ffbFile,
                              sizeof(ffbFile),
                              &ulCount);

   }                                      /* endwhile                */
   if ((arReturn != 0) && (arReturn != ERROR_NO_MORE_FILES))
   {

      printf("\nError %ld encountered\n",
             arReturn);

   }                                      /* endif                   */
   return  arReturn;
}

int DumpEA(CHAR *pszFile,PFEA2 pdAttribute)
{
   APIRET            rc;
   USHORT            i;
   ULONG             ulGBufLen,ulFBufLen,ulEBufLen;
   PFEA2             pFEA2;
   EAOP2             eaopGet;
   PGEA2LIST         pGEA2List;
   ULONG             ulSize;
   ULONG             ulDataStart;
   PEAINFO           ptrEAData,ptrEADataHolder;

   for (i = 0; i < MAXEATYPE; i++)
   {

      /*********************************************************/
      /* does EA name match one of the EA's we're interested in*/
      /*********************************************************/

      if (!strcmp(pdAttribute->szName,
                  pszKeyEAName[i]))
      {

         /*******************************************************/
         /* build input/output data buffer first build          */
         /* fpFEA2List structure                                */
         /*******************************************************/

         ulFBufLen = sizeof(FEA2LIST)+pdAttribute->cbName+1+/*
                                          actual name        */
            pdAttribute->cbValue;       /* actual value       */
         pFEA2 = (PFEA2)calloc(1,
                               ulFBufLen);
         if (!pFEA2)
            return  FALSE;

         /*******************************************************/
         /* only one pFEA2 attribute in this list               */
         /*******************************************************/

         eaopGet.fpFEA2List = (FEA2LIST *)pFEA2;
         eaopGet.fpFEA2List->cbList = ulFBufLen;

         /*******************************************************/
         /* next build fpGEA2List structure                     */
         /*******************************************************/

         ulGBufLen = sizeof(GEA2LIST)+pdAttribute->cbName+1;

         pGEA2List = (GEA2LIST *)calloc(1,
                                        ulGBufLen);
         if (!pGEA2List)
         {
            free(pFEA2);
```

```
   return  FALSE;
}

/*****************************************************/
/* initialize fpGEA2List                             */
/*****************************************************/

pGEA2List->cbList = ulGBufLen;
pGEA2List->list[0].oNextEntryOffset = 0;
pGEA2List->list[0].cbName = pdAttribute->cbName;
strcpy(pGEA2List->list[0].szName,
       pdAttribute->szName);
eaopGet.fpGEA2List = (GEA2LIST *)pGEA2List;

/*****************************************************/
/* get EA's                                          */
/*****************************************************/

ulEBufLen = ulFBufLen+ulGBufLen;
rc = DosQueryPathInfo(pszFile,
                      FIL_QUERYEASFROMLIST,
                      (PVOID)&eaopGet,
                      ulEBufLen);

if (!rc)
{
   printf("\nEA Data for EA %s  ",
          pdAttribute->szName);

   ulSize = sizeof(FEA2LIST);

   /**************************************************/
   /* get the first list                             */
   /**************************************************/

   pFEA2 = (PFEA2)eaopGet.fpFEA2List->list;

   /**************************************************/
   /* offset to start of EAdata                      */
   /**************************************************/

   ulDataStart = ulSize+pFEA2->cbName;
   ptrEAData = (PEAINFO)((PBYTE)eaopGet.fpFEA2List+
      ulDataStart);

   /**************************************************/
   /* allocate memory with space for null            */
   /**************************************************/

   ptrEADataHolder = calloc(1,
                            sizeof(EAINFO)+
                               ptrEAData->usEALength+1);
   printf("Type = 0x%x  ",
          ptrEAData->usEAType);
   printf("Length = 0x%x",
          ptrEAData->usEALength);

   /**************************************************/
   /* move Data into placeholder memory              */
   /**************************************************/

   memcpy(ptrEADataHolder,
          ptrEAData->bEAData,
          ptrEAData->usEALength);
   printf("\nData = %s",
          ptrEADataHolder);
   free(ptrEADataHolder);
}
```

```
        free(pFEA2);
        free(pGEA2List);

        if (rc)
        {
           printf("\nDosQueryPathInfo failed, rc = %d",
                  rc);
           return  FALSE;
        }
     }                                  /* if EA is one of the
                                           key types           */
  }                                     /* loop through all EA
                                           key types           */
  return  TRUE;
}
```

## CHKEA.MAK

```
CHKEA.EXE:                      CHKEA.OBJ
        LINK386 @<<
CHKEA
CHKEA
CHKEA
OS2386
CHKEA
<<

CHKEA.OBJ:                      CHKEA.C
        ICC -C+ -Kb+ -Ss+ CHKEA.C
```

## CHKEA.DEF

```
NAME CHKEA WINDOWCOMPAT NEWFILES
DESCRIPTION 'Extended Attribute Example
            Copyright (c) 1993-1995 by Arthur Panov
            All rights reserved.'
PROTMODE
```

CHKEA.EXE expects a command-line input argument that is the name of the file of interest. Wildcard characters are accepted. First, a determination is made if the file object can be located on the hard disk; if successful, the full name of the object is constructed.

```
DosQueryPathInfo ( apchArgs[1],
                   FIL_QUERYFULLNAME,
                   achPath,
                   CCHMAXPATHCOMP ) ;
pchPath = strrchr ( achPath, '\\' ) ;

if ( pchPath != NULL ) {

   pchPath++ ;
   *pchPath = 0 ;

} /* endif */

ulCount = 1 ;
hdFile = HDIR_SYSTEM ;
```

```
arReturn = DosFindFirst ( apchArgs[1],
                          &hdFile,
                          FILE_DIRECTORY,
                          &ffbFile,
                          sizeof ( FILEFINDBUF4 ),
                          &ulCount,
                          FIL_QUERYEASIZE ) ;
```

The *DosFindFirst* API is the most useful function call available to a programmer when attempting to locate file objects.

```
APIRET APIENTRY  DosFindFirst(PSZ           pszFileSpec,
                              PHDIR         phdir,
                              ULONG         flAttribute,
                              PVOID         pfindbuf,
                              ULONG         cbBuf,
                              PULONG        pcFileNames,
                              ULONG         ulInfoLevel);
```

The definition for this API can be found in the BSEDOS.H header file, which is part of the OS/2 Developer's Toolkit. Table 4.7 presents the arguments of interest.

Table 4.7 Arguments of *DosFindFirst*

| Arguments | Value(s) | Meaning |
|---|---|---|
| *phdir* | HDIR_SYSTEM | Use STDOUT for handle |
| *phdir* | HDIR_CREATE | Handle is created |
| *flAttribute* | bit encoded | Type of object to search for |
| *pfindbuf* | depends on ulInfoLevel | Result of the request |
| *ulInfoLevel* | FIL_STANDARD | Standard file information is returned |
| *ulInfoLevel* | FIL_QUERYEASIZE | File EA size is returned |
| *ulInfoLevel* | FIL_QUERYEASFROMLIST | Actual EA data is returned |

Table 4.8 lists the acceptable values for the *flAttribute* argument.

Table 4.8 Acceptable Values for *flAttribute*

| Value | Description |
|---|---|
| MUST_HAVE_ARCHIVED | Files returned must have the archive bit set |
| MUST_HAVE_DIRECTORY | Files returned must have the directory bit set |
| MUST_HAVE_SYSTEM | Files returned must have the system bit set |
| MUST_HAVE_HIDDEN | Files returned must have the hidden bit set |
| MUST_HAVE_READONLY | Files returned must have the read-only bit set |
| FILE_ARCHIVED | Files with archive bit set are not returned unless this value is specified |
| FILE_DIRECTORY | Files with directory bit set are not returned unless this value is specified |
| FILE_SYSTEM | Files with system bit set are not returned unless this value is specified |
| FILE_HIDDEN | Files with hidden bit set are not returned unless this value is specified |
| FILE_READONLY | Files with read-only bit set are not returned unless this value is specified |

*phdir* is an input/output parameter. On input it specifies the type of file handle required by the application. HDIR_SYSTEM tells the operating system to assign a handle that will always be available to the process. This is a handle for standard output. HDIR_CREATE will cause the system to allocate a handle and return it to the application in *phdir*. Since *pszFileSpec* can accept wildcard characters, the handle returned can be used in conjunction with the *DosFindNext* to find the next file object that matches the *pszFileSpec*.

*flAttribute* is an input bit-encoded flag that tells *DosFindFirst* what types of file objects to look for. These bits represent conditions that may be true or must be true. For example, a programmer may want to locate a directory with a particular name that may be hidden; although there are files that can correspond to the *pszFileSpec* specified, only the directories are of interest. The following bit combination could be used.

```
flAttribute =  MUST_HAVE_DIRECTORY | FILE_HIDDEN;
```

The *pfindbuf* is a pointer to the buffer that must be allocated prior to making the *DosFindFirst* call, and it must be passed to the API as a pointer. On output the buffer will contain the information specified by the next parameter *ulInfoLevel*, which can have three valid values associated with it (FIL_STANDARD, FIL_QUERYEASIZE, FIL_QUEARYEASFROMLIST).

The first value requests *DosFindFirst* to return FIL_STANDARD information about the file. FIL_STANDARD information contains the data associated with the FILEFINDBUF3 structure.

FIL_QUERYEASIZE information is requested by specifying FIL_QUERYEASIZE for the *ulInfoLevel,* and it returns the data associated with the FILEFINDBUF4 structure. Finally, FIL_QUERYEASFROMLIST information is obtained by specifying the value FIL_QUERYEASFROMLIST for the *ulInfoLevel.* It returns an EAOP2 data structure.

The FIL_QUERYEASFROMLIST request is slightly different from the previous two levels. On input *pfindbuf* must contain the EAOP2 data structure with the correct names of the EAs to be queried. Since EA data structures are variable in length, the *fpGEA2List* must contain a pointer to the GEA2 list, which in turn must have the correct value specified for the *oNextEntryOffset* and *szName*. The *szName* specifies the EA to be returned, and the *oNextEntryOffset* contains the number of bytes from the beginning of the first entry to the end of the next entry. On output the EAOP2 contains a pointer to the *fpFEA2List.* The *fpFEA2List* points to the list of FEA2 structures that have the actual EA information. All of the input records must be aligned on a two-word boundary, and the last in the list of GEA2 structures *oNextEntryOffset* value must be set to zero. The following are the various data buffers that are returned depending on the level of information requested.

- FIL_STANDARD Output generally contains the basic file information without EAs.

```
typedef struct _FILEFINDBUF3
        {
        ULONG   oNextEntryOffset;
        FDATE   fdateCreation;
        FTIME   ftimeCreation;
        FDATE   fdateLastAccess;
        FTIME   ftimeLastAccess;
        FDATE   fdateLastWrite;
        FTIME   ftimeLastWrite;
        ULONG   cbFile;
        ULONG   cbFileAlloc;
        ULONG   attrFile;
        UCHAR   cchName;
        CHAR    achName[CCHMAXPATHCOMP];
        }  FILEFINDBUF3;
```

- FIL_QUERYEASIZE Output contains the same information as FIL_STANDARD plus EA size.

```
typedef struct _FILEFINDBUF4
        {
        ULONG   oNextEntryOffset;
        FDATE   fdateCreation;
        FTIME   ftimeCreation;
        FDATE   fdateLastAccess;
        FTIME   ftimeLastAccess;
        FDATE   fdateLastWrite;
        FTIME   ftimeLastWrite;
        ULONG   cbFile;
        ULONG   cbFileAlloc;
        ULONG   attrFile;
        ULONG   cbList;
        UCHAR   cchName;
        CHAR    achName[CCHMAXPATHCOMP];
        }  FILEFINDBUF4;
```

The *cbList* field contains the size of the entire EA set for this file object, in bytes.

- FIL_QUERYEASFROMLIST Input contains the GEA2 information. Output contains the FEA2 information.

```
typedef struct _GEA2LIST
        {
        ULONG   cbList;
        GEA2    list[1];
        }  GEA2LIST;

typedef GEA2LIST * PGEA2LIST;

typedef struct _GEA2
        {
        ULONG   oNextEntryOffset;
        BYTE    cbName;
        CHAR    szName[1];
        }  GEA2;

typedef struct _FEA2LIST
        {
        ULONG   cbList;
        FEA2    list[1];
        }  FEA2LIST;
typedef FEA2LIST * PFEA2LIST;

typedef struct _FEA2
        {
        ULONG   oNextEntryOffset;
        BYTE    fEA;
        BYTE    cbName;
        USHORT  cbValue;
        CHAR    szName[1];
        }  FEA2;

typedef struct _EAOP2
        {
        PGEA2LIST   fpGEA2List;
        PFEA2LIST   fpFEA2List;
        ULONG       oError;
        }  EAOP2;
```

Figure 4.4 illustrates the EAOP2 structure in memory.

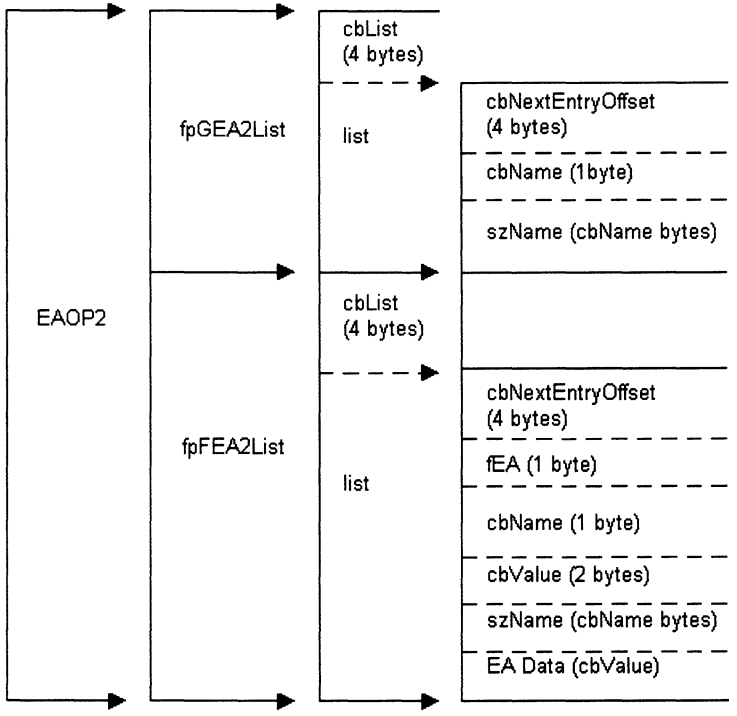



Figure 4.4 Map of EAOP2 memory buffer

*DosFindFirst* also accomplishes one other thing. It provides us with the size of the EAs associated with the file. A buffer of this size, *pbBuffer*, is allocated.

*DosEnumAttribute* is used to identify the names and sizes of the EAs associated with a particular file object.

```
APIRET APIENTRY  DosEnumAttribute(    ULONG  ulRefType,
                                      PVOID  pvFile,
                                      ULONG  ulEntry,
                                      PVOID  pvBuf,
                                      ULONG  cbBuf,
                                      PULONG pulCount,
                                      ULONG  ulInfoLevel);
```

The *ulRefType* tells the *DosEnumAttribute* about the next input parameter. When the value is 0, the *pvFile* argument contains a file handle; when the value is 1, the *pvFile* argument contains a pointer to a null-terminated string representing the name of the file object.

If the *pvFile* contains a handle, then this handle must be obtained by an earlier call to a *DosOpen* or similar API.

*ulEntry* describes the ordinal of the file object's EA entry. This value must be non-zero and positive. The value of 1 is indicative of the first EA entry in the list, 2 of the second one, and so on.

*pvBuf* is the pointer to the output buffer. Only FIL_STANDARD information can be returned; thus the ulInfoLevel is always 1 (ENUMEA_LEVEL_NO_VALUE).
*cbBuf* is the length of the buffer referenced by the *pvBuf*.

*cbBuf* is the length of the buffer referenced by the *pvBuf.*

*pulCount* is an input/output type argument. On input, the value contains the number of EAs for which the information is requested. If the value of -1L is specified, all of the EAs are queried, and the information is returned in the *pvBuf*, provided the buffer is of adequate size. On output this argument contains the actual number of EAs for which the information was returned. If the buffer is big enough, all of the requested EAs for the file will be returned. On output the buffer contains the list of those FEA2 structures that are aligned on a two-word boundary. The last structure in the list will have the *oNextEntryOffset* value of zero.

```
typedef struct _FEA2
        (
        ULONG   oNextEntryOffset;
        BYTE    fEA;
        BYTE    cbName;
        USHORT  cbValue;
        CHAR    szName[1];
        }  FEA2;

      arReturn = DosEnumAttribute ( ENUMEA_REFTYPE_PATH,
                                    achFile,
                                    1,
                                    pbBuffer,
                                    ffbFile.cbList,
                                    &ulCount,
                                    ENUMEA_LEVEL_NO_VALUE ) ;

      printf ( "\nThis object contains %ld EAs.\n", ulCount ) ;
```

In this example, *DosEnumAttribute* uses a "1" as the EA ordinal, indicating the function is to start enumerating at the first EA. Since *pbBuffer* is big enough to hold all the EA, it should all be placed in the buffer after just one function call to *DosEnumAttribute.*

```
pdAttribute = (PFEA2)pbBuffer;

while (ulCount != 0)
{

   printf("\nFound EA with name \"%s\"",
          pdAttribute->szName);

   DumpEA(achFile,
          pdAttribute);

   ulCount--;
   pdAttribute = (PFEA2)(((PBYTE)pdAttribute)+
      pdAttribute->oNextEntryOffset);

}                                  /* endwhile              */
```

Once the EAs are enumerated, a *while* loop is used to loop through and list each EA. The user function *DumpEA* is covered in more detail later. The next EA is found by adding the *oNextEntryOffset* to the *pbBuffer* pointer. Notice the casting involved here. Remember, additions should be made in PBYTE-increments, not in PFEA2-increments.

```
      arReturn = DosFindNext ( hdFile,
                               &ffbFile,
                               sizeof ( ffbFile ),
                               &ulCount ) ;
```

Once all the EAs are listed for one file object, *DosFindNext* is used to find the next file object that matches the wildcard criteria.

In order to obtain the values of the EAs, Level FIL_QUERYEASFROMLIST information should be specified and *DosQueryFileInfo* or *DosQueryPathInfo* should be used. Also, it is important to remember that while one process is reading the EA information, another one can be changing it. To prevent this from becoming a problem, the application must open a file with the sharing flag set to the deny-write state. This will prevent another user from changing the information in the EAs while in use. Note that the *DosEnumAttribute* may return a different EA for the same specified ordinal number, because ordinals are assigned only to the existing EAs. An application can delete an EA, then turn around and write another one in its place. The ordinal numbers are not preserved, and thus are not unique. The following formula (from the *OS/2 2.1 Control Program Programming Reference* manual) shows the information needed to calculate the required buffer size.

The buffer size is calculated as follows:

4 bytes (for *oNextEntryOffset*) +
1 byte (for *fEA*) +
1 byte (for *cbName*) +
2 bytes (for *cbValue*) +
Value of *cbName* (for the name of EA) +
1 byte (for NULL in *cbName*) +
Value of *cbValue*(for the value of EA)

**Gotcha!**
Each EA list entry *must* start on a double-word boundary.

The *DumpEA* function checks the FEA2 structure to see if the EA matches the types, .LONGNAME, .ICONPOS, or .TYPE. These types were selected as examples, simply because each is an ASCII string.

```
ulFBufLen = sizeof(FEA2LIST)+pdAttribute->cbName+1+  /* actual name          */
                             pdAttribute->cbValue;     /* actual value         */

pFEA2 = (PFEA2)calloc(1, ulFBufLen);
if (!pFEA2)
   return  FALSE;

/*****************************************************/
/* only one pFEA2 attribute in this list             */
/*****************************************************/

eaopGet.fpFEA2List = (FEA2LIST *)pFEA2;
eaopGet.fpFEA2List->cbList = ulFBufLen;
```

The first step is building the *fpFEA2List* structure for input. The size of the buffer is calculated by adding the structure size, plus the size of the EA name, *cbName*, plus the size of the EA data, *cbValue*, plus one byte for a '\0' appended to the name. The *fpFEA2List* structure in the *eaopGet* structure is set equal to the memory that has been allocated. The only other initialization involved is setting *cbList* equal to the size of the output buffer.

```
ulGBufLen = sizeof(GEA2LIST)+pdAttribute->cbName+1;
pGEA2List = (GEA2LIST *)calloc(1,
                               ulGBufLen);
if (!pGEA2List)
{
   free(pFEA2);
   return  FALSE;
}

pGEA2List->cbList = ulGBufLen;
pGEA2List->list[0].oNextEntryOffset = 0;
pGEA2List->list[0].cbName = pdAttribute->cbName;
strcpy(pGEA2List->list[0].szName,
       pdAttribute->szName);
eaopGet.fpGEA2List = (GEA2LIST *)pGEA2List;
```

The *fpGEA2List* structure is used to tell the *DosQuery* functions which EAs the programmer is interested in. The buffer size is calculated like the *fpFEA2List* buffer. The offset to the next list entry is set to 0, because this example is looking for only one EA at a time. The *cbList* variable is the buffer size. The *cbName* variable is the EA name string size. The actual name is copied into the *szName* buffer. The last assignment is setting *fpGEA2List* in the *eaopGet* structure equal to the *pGEA2List* structure that has just been created.

*DosQueryPathInfo* is used to retrieve the actual EA data. The prototype for the function is:

```
APIRET DosQueryPathInfo( PSZ pszPathName, ULONG ulInfoLevel,
                         PVOID pInfoBuf, ULONG cbInfoBuf)
```

The first parameter is the filename to use to query the information. The second parameter is the level of information to retrieve. The value FIL_QUERYEASFROMLIST will retrieve the EA information. The third parameter is a pointer to the EAOP2 structure. The last parameter is the size of the EAOP2. This value is equal to the size of the *fpFEA2List* structure plus the size of the *fpGEA2List* structure.

```
rc = DosQueryPathInfo(pszFile,
                      FIL_QUERYEASFROMLIST,
                      (PVOID)&eaopGet,
                      ulEBufLen);
ulSize = sizeof(FEA2LIST);

pFEA2 = (PFEA2)eaopGet.fpFEA2List->list;

ulDataStart = ulSize+pFEA2->cbName;
ptrEAData = (PEAINFO)((PBYTE)eaopGet.fpFEA2List+
   ulDataStart);

ptrEADataHolder = calloc(1,
                         sizeof(EAINFO)+
                            ptrEAData->usEALength+1);
printf("\nType = %x",
       ptrEAData->usEAType);
printf("\nLength = %x",
       ptrEAData->usEALength);
memcpy(ptrEADataHolder,
       ptrEAData->bEAData,
       ptrEAData->usEALength);
printf("\nData = %s",
       ptrEADataHolder);
```

The last step in the *DumpEA* function is actually to print out the EA data. The data is returned in the *fpFEA2List* structure that was set up on input. First, the offset into the *fpFEA2List* where the EA data is

located is found by adding the size of the *FEA2* structure plus the size of the attribute name. If this sounds confusing, take a look at Figure 4.4 on page 49 to help illustrate this. The EA data is formatted in the following manner. The first USHORT contains the type of EA data. The second USHORT contains size of the EA data. All the bytes that follow contain the actual data located in that EA. This data is copied into a memory buffer that contains enough space for a '\0' character at the end. The EA data does *not* contain the '\0' character at the end of the data, because not all EA data is in the form of an ASCII null-terminated string.

# Chapter 5

# Interprocess Communication

OS/2 provides several different methods of interprocess communication that are all fairly easy to implement. In OS/2 1.x there were five distinct ways available for a process to communicate with another process. These communications methods used flags, semaphores, pipes, queues, and shared memory to send and receive messages and signals. Four of the most common methods were retained in OS/2 2.0; the one that was dropped was the *DosFlagProcess* API. The functionality provided by *DosFlagProcess* is now provided by *DosRaiseException* and related APIs.

The easiest interprocess communication (IPC) method to implement is unnamed and named pipes. An unnamed pipe is a circular memory buffer that can be used to communicate between related processes. The parent process must set the inheritance flags to true in order for the child process to inherit the handles and allow the parent and the child processes to communicate. Communication is bidirectional, and the pipe remains open until both the read handle and the write handle are closed. Named pipes are also an easy way to provide remote communication. A process on the requester workstation can communicate with a process running on the server workstation as well as with a process running locally. However, the client-server remote connectivity can be achieved only with the help of some type of local area network server.

## An OS/2 Named Pipe Client-Server Example

SERVER.C is, as the name suggests, the server of the Named Pipe IPC mechanism. The program allows remote and local communications and performs simple character redirection. The characters are highlighted in different colors to distinguish server and client modes of operation. As the user types in characters at the client, they immediately echo on the server. There is no implied limitation that the server can receive only, and the client can send only. The particular implementation is specific to this example.

The SERVER.EXE application can be started by simply typing *Server* followed by a carriage return from the command line. This will start the server component of the program pair. The *Server* must be started first, since it is the *Server* that creates the named pipe and allows the *Client* to connect to it. After the server starts successfully, the *Client* can be started by typing *Client [ServerName]* followed by a carriage return from the command line. Note that the *[ServerName]* is an optional parameter and is used only if a remote pipe connection is being attempted. If the *Server* and the *Client* are running in the same workstation, and the workstation is capable of running the IBM OS/2 LAN Server software, the *Client-Server* communication can be achieved with both local and remote connections. However, if the IBM OS/2 LAN Server is not active, or the user is not logged on to the IBM OS/2 LAN Server domain, attempting a remote connection will produce an error stating that the pipe name was not found. This is correct, and usually points to an inactive server or an unauthorized user. The best way to look at this example is to open two OS/2 window sessions and to allow one session to run the SERVER.EXE and the other to run the CLIENT.EXE. This way it will be easier to see the *Client-Server* communication.

**SERVER.C**

```
#define  INCL_DOSNMPIPES
#define  INCL_DOSPROCESS
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include "common.h"
INT main(void)

{
   USHORT           *usHiCh;
   USHORT           *usLoCh;
   ULONG            ulHiLoCh;
   APIRET           arReturn;
   ULONG            ulBytesDone;

   arReturn = DosExitList(EXLST_ADD,
                          CleanUp);

   ulHiLoCh = 0;
   usLoCh = (USHORT *)&ulHiLoCh;
   usHiCh = usLoCh+1;
   chToken = SERVER_MODE;

   printf(SERVER_COLOR);
   printf(" Hit Ctrl+C to exit the server at any time \n");
   printf(" Starting the program in Server Mode...\n\n");

   arReturn = ConnFromClient();
   printf("\n  The Pipe Creation / Connection API "
      " returned rc = %02X\n\n",
          arReturn);

   while (!arReturn && (chToken != DISCON_MODE))
   {

      arReturn = RecvFromClient(&ulHiLoCh,
                                &ulBytesDone);

      if (ulHiLoCh == TOKEN_F3_DISCON || !ulBytesDone)
      {
         chToken = DISCON_MODE;
         break;
      }                                 /* endif                */
      if (*usLoCh == *usHiCh)
         putch(*usHiCh);
      else
         putch('*');

      if (*usLoCh == RETURN_CHAR)
      {
         putch(LINE_FEED_CHAR);
      }                                 /* endif                */
   }                                    /* endwhile             */
   arReturn = DosClose(hpPipe);
   printf(NORMAL_COLOR);
   return  arReturn;
}

APIRET ConnFromClient(VOID)
{
   CHAR             achInitBuf[HAND_SHAKE_LEN+1];
   ULONG            ulOpenMode;
   ULONG            ulPipeMode;
   ULONG            ulOutBufSize;
   ULONG            ulInpBufSize;
```

```
   ULONG            ulTimeOut;
   USHORT           arReturn;
   ULONG            ulBytesDone;

   memset(achInitBuf,
          0,
          sizeof(achInitBuf));

   ulOpenMode = DEFAULT_MAKE_MODE;
   ulPipeMode = DEFAULT_PIPE_MODE;
   ulOutBufSize = DEFAULT_OUTB_SIZE;
   ulInpBufSize = DEFAULT_INPB_SIZE;
   ulTimeOut = DEFAULT_TIME_OUTV;

   arReturn = DosCreateNPipe(DEFAULT_PIPE_NAME,
                             &hpPipe,
                             ulOpenMode,
                             ulPipeMode,
                             ulOutBufSize,
                             ulInpBufSize,
                             ulTimeOut);

   if (!arReturn)
   {
      printf(" You can start the CLIENT program now.\n");
      printf(" Typing in the CLIENT window will make\n");
      printf(" the keystrokes echo in this SERVER window\n\n");
      arReturn = DosConnectNPipe(hpPipe);
      if (!arReturn)
      {
         arReturn = DosRead(hpPipe,
                            achInitBuf,
                            (ULONG)HAND_SHAKE_LEN,
                            &ulBytesDone);

         if (!strcmp(achInitBuf,
                     HAND_SHAKE_INP) && !arReturn)
         {
            arReturn = DosWrite(hpPipe,
                                HAND_SHAKE_OUT,
                                strlen(HAND_SHAKE_OUT),
                                &ulBytesDone);
         }
         else
         {
            arReturn = HAND_SHAKE_ERROR;
         }                                  /* endif                */
      }                                     /* endif                */
   }                                        /* endif                */
   return  arReturn;
}

APIRET RecvFromClient(PULONG pulHiLoCh,PULONG pulBytesDone)
{

   return  DosRead(hpPipe,
                   pulHiLoCh,

                   sizeof(pulHiLoCh),
                   pulBytesDone);

}

VOID APIENTRY CleanUp(ULONG ulTermCode)
{
#define  MY_STDOUT     1
   ULONG            ulBytesDone;

   DosClose(hpPipe);
```

```
   DosWrite(MY_STDOUT,
            NORMAL_COLOR,
            strlen(NORMAL_COLOR),
            &ulBytesDone);

   DosExitList(EXLST_EXIT,
               0);

}
```

**SERVER.H**

```
#define SERVER_MODE             1
#define CLIENT_MODE             2
#define SERVER_COLOR            "\n•[0;32;40m"
#define CLIENT_COLOR            "\n•[0;31;40m"
#define NORMAL_COLOR            "\n•[0;37;40m"
#define REMOTE_PIPE             2
#define DISCON_MODE             3
#define BAD_INPUT_ARGS          99
#define MAX_PIPE_NAME_LEN       80
#define MAX_SERV_NAME_LEN       8
#define DEFAULT_PIPE_NAME       "\\PIPE\\MYPIPE"
#define DEFAULT_MAKE_MODE       NP_ACCESS_DUPLEX
#define DEFAULT_PIPE_MODE       NP_WMESG | NP_RMESG | 0x01
#define DEFAULT_OPEN_FLAG       OPEN_ACTION_OPEN_IF_EXISTS
#define DEFAULT_OPEN_MODE       OPEN_FLAGS_WRITE_THROUGH | \
                                OPEN_FLAGS_FAIL_ON_ERROR | \
                                OPEN_FLAGS_RANDOM |        \
                                OPEN_SHARE_DENYNONE |      \
                                OPEN_ACCESS_READWRITE
#define DEFAULT_OUTB_SIZE       0x1000
#define DEFAULT_INPB_SIZE       0x1000
#define DEFAULT_TIME_OUTV       20000L
#define TOKEN_F2_SWITCH         0x0000003CL
#define TOKEN_F3_DISCON         0x0000003DL
#define RETURN_CHAR             0x0D
#define LINE_FEED_CHAR          0x0A
#define FUNC_KEYS_CHAR          0x00
#define EXTD_KEYS_CHAR          0xE0
#define HAND_SHAKE_LEN          0x08
#define HAND_SHAKE_INP          "pIpEtEsT"
#define HAND_SHAKE_OUT          "PiPeTeSt"
#define HAND_SHAKE_ERROR        101
#define PROGRAM_ERROR           999

CHAR    achPipeName [MAX_PIPE_NAME_LEN] ;
HPIPE   hpPipe ;
CHAR    chToken ;

USHORT BadArgs ( USHORT usNumArgs, PCHAR apchArgs [] ) ;
APIRET ConnToClient ( VOID ) ;
APIRET ConnToServer ( VOID ) ;
APIRET SendToClient ( ULONG ulHiLoCh ) ;
APIRET RecvFromServer ( PULONG pulHiLoCh ) ;
```

**SERVER.DEF**

```
NAME SERVER WINDOWCOMPAT
DESCRIPTION 'SERVER example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

First, a *DosExitList* call is made in order to allow the SERVER.EXE to clean up properly in an event of a Ctrl-/Ctrl-Brk condition.

```
APIRET DosExitList( ULONG ulOrderCode, PFNEXITLIST pfn )
```

*ulOrderCode* consists of two lower-order bytes that have meaning and a high-order word that must be 0. The lower-order byte can have the values listed in Table 5.1.

Table 5.1 Values for Lower-Order Byte of *ulOrderCode*

| Value | Description |
|---|---|
| EXLST_ADD | Add an address to the termination list |
| EXLST_REMOVE | Remove an address from the termination list |
| EXLST_EXIT | When termination processing completes, transfer to the next address on the termination list |

The high-order byte of the low-order word must be zero if EXLST_REMOVE, or EXLST_EXIT is specified. If, however, EXLST_ADD is specified, the high-order byte will indicate the invocation order.

The second parameter for *DosExitList* is an address of the routine to be executed—*pfn*.

The *CleanUp()* routine closes the named pipe handle and resets the window text color back to white on black.

Next, *ConnToClient()* must issue two calls: *DosCreateNPipe()* and *DosConnectNPipe()*. Issuing the *DosConnectNPipe()* call is what allows the client to perform a *DosOpen()* successfully. After the first few necessary setup APIs are called, a simple handshake operation is performed by reading a known string from the pipe and writing a known string back.

```
APIRET DosCreateNPipe(PSZ pszName,
                      PHPIPE pHpipe,
                      ULONG openmode,
                      ULONG pipemode,
                      ULONG cbInbuf,
                      ULONG cbOutbuf,
                      ULONG msec);
```

The *DosCreateNPipe()* API expects seven arguments. The first parameter, *DEFAULT_PIPE_NAME*, is an ASCII string that contains the name of the pipe to be created, *pszName*. The second is a pointer to the pipe handle that will be returned when the function returns. The next parameter is the open mode used for the pipe. The flag used in the example is NP_ACCESS_DUPLEX, which provides inbound and outbound communication. The fourth parameter is the pipe mode. This parameter is a set of bitfields that define the pipe mode. The flags used in this example are NP_WMESG | NP_RMESG | 0x01. These flags indicate the pipe can send and receive messages, and also that only one instance of the pipe can be created. The pipe can be created in either byte or message mode only. If a byte mode pipe is created, then *DosRead()* and *DosWrite()* must use byte stream mode when reading from or writing to the pipe. If a message mode pipe is created, then *DosRead()* and *DosWrite()* automatically will use the first two bytes of each message, called the header, to determine the size of the message. Message mode pipes can be read from and written to using byte or message streams. Byte mode pipes, on the other hand, can be used only in byte stream mode. If a message stream is used, the operating system will encode the message header without the user having to calculate the value. Care should be taken when deciding what size buffers should be used during communications. The transaction buffer should be two bytes greater than the largest expected message.

```
APIRET DosConnectNPipe(HPIPE hpipe);
```

The *DosConnectNPipe()* only takes one argument, the named pipe handle. At this point, the pipe is ready for a client connection.

**CLIENT.C**

```
#define  INCL_DOSNMPIPES
#define  INCL_DOSPROCESS
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include "common.h"
INT main(USHORT usNumArgs,PCHAR apchArgs[])

{
   USHORT            *usHiCh;
   USHORT            *usLoCh;
   ULONG             ulHiLoCh;
   APIRET            arReturn;

   arReturn = DosExitList(EXLST_ADD,
                          CleanUp);

   achPipeName[0] = 0;
   chToken = 0;
   ulHiLoCh = 0;
   usLoCh = (USHORT *)&ulHiLoCh;
   usHiCh = usLoCh+1;
   chToken = CLIENT_MODE;

   printf(CLIENT_COLOR);
   printf(" Hit F3 to exit client program. \n\n");
   printf(" Starting the program in Client Mode...\n\n");

   if (usNumArgs == REMOTE_PIPE)
   {
      sprintf(achPipeName,
              "\\\\%s",
              apchArgs[1]);
   }

   strcat(achPipeName,
          DEFAULT_PIPE_NAME);
   printf(" Connecting to pipe : %s\n\n",
          achPipeName);

   arReturn = ConnToServer();
   if (!arReturn)
   {
      printf(" You can start typing in this CLIENT window\n");
      printf(
       " and watch for your keystrokes in the SERVER window\n\n")
         ;
   }

   while (!arReturn && (chToken != DISCON_MODE))
   {

      *usHiCh = getch();
      if ((*usHiCh == FUNC_KEYS_CHAR) || (*usHiCh ==
         EXTD_KEYS_CHAR))
      {
         *usLoCh = getch();
      }
      else
```

```
      {
         *usLoCh = *usHiCh;
      }                                  /* endif                  */
      arReturn = SendToServer(ulHiLoCh);

      if (ulHiLoCh == TOKEN_F3_DISCON)
      {
         chToken = DISCON_MODE;
         break;
      }                                  /* endif                  */
      if (*usLoCh == *usHiCh)
         putch(*usHiCh);
      else
         putch('*');

      if (*usLoCh == RETURN_CHAR)
      {
         putch(LINE_FEED_CHAR);
      }                                  /* endif                  */
   }                                     /* endwhile               */
   arReturn = DosClose(hpPipe);
   printf(NORMAL_COLOR);
   return  arReturn;
}

APIRET ConnToServer(VOID)
{
   CHAR              achInitBuf[HAND_SHAKE_LEN+1];
   ULONG             ulOpenFlag;
   ULONG             ulOpenMode;
   ULONG             ulActionTaken;
   INT               arReturn;
   ULONG             ulBytesDone;

   memset(achInitBuf,
          0,
          sizeof(achInitBuf));

   ulOpenFlag = DEFAULT_OPEN_FLAG;
   ulOpenMode = DEFAULT_OPEN_MODE;

   arReturn = DosOpen(achPipeName,
                      &hpPipe,
                      &ulActionTaken,
                      0,
                      0,
                      ulOpenFlag,
                      ulOpenMode,
                      0);

   if (!arReturn)
   {
      arReturn = DosWrite(hpPipe,
                          HAND_SHAKE_INP,
                          strlen(HAND_SHAKE_INP),
                          &ulBytesDone);

      if (!arReturn)
      {
         arReturn = DosRead(hpPipe,
                            achInitBuf,
                            (ULONG)HAND_SHAKE_LEN,
                            &ulBytesDone);

         if (strcmp(achInitBuf,
                    HAND_SHAKE_OUT))
         {
            arReturn = HAND_SHAKE_ERROR;
         }                               /* endif                  */
```

```
      }                                    /* endif                */
   }                                       /* endif                */
   if (arReturn)
   {
      printf("\n  The Pipe Open / Connection API "
         "returned rc = %02x\n",
            arReturn);
      printf("\n  Make sure the Server is running.\n\n");
   }                                       /* endif                */
   return  arReturn;
}

APIRET SendToServer(ULONG ulHiLoCh)
{
   ULONG            ulBytesDone;

   return  DosWrite(hpPipe,
                    &ulHiLoCh,

                    sizeof(ulHiLoCh),
                    &ulBytesDone);
}

VOID APIENTRY CleanUp(ULONG ulTermCode)
{
#define  MY_STDOUT     1
   ULONG            ulBytesDone;

   DosClose(hpPipe);
   DosWrite(MY_STDOUT,
            NORMAL_COLOR,
            strlen(NORMAL_COLOR),
            &ulBytesDone);

   DosExitList(EXLST_EXIT,
               0);

}
```

## CLIENT.DEF

```
NAME CLIENT WINDOWCOMPAT
DESCRIPTION 'CLIENT example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

## COMMON.H

```
#define SERVER_MODE              1
#define CLIENT_MODE              2
#define SERVER_COLOR             "\n•[0;32;40m"
#define CLIENT_COLOR             "\n•[0;36;40m"
#define NORMAL_COLOR             "\n•[0;37;40m"
#define REMOTE_PIPE              2
#define DISCON_MODE              3
#define BAD_INPUT_ARGS           99
#define MAX_PIPE_NAME_LEN        80
#define MAX_SERV_NAME_LEN        8
#define DEFAULT_PIPE_NAME        "\\PIPE\\MYPIPE"
#define DEFAULT_MAKE_MODE        NP_ACCESS_DUPLEX
#define DEFAULT_PIPE_MODE        NP_WMESG | NP_RMESG | 0x01
#define DEFAULT_OPEN_FLAG        OPEN_ACTION_OPEN_IF_EXISTS
#define DEFAULT_OPEN_MODE        OPEN_FLAGS_WRITE_THROUGH | \
                                 OPEN_FLAGS_FAIL_ON_ERROR | \
                                 OPEN_FLAGS_RANDOM |        \
```

```
                                    OPEN_SHARE_DENYNONE |       \
                                    OPEN_ACCESS_READWRITE
#define DEFAULT_OUTB_SIZE           0x1000
#define DEFAULT_INPB_SIZE           0x1000
#define DEFAULT_TIME_OUTV           20000L
#define TOKEN_F3_DISCON             0x0000003DL
#define RETURN_CHAR                 0x0D
#define LINE_FEED_CHAR              0x0A
#define FUNC_KEYS_CHAR              0x00
#define EXTD_KEYS_CHAR              0xE0
#define HAND_SHAKE_LEN              0x08
#define HAND_SHAKE_INP              "pIpEtEsT"
#define HAND_SHAKE_OUT              "PiPeTeSt"
#define HAND_SHAKE_ERROR            101
#define PROGRAM_ERROR               999

CHAR    achPipeName [MAX_PIPE_NAME_LEN] ;
HPIPE   hpPipe ;
CHAR    chToken ;

USHORT BadArgs ( USHORT usNumArgs, PCHAR apchArgs [] ) ;
APIRET ConnFromClient ( VOID ) ;
APIRET ConnToServer ( VOID ) ;
APIRET SendToServer ( ULONG ulHiLoCh ) ;
APIRET RecvFromClient ( PULONG pulHiLoCh, PULONG pulBytesDone ) ;
VOID APIENTRY CleanUp ( ULONG  ulTermCode );/* ExitList routines must be declared with
VOID APIENTRY*/
```

**CLNTSRVR.MAK**

```
ALL:    CLIENT.EXE SERVER.EXE

CLIENT.EXE:                         CLIENT.OBJ
        LINK386  @<<
CLIENT
CLIENT
CLIENT
OS2386
CLIENT
<<

SERVER.EXE:                         SERVER.OBJ
        LINK386  @<<
SERVER
SERVER
SERVER
OS2386
SERVER
<<


CLIENT.OBJ:                         CLIENT.C
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ -Q CLIENT.C

SERVER.OBJ:                         SERVER.C
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ -Q SERVER.C
```

When the *Client* is started, the initialization call is made to *ConnToServer()*. The client application must perform a *DosOpen()* first in order to obtain a pipe handle. Once the pipe handle is obtained, the application can freely read from the pipe and write to the pipe. In this case, the first write/read pair is used for primitive handshaking communication.

The most interesting set of parameters for the *DosOpen()* call on the client side is the *ulOpenFlag,* which contains the value OPEN_ACTION_OPEN_IF_EXISTS, and the *ulOpenMode,* which contains the

OPEN_FLAGS_WRITE_THROUGH | OPEN_FLAGS_FAIL_ON_ERROR | OPEN_FLAGS_RANDOM | OPEN_SHARE_DENYNONE | OPEN_ACCESS_READWRITE value.

Next, the *while* loop is entered. It can be stopped only if an API error is encountered, or if the user presses the F3 function key at the *Client* window. The buffer that is being transmitted from the *Client* to the *Server* represents the character received from the keyboard buffer used by the *Client* application. A double word is used to allow proper character translation for the F1–F12 function keys and some other extended keyboard keys. (The function key keystroke generates two characters; the first is always a 0x00, followed by the 0xYY, where YY is a unique function key identifier.)

The remote pipe connection from the *Client* to the *Server* is achieved by starting the CLIENT.EXE with the following command-line syntax:

```
CLIENT [MYSERVER]
```

where *MYSERVER* is the remote Server machine name. (The NetBIOS machine name for IBM OS/2 LAN Server is found in the IBMLAN.INI file). The pipe names that are created by the *Client* have the following format:

| | |
|---|---|
| local named pipe name : | \PIPE\MYPIPE |
| remote named pipe name : | \\MYSRVR\PIPE\MYPIPE |

The functionality that this example application provides is the same in both remote and local connectivity modes. As a matter of fact, neither the *Client* nor the *Server* differentiates between the remote and local case; only the pipe name is significant. This is the subtle beauty of the named pipes IPC!

The main reason for choosing pipes as an IPC method is ease of implementation, but it is not the best choice for all cases. Pipes are useful only when a process has to send a lot of information to or receive information from another process. Even though it is possible to allow pipe connections with multiple processes, connect and disconnect algorithms must always be implemented for such situations. The remote connection advantage of named pipes sometimes outweighs the complexity of connect-disconnect algorithms. Since it is not possible under OS/2 to communicate remotely with queues or remote shared memory, pipes sometimes become not only the best but the only IPC choice.

**Gotcha!**
It is not unusual for an application to receive a return value of ERROR_TOO_MANY_HANDLES when attempting to open additional pipes. The system initially allows 20 file handles per process; once the limit is reached, the above error will appear. To prevent this from happening, the *DosSetMaxFH(ULONG ulNumberHandles)* call must be issued, where *ulNumberHandles* is the new maximum number of handles allowed to be open. This call will be successful if system resources have not been exhausted. It is a good idea to issue this call only when needed, since additional file handles consume system resources that may be used elsewhere in the system.

## DOS-OS/2 Client-Server Connection

To make the pipe connectivity example complete, a DOS-named pipe client must be discussed. The DOS based, D_CLIENT.EXE, is only slightly different from its big brother, the OS/2 based CLIENT.EXE.

There are no logical diffrences between the two; the difference lies in the APIs. The *DosOpen()/DosRead()/DosWrite()* OS/2 calls are replaced with *open()/read()/write()* DOS calls.

**D_CLIENT.C**

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <fcntl.h>
#include "dcommon.h"
int main(unsigned short usNumArgs,char *apchArgs[])

{
   unsigned short   *usHiCh;
   unsigned short   *usLoCh;
   unsigned long    ulHiLoCh;
   unsigned short   arReturn;

   achPipeName[0] = 0;
   chToken = 0;
   ulHiLoCh = 0;
   usLoCh = (unsigned short *)&ulHiLoCh;
   usHiCh = usLoCh+1;
   chToken = CLIENT_MODE;

/*   ANSI.SYS must be loaded for the COLOR string to work      */
/*   printf (CLIENT_COLOR);                                    */

   printf("\n\n Hit F3 to exit client program. \n\n");
   printf(" Starting the program in Client Mode...\n\n");

   if (usNumArgs == REMOTE_PIPE)
   {
      sprintf(achPipeName,
              "\\\\%s",
              apchArgs[1]);
   }

   strcat(achPipeName,
          DEFAULT_PIPE_NAME);
   printf(" Connecting to pipe : %s\n\n",
          achPipeName);

   arReturn = ConnToServer();
   if (arReturn != EOF)
   {
      printf(" You can start typing in this CLIENT window\n");
      printf(
       " and watch for your keystrokes in the SERVER window\n\n")
         ;
   }

   while ((arReturn != EOF) && (chToken != DISCON_MODE))
   {

      *usHiCh = getch();
      if ((*usHiCh == FUNC_KEYS_CHAR) || (*usHiCh ==
         EXTD_KEYS_CHAR))
      {
         *usLoCh = getch();
      }
      else
      {
         *usLoCh = *usHiCh;
      }                                  /* endif                */
```

```
      arReturn = SendToServer(ulHiLoCh);

      if (ulHiLoCh == TOKEN_F3_DISCON)
      {
         chToken = DISCON_MODE;
         break;
      }                                    /* endif                  */
      if (*usLoCh == *usHiCh)
         putch(*usHiCh);
      else
         putch('*');

      if (*usLoCh == RETURN_CHAR)
      {
         putch(LINE_FEED_CHAR);
      }                                    /* endif                  */
   }                                       /* endwhile               */
   close(hpPipe);

/*   ANSI.SYS must be loaded for the COLOR string to work            */
/*   printf (NORMAL_COLOR);                                          */

   return  arReturn;
}

unsigned short ConnToServer(void)
{
   char              achInitBuf[HAND_SHAKE_LEN+1];
   unsigned long     ulOpenFlag;
   unsigned long     ulOpenMode;
   unsigned long     ulActionTaken;
   int               arReturn = 0;
   unsigned long     ulBytesDone;

   memset(achInitBuf,
          0,
          sizeof(achInitBuf));

   ulOpenFlag = DEFAULT_OPEN_FLAG;
   ulOpenMode = DEFAULT_OPEN_MODE;

   arReturn = hpPipe = open(achPipeName,
                            O_RDWR|O_BINARY);

   if (arReturn != EOF)
   {
      arReturn = write(hpPipe,
                       HAND_SHAKE_INP,
                       strlen(HAND_SHAKE_INP));

      if (arReturn != EOF)
      {

         arReturn = read(hpPipe,
                         achInitBuf,
                         HAND_SHAKE_LEN);

         if (strcmp(achInitBuf,
                    HAND_SHAKE_OUT))
         {
            arReturn = HAND_SHAKE_ERROR;
         }                                 /* endif                  */
      }                                    /* endif                  */
   }                                       /* endif                  */
   if (arReturn == EOF)
   {
      printf("\n  The Pipe Open / Connection API "
         "returned rc = %02d\n",
             arReturn);
```

```
      printf("\n  Make sure the Server is running.\n\n");
   }                                        /* endif               */
   return  arReturn;
}

unsigned short SendToServer(unsigned long ulHiLoCh)
{
   unsigned long    ulBytesDone;

   return (write(hpPipe,
                 &ulHiLoCh,
                 sizeof(ulHiLoCh)));
}
```

## D_CLIENT.MAK

```
ALL:    D_CLIENT.EXE

D_CLIENT.EXE:                           D_CLIENT.OBJ
        LINK  /NOD @<<
D_CLIENT
D_CLIENT
D_CLIENT
LLIBCER

<<

D_CLIENT.OBJ:         D_CLIENT.C D_CLIENT.MAK
        cl -c -AL D_CLIENT.C
```

## D_CLIENT.DEF

```
NAME D_CLIENT WINDOWCOMPAT
DESCRIPTION 'CLIENT example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

## DCOMMON.H

```
/*     this nibble applies if file already exists                xxxx */

#define OPEN_ACTION_FAIL_IF_EXISTS     0x0000  /* ---- ---- ---- 0000 */
#define OPEN_ACTION_OPEN_IF_EXISTS     0x0001  /* ---- ---- ---- 0001 */
#define OPEN_ACTION_REPLACE_IF_EXISTS  0x0002  /* ---- ---- ---- 0010 */

/*     this nibble applies if file does not exist           xxxx      */

#define OPEN_ACTION_FAIL_IF_NEW        0x0000  /* ---- ---- 0000 ---- */
#define OPEN_ACTION_CREATE_IF_NEW      0x0010  /* ---- ---- 0001 ---- */

/* DosOpen/DosSetFHandState flags */

#define OPEN_ACCESS_READONLY         0x0000  /* ---- ---- ---- -000 */
#define OPEN_ACCESS_WRITEONLY        0x0001  /* ---- ---- ---- -001 */
#define OPEN_ACCESS_READWRITE        0x0002  /* ---- ---- ---- -010 */
#define OPEN_SHARE_DENYREADWRITE     0x0010  /* ---- ---- -001 ---- */
#define OPEN_SHARE_DENYWRITE         0x0020  /* ---- ---- -010 ---- */
#define OPEN_SHARE_DENYREAD          0x0030  /* ---- ---- -011 ---- */
#define OPEN_SHARE_DENYNONE          0x0040  /* ---- ---- -100 ---- */
#define OPEN_FLAGS_NOINHERIT         0x0080  /* ---- ---- 1--- ---- */
#define OPEN_FLAGS_NO_LOCALITY       0x0000  /* ---- -000 ---- ---- */
#define OPEN_FLAGS_SEQUENTIAL        0x0100  /* ---- -001 ---- ---- */
```

```
#define OPEN_FLAGS_RANDOM           0x0200  /* ---- -010 ---- ---- */
#define OPEN_FLAGS_RANDOMSEQUENTIAL 0x0300  /* ---- -011 ---- ---- */
#define OPEN_FLAGS_NO_CACHE         0x1000  /* ---1 ---- ---- ---- */
#define OPEN_FLAGS_FAIL_ON_ERROR    0x2000  /* --1- ---- ---- ---- */
#define OPEN_FLAGS_WRITE_THROUGH    0x4000  /* -1-- ---- ---- ---- */
#define OPEN_FLAGS_DASD             0x8000  /* 1--- ---- ---- ---- */


#define SERVER_MODE             1
#define CLIENT_MODE             2
#define SERVER_COLOR            "\n•[0;32;40m"
#define CLIENT_COLOR            "\n•[0;36;40m"
#define NORMAL_COLOR            "\n•[0;37;40m"
#define REMOTE_PIPE             2
#define DISCON_MODE             3
#define BAD_INPUT_ARGS          99
#define MAX_PIPE_NAME_LEN       80
#define MAX_SERV_NAME_LEN       8
#define DEFAULT_PIPE_NAME       "\\PIPE\\MYPIPE"
#define DEFAULT_MAKE_MODE       NP_ACCESS_DUPLEX
#define DEFAULT_PIPE_MODE       NP_WMESG | NP_RMESG | 0x01
#define DEFAULT_OPEN_FLAG       OPEN_ACTION_OPEN_IF_EXISTS
#define DEFAULT_OPEN_MODE       OPEN_FLAGS_WRITE_THROUGH | \
                                OPEN_FLAGS_FAIL_ON_ERROR | \
                                OPEN_FLAGS_RANDOM |        \
                                OPEN_SHARE_DENYNONE |      \
                                OPEN_ACCESS_READWRITE
#define DEFAULT_OUTB_SIZE       0x1000
#define DEFAULT_INPB_SIZE       0x1000
#define DEFAULT_TIME_OUTV       20000L
#define TOKEN_F3_DISCON         0x0000003DL
#define RETURN_CHAR             0x0D
#define LINE_FEED_CHAR          0x0A
#define FUNC_KEYS_CHAR          0x00
#define EXTD_KEYS_CHAR          0xE0
#define HAND_SHAKE_LEN          0x08
#define HAND_SHAKE_INP          "pIpEtEsT"
#define HAND_SHAKE_OUT          "PiPeTeSt"
#define HAND_SHAKE_ERROR        101
#define PROGRAM_ERROR           999

char           achPipeName [MAX_PIPE_NAME_LEN] ;
unsigned short hpPipe ;
char           chToken ;

unsigned short ConnToClient   ( void ) ;
unsigned short ConnToServer   ( void ) ;
unsigned short SendToServer   ( unsigned long ulHiLoCh ) ;
unsigned short RecvFromClient ( unsigned long * pulHiLoCh, unsigned long * pulBytesDone );
```

# An OS/2 QUEUE Client-Server Example

The next example pair is QSERVER.C and QCLIENT.C. In this example, the communication process is a little bit more complex than the one in the named pipe illustration. Here the point is to show how several different processes can communicate with one central process. The functionality is similar to the named pipe example, but with one key difference: The queue *Server* process does not send anything to the queue *Client* processes. In fact, only the queue *Client* process can send information to the queue *Server.* However, this does not mean that the queue *Server* cannot issue a *DosWriteQueue()* call itself; it is just not part of this example. It is left to the reader to implement this additional functionality. By using the QSERVER.C as a prototype template, the *WriteToQue* function call can enhance the QSERVER.C example program to issue *DosWriteQueue* calls. The QSERVER.C-QCLIENT.C example makes use of both the OS/2 queue APIs and named shared memory segments.

The concept of an OS/2 queue is somewhat simple. It is, in fact, an ordered set of elements. The elements are 32-bit values that are passed from the *Client* to the *Server* of the queue. The *Server* of the queue is the process that created the queue by issuing the *DosCreateQueue()* API call.

```
APIRET DosCreateQueue( PHQUEUE phq, ULONG ulPriority, PSZ pszName )
```

*phq* is a pointer to the queue handle of the queue that is being created. *ulPriority* is a set of two flags OR'ed together. The first flag can have the values listed in Table 5.2.

Table 5.2. Values of Low Byte of *ulPriority*.

| Value | Description |
|---|---|
| QUE_FIFO | FIFO queue |
| QUE_LIFO | LIFO queue |
| QUE_PRIORITY | Priority queue |

The second flag can have the values listed in Table 5.3.

Table 5.3. Values of High Byte of *ulPriority*.

| Value | Description |
|---|---|
| QUE_NOCONVERT_ADDRESS | Does not convert addresses of 16-bit elements that are placed in the queue |
| QUE_CONVERT_ADDRESS | Converts addresses of 16-bit elements to 32-bit elements |

The last parameter is a pointer to the ASCII name of the queue.

Only the *Server* of the queue can read from the queue. When the queue is read, one element is removed from it. The *Server* and the *Client* can both issue calls to write, query, and close the queue. However, only the *Server* can issue calls to create, read, peek, and purge the queue. The *Client* must issue a *DosOpenQueue* call prior to attempting to write elements to the queue or to query the queue elements.

```
APIRET DosOpenQueue( PPID ppid, PHQUEUE phq, PSZ pszName )
```

*ppid* is a pointer to the process ID of the queue's server process. *phq* is a pointer to the write handle of the queue. *pszName* is the ASCII name of the queue to be opened.

The queue elements can be prioritized and processed in particular order. The order depends on the *ulQueueFlags* value used when creating the queue. This value cannot be changed once the queue has been created.

Specifying a priority will cause the *DosReadQueue* API to read the queue elements in descending priority order. Priority 15 is the highest, and 0 is the lowest. FIFO order will be used for the elements with equal priority. The elements of the queue can be used to pass data to the server directly or indirectly. The indirection comes from using pointers to shared memory. When pointers are used, the shared memory can be of two types: named shared memory and unnamed shared memory. Related processes generally use named shared memory, while the rest use unnamed shared memory. In this example, the named shared memory method is implemented. OS/2 queues do not perform any data copying. They only pass pointers. They leave the rest of the work for the programmer.

```
APIRET DosReadQueue( HQUEUE hQue, PREQUESTDATA pData, PULONG pcbData,
                     PPVOID ppBuf, ULONG ulElement, BOOL32 bWait,
                     PBYTE pbPriority, HEV hevSem )
```

*hQue* is a handle of the queue to be read from. *pData* is a pointer to a REQUESTDATA structure that returns a PID and an event code. *pcbData* is an output parameter that specifices the length of the data to be removed. *ppBuf* is an output parameter that is a pointer to the element being removed from the queue. *ulElement* is an indicator that can be either 0, meaning remove the first element from the queue, or a value returned by *DosPeekQueue*. Table 5.4 lists the values for *bWait*.

Table 5.4. Values for *bWait*

| Value | Description |
|---|---|
| DCWW_WAIT | The thread will wait for an element to be added to the queue |
| DCWW_NOWAIT | Return immediately with ERROR_QUE_EMPTY if no data is available |

*pbPriority* is an output parameter that indicates the priority of the element being read. *hevSem* is a handle of an event semaphore that will be posted when data is added to the queue, and DCWW_NOWAIT is specified.

The OS/2 QUEUE *Client-Server* example is best illustrated by starting several OS/2 window sessions from the desktop and making all of them visible to the user at the same time. The queue *Server* process must be started first. Once the queue is created and the queue *Server* is started, the queue *Clients* can use the queue to pass various information to the queue *Server*. In this case the information that is passed is the keystrokes the user enters from each one of the *Client* processes. Figure 5.1 illustrates this procedure.

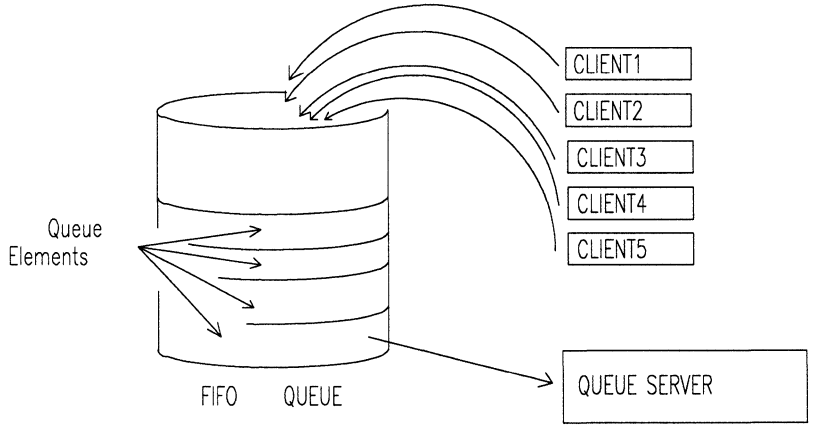



Figure 5.1 Diagram of a queue.

Each one of the queue *Clients* will send keystroke characters to the queue *Server* via FIFO queue. Once the characters are received by the queue *Server*, they will be displayed in color depending on the *Client* that sent them. Table 5.5 describes the queue client text colors.

Table 5.5 Queue Client Text Colors

| Number | Color |
|---|---|
| Client 1 | Red |
| Client 2 | Green |
| Client 3 | Yellow |
| Client 4 | Blue |
| Client 5 | Magenta |

The QSERVER.EXE allows only up to five active QCLIENT.EXE connections at any one time. Once the maximum number of clients has been reached, entering QCLIENT.EXE followed by a carriage return from the command line will produce a program error message describing the maximum number of clients.

The complete listing of QSERVER.C follows.

**QSERVER.C**

```
#define  INCL_DOSQUEUES
#define  INCL_DOSMEMMGR
#define  INCL_DOSPROCESS
#define  INCL_DOSERRORS
#define  INCL_DOSSEMAPHORES
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include "qcommon.h"
APIRET InitServerQueEnv(VOID);

APIRET ReadFromQue(PULONG pulHiLoCh);
APIRET ToScreen(VOID *buffer,ULONG length);

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{
   USHORT           *usHiCh;
   USHORT           *usLoCh;
   ULONG            ulHiLoCh;
   APIRET           arReturn = 0;

   arReturn = DosExitList(EXLST_ADD,
                          CleanUp);

   ulHiLoCh = 0;
   usLoCh = (USHORT *)&ulHiLoCh;
   usHiCh = usLoCh+1;

   printf(SERVER_COLOR);
   printf(" Server process is creating");
   printf(" and initializing the Queue...\n");
   arReturn = InitServerQueEnv();
   if (!arReturn)
      printf("\n Press Ctrl-C, or Ctrl-Break to exit\n\n");

   while (!arReturn)
   {
      arReturn = ReadFromQue(&ulHiLoCh);

      if (!arReturn)
      {

         if (*usLoCh == *usHiCh)
            putch(*usHiCh);
```

```
         else
            if (ulHiLoCh == TOKEN_F3_DISCON)
            {
               ToScreen(WHITE_COLOR,
                        strlen(WHITE_COLOR));
               ToScreen("\n\r client exited\n\n\r",
                        strlen("\n\r client exited\n\n\r"));
            }
            else
               putch('*');
         if (*usLoCh == RETURN_CHAR)
            putch(LINE_FEED_CHAR);
      }
      else
         if (arReturn == ERROR_QUE_EMPTY)

            arReturn = (SHORT)NULL;
   }

   arReturn = DosCloseQueue(hqQueue);
   printf(WHITE_COLOR);
   return  arReturn;
}

APIRET InitServerQueEnv(VOID)
{
   APIRET           arReturn;
   SHORT            sIndex;

   arReturn = DosAllocSharedMem((PVOID)&pmqsClient,
                                DEFAULT_SEG_NAME,
                                DEFAULT_PAGE_SIZE,
                                DEFAULT_SEG_FLAG);

   if (!arReturn)
   {
      arReturn = DosCreateQueue(&hqQueue,
                                DEFAULT_QUE_FLAG,
                                DEFAULT_QUE_NAME);

      if (!arReturn)
      {
         printf("\n Queue created successfully \n");
         for (sIndex = 0; sIndex < MAX_CLIENTS; sIndex++)
         {
            pmqsClient[sIndex].szColor[0] = (BYTE)NULL;
            pmqsClient[sIndex].ulPid = (PID)NULL;
         }                                /* endfor                */
         arReturn = DosCreateEventSem(DEFAULT_SEM_NAME,
                                      &hsmSem,
                                      ULONG_NULL,
                                      TRUE);
         if (arReturn)
            printf("\n DosCreateEventSem returned ""%02d\n",
                   arReturn);
      }
      else
      {
         printf("\n  DosCreateQueue API returned ""%02d\n",
                arReturn);
      }                                   /* endif                 */
   }
   else
   {
      printf(" \n  Could not allocate "
         "Shared Memory ( %02d ) \n",
             arReturn);
   }                                      /* endif                 */
   return  arReturn;
```

```
}

APIRET ReadFromQue(PULONG pulHiLoCh)
{
   APIRET           arReturn;
   REQUESTDATA      rdRequest;
   ULONG            ulSzData;
   BYTE             bPriority;

   arReturn = DosReadQueue(hqQueue,
                           &rdRequest,
                           &ulSzData,
                           &pvData,
                           0,
                           DCWW_NOWAIT,
                           &bPriority,
                           hsmSem);

   if (!arReturn)
   {
      pmqsClient[rdRequest.ulData].ulPid = rdRequest.pid;
      *pulHiLoCh = ulSzData;

      ToScreen(pmqsClient[rdRequest.ulData].szColor,
               strlen(pmqsClient[rdRequest.ulData].szColor));

   }                                    /* endif                */
   return  arReturn;
}

APIRET ToScreen(VOID *buffer,ULONG length)
{
#define  MY_STDOUT     1
   ULONG            ulBytesDone;

   return (DosWrite(MY_STDOUT,
                    buffer,
                    length,
                    &ulBytesDone));

}

VOID APIENTRY CleanUp(ULONG ulTermCode)
{

   DosCloseQueue(hqQueue);

   ToScreen(WHITE_COLOR,
            strlen(WHITE_COLOR));

   DosExitList(EXLST_EXIT,
               0);
}
```

**QSERVER.DEF**

```
NAME QSERVER WINDOWCOMPAT
DESCRIPTION 'Queue example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

Now that the intended operation of the OS/2 QUEUE *Client-Server* has been described, the implementation itself can be discussed in greater detail.

During the initialization *Server* uses the *InitServerQueEnv()* first to allocate the named shared memory segment, next to create the queue, and last to create the queue event semaphore.

The named shared memory segment is used as a common communications area for all of the *Clients* and the *Server*. The shared named memory segment later will contain client-specific information: the *Client* process ID and the client text color ANSI escape sequence. The memory map in Figure 5.2 shows the way the shared named memory segment is used.

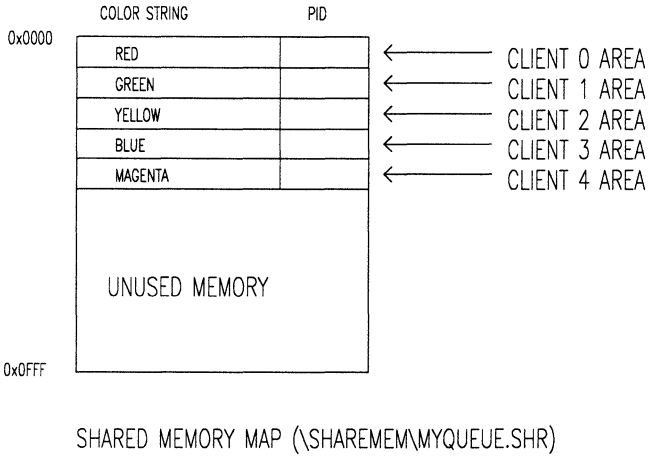



**Figure 5.2 Shared memory map.**

A client area is dedicated to each one of the queue *Clients* and contains the entire *MYQUESTRUC* structure. After the shared memory is allocated, the queue *Server* creates the queue and initializes the named shared segment to nulls. The last API that is called by the initialization routine is *DosCreateEventSem.* Even though the semaphore that is created will not be used as a semaphore during this application, its handle is required later for the *DosReadQueue.* The reason it is required in this case is because the queue is read in nonblocking mode, and the API requires a semaphore handle in that case. Choosing to read the queue in nonblocking fashion allows the queue *Server* main thread to perform other functions while waiting for the new queue elements.

```
APIRET DosCreateEventSem(  PSZ pszName, PHEV phev,
                           ULONG flAttr, BOOL32 fState )
```

*pszName* is a pointer to the ASCII name of the semaphore, *phev* is an output parameter that is a pointer to the semaphore handle. *flAttr* is either DC_SEM_SHARED to indicate the semaphore is shared, or 0. All named semaphores are shared, so if *pszName* is not null, this argument is unused.

*fState* can be either TRUE, meaning the semaphore is initially "posted" or FALSE, meaning the semaphore is initially "set."

In the initialization of the queue *Client* environment, the *InitClientQueEnv()* function call attempts to obtain the named shared memory handle. Once the handle is returned, the queue *Client* begins to scan the client areas, checking for the valid color string. The moment the *Client* finds an unused color string area, it assumes it is free and copies its color attribute there. It also saves the unique position identification

assumes it is free and copies its color attribute there. It also saves the unique position identification number in the global *sIndex* variable. If the *Client* determines that five other *Clients* are already active, it will display an error message and exit. On the other hand, if the *sIndex* value is acceptable (less than maximum number of *Clients*), the *Client* will issue the *DosOpenQueue()* API call, thus completing the initialization by connecting to the queue.

**QCLIENT.C**

```
#define  INCL_DOSQUEUES
#define  INCL_DOSMEMMGR
#define  INCL_DOSPROCESS
#define  INCL_DOSERRORS
#define  INCL_DOSSEMAPHORES
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include "qcommon.h"
APIRET InitClientQueEnv(VOID);

APIRET WriteToQue(ULONG ulHiLoCh);

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{
   USHORT           *usHiCh;
   USHORT           *usLoCh;
   ULONG            ulHiLoCh;
   APIRET           arReturn;

   ulHiLoCh = 0;
   usLoCh = (USHORT *)&ulHiLoCh;
   usHiCh = usLoCh+1;
   chToken = CLIENT_MODE;

   strcpy(aachColors[0],
          CLIENT0_COLOR);
   strcpy(aachColors[1],
          CLIENT1_COLOR);
   strcpy(aachColors[2],
          CLIENT2_COLOR);
   strcpy(aachColors[3],
          CLIENT3_COLOR);
   strcpy(aachColors[4],
          CLIENT4_COLOR);

   printf(WHITE_COLOR);
   printf(" Client process is initializing");
   printf(" and connecting to the Queue...\n");
   arReturn = InitClientQueEnv();

   if (!arReturn)
      printf("\n Press F3 to exit\n\n");

   while (!arReturn && (chToken != DISCON_MODE))
   {
      *usHiCh = getch();
      if ((*usHiCh == FUNC_KEYS_CHAR) || (*usHiCh ==
         EXTD_KEYS_CHAR))
      {
         *usLoCh = getch();
      }
      else
      {
         *usLoCh = *usHiCh;
```

```
      }                                    /* endif                  */
      arReturn = WriteToQue(ulHiLoCh);

      if (ulHiLoCh == TOKEN_F3_DISCON)
      {
         chToken = DISCON_MODE;
         pmqsClient[usClientIndex].szColor[0] = '\0';
         pmqsClient[usClientIndex].ulPid = 0;
         break;
      }                                    /* endif                  */
      if (*usLoCh == *usHiCh)
         putch(*usHiCh);
      else
         putch('*');

      if (*usLoCh == RETURN_CHAR)
      {
         putch(LINE_FEED_CHAR);
      }                                    /* endif                  */
   }

   if (arReturn == 0)
   {
      arReturn = DosCloseQueue(hqQueue);
   }                                       /* endif                  */
   printf("\n•[0;37;40m");
   return  arReturn;
}

APIRET InitClientQueEnv(VOID)
{
   APIRET           arReturn;
   SHORT            sIndex;
   PID              pidOwner;

   arReturn = DosGetNamedSharedMem((PVOID)&pmqsClient,
                                   DEFAULT_SEG_NAME,
                                   PAG_WRITE|PAG_READ);

   if (!arReturn)
   {
      for (sIndex = 0; sIndex <= MAX_CLIENTS; sIndex++)
      {
         if ((pmqsClient[sIndex].szColor[0] == 0) && (sIndex <
            MAX_CLIENTS))
         {
            strcpy(pmqsClient[sIndex].szColor,
                   aachColors[sIndex]);
            usClientIndex = sIndex;
            break;
         }                                 /* endif                  */
      }                                    /* endfor                 */
      if (sIndex > MAX_CLIENTS)
      {
         arReturn = PROGRAM_ERROR;
         printf("\n\n  Maximum number of clients is FIVE !\n");
      }                                    /* endif                  */
      if (!arReturn)
      {
         arReturn = DosOpenQueue(&pidOwner,
                                 &hqQueue,
                                 DEFAULT_QUE_NAME);
         if (!arReturn)
         {
            printf(" %s",
                   aachColors[usClientIndex]);
            printf("\n Client #%d has connected to the Queue\n",
                   usClientIndex);
         }                                 /* endif                  */
```

```
      }                                    /* endif                */
   }                                       /* endif                */
   return  arReturn;
}

APIRET WriteToQue(ULONG ulHiLoCh)
{
   return  DosWriteQueue(hqQueue,

                         (ULONG)usClientIndex,
                         ulHiLoCh,
                         pvData,
                         ULONG_NULL);
}
```

## QCLIENT.DEF

```
NAME QCLIENT WINDOWCOMPAT
DESCRIPTION 'Queue example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

## QCOMMON.H

```
#define MAX_INPUT_ARGS          2
#define MAX_CLIENTS             5
#define SERVER_MODE             1
#define CLIENT_MODE             2
#define DISCON_MODE             3
#define BAD_INPUT_ARGS          99
#define DEFAULT_QUE_NAME        "\\QUEUES\\MYQUEUE"
#define DEFAULT_SEM_NAME        "\\SEM32\\EVENTQUE"
#define DEFAULT_PAGE_SIZE       4096
#define DEFAULT_QUE_FLAG        QUE_FIFO | QUE_CONVERT_ADDRESS
#define DEFAULT_SEG_NAME        "\\SHAREMEM\\MYQUEUE.SHR"
#define DEFAULT_SEG_FLAG        PAG_WRITE | PAG_COMMIT
#define TOKEN_F2_SWITCH         0x0000003CL
#define TOKEN_F3_DISCON         0x0000003DL
#define RETURN_CHAR             0x0D
#define LINE_FEED_CHAR          0x0A
#define FUNC_KEYS_CHAR          0x00
#define EXTD_KEYS_CHAR          0xE0
#define CLEAR_HI_WORD           0x0000FFFFL
#define ULONG_NULL              0L
#define PROGRAM_ERROR           999

#define MAX_COLOR_LEN           14

char        SERVER_COLOR    [MAX_COLOR_LEN]  =  {10,13,27,91,48,59,51,50,59,52,48,109,0};/*
"\n•[0;32;40m"*/
char         WHITE_COLOR    [MAX_COLOR_LEN]  =  {10,13,27,91,48,59,51,55,59,52,48,109,0};/*
"\n•[0;37;40m"*/
char           CLIENT0_COLOR   [MAX_COLOR_LEN]   =   {27,91,48,59,51,49,59,52,55,109,0};/*
"•[0;31;40m"*/
char           CLIENT1_COLOR   [MAX_COLOR_LEN]   =   {27,91,48,59,51,50,59,52,55,109,0};/*
"•[0;32;40m"*/
char           CLIENT2_COLOR   [MAX_COLOR_LEN]   =   {27,91,48,59,51,51,59,52,55,109,0};/*
"•[0;33;40m"*/
char           CLIENT3_COLOR   [MAX_COLOR_LEN]   =   {27,91,48,59,51,54,59,52,55,109,0};/*
"•[0;34;40m"*/
char           CLIENT4_COLOR   [MAX_COLOR_LEN]   =   {27,91,48,59,51,53,59,52,55,109,0};/*
"•[0;35;40m"*/
```

```
typedef struct  _MYQUEUESTRUC {
   BYTE szColor [MAX_COLOR_LEN] ;
   PID ulPid ;
} MYQUEUESTRUC, * PMYQUEUESTRUC ;

HQUEUE         hqQueue ;
USHORT         usClientIndex = MAX_CLIENTS ;
PVOID          pvData ;
HEV            hsmSem ;
PMYQUEUESTRUC pmqsClient ;
CHAR           chToken = 0 ;
CHAR           aachColors [MAX_CLIENTS] [MAX_COLOR_LEN] ;
VOID APIENTRY CleanUp ( ULONG  ulTermCode );/* ExitList routines must be declared with
VOID APIENTRY*/
```

**Q_CS.MAK**

```
ALL:    QCLIENT.EXE QSERVER.EXE

qserver.EXE:           qserver.OBJ
        LINK386 @<<
qserver
qserver
qserver
OS2386
qserver
<<

qserver.OBJ:                        qserver.C q_CS.MAK
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ -Q qserver.C

qclient.EXE:                        qclient.OBJ
        LINK386 @<<
qclient
qclient
qclient
OS2386
qclient
<<

qclient.OBJ:                        qclient.C q_CS.MAK
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ -Q qclient.C
```

First, the queue server attempts to read the queue; if any elements are present, they are decoded and displayed in their corresponding color; otherwise the *Server* loops to check for the next queue element. The ERROR_QUE_EMPTY is ignored and reset to 0. It is normal for the *Server* to receive this particular error since it is possible for the queue to have no messages from any of the *Clients*.

Readers may wonder why the queue is read continuously in nonblocking mode when it can be read in blocking mode, which will assure a returned queue element prior to completing the *DosReadQueue* call. The answer is simple. If the *DosReadQueue* API was implemented with the blocking flag set to true, it would be difficult for the main thread to do anything other than wait. An additional thread would have to implemented to handle any other type of work. It is also possible to implement a separate thread that waits on the queue event semaphore and displays the characters only when the semaphore was posted Because either method would be more complex, we chose the current implementation for this sample program. The point here is to show the differences between the OS/2 queues and the OS/2 named pipes.

The *Client* does nothing more than read a keystroke character and write that character to the queue by issuing a *WriteToQue()* function call, which in turn calls the *DosWriteQueue()* API.

```
APIRET DosWriteQueue( HQUEUE hQue, ULONG ulRequest,
                      ULONG cbData, PVOID pbData, ULONG ulPriority )
```

*hQue* is a handle of the queue to which data is to be written. *ulRequest* is a user-defined value passed with *DosPeekQueue*. *cbData* is length of the data that is being written. *pbData* is a pointer to the data. *ulPriority* is a priority of the data being added to the queue. Any value between 0 and 15 is accepted. A value of 15 indicates the element is added to the top of the queue, and a value of 0 indicates the element is the last element in the queue.

This example shows that the OS/2 queues are somewhat cumbersome to implement; however, they are very useful when several processes have to talk to a single process, even if the processes are unrelated.

Note: The *InitClientQueEnv* function has a potential timing problem. If multiple clients decide to initialize concurrently, a race condition will ensue. To avoid a potential problem, a Mutex semaphore should be installed to protect the access to the shared memory. The implementation is left as an exercise for the reader.

## An OS/2 Semaphore vs. Flag Variable Example

There are three different types of semaphores: Event, Mutex, and MuxWait. Event semaphores are used when a thread or a process needs to notify other threads or processes that some event has occurred. Mutex semaphores enable multiple threads or processes to coordinate or serialize their access to some shared resource. MuxWait semaphores, on the other hand, enable threads or processes to wait for multiple events to occur.

With this brief introduction, here is the last IPC example pair: STHREAD.C and FTHREAD.C. This case uses the concept of semaphores for task or event synchronization, also known as signaling. If a process is waiting for a resource to become available, such as a file or a port access right, and the resource is being used by another process, the current task must wait. In the earlier DOS operating systems the synchronization was accomplished primitively through the use of flags. The developer would set a flag, then wait for the flag to be cleared, thus signaling that the resource was free to be used. Since only one process could execute at a time under DOS, this was an acceptable form of pseudo interprocess communication. Under OS/2, however, it is not a good idea to use flags to perform the equivalent semaphore functions. An example of this bad flag synchronization processing is evident in FTHREAD.C, which employs the following construct:

```
while (FlagBusy);    /* Wait for flag to clear */
```

If a task requires this type of processing, a semaphore should be used. The STHREAD.C example demonstrates the difference in the number of machine cycles that are spent waiting for a semaphore to clear as opposed to waiting for a flag to clear. The STHREAD.EXE creates several threads and then decides to wait on a semaphore. The default number of threads is 10, but that number can be changed by providing an input argument to the STHREAD.EXE program. While this wait is in process, the user is free to type characters at the keyboard, which will be echoed to the console immediately. In contrast, the FTHREAD.EXE uses the same logic but employs a flag variable to perform the wait inside the threads, which dramatically increases CPU usage, and the keystrokes will appear greatly delayed. The FTHREAD.EXE also can accept an input argument specifying the number of threads to be created to wait on the same flag variable. Even with as little as 30 threads, the difference between waiting on a flag variable and waiting on a semaphore is dramatic.

**FTHREAD.C**

```
#define  INCL_DOSPROCESS
#define  INCL_DOSSEMAPHORES
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#define  DEFAULT_THREAD_FLAGS 0
#define  DEFAULT_THREAD_STACK 0x4000
#define  MY_BEGIN_SEMAPHORE "\\SEM32\\BEGIN"
#define  MAX_SEM_WAIT  - 1L
#define  DEFAULT_NUM_THREADS 10
#define  MAX_NUM_THREADS 255
USHORT usWaitOnFlag = TRUE;

VOID APIENTRY MyThreadOne(void);

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{
   USHORT           usNumThreads;
   TID              tidThread;
   INT              iCharRead;
   USHORT           usReturn = 0;

   printf("\n FTHREAD.EXE demonstrates very poor processor");
   printf("\n time management by allowing a user to start a ");
   printf("\n number of threads (1-255) and have all of them");
   printf("\n wait on one global flag, while allowing keystrokes"
      );
   printf("\n to be entered at the keyboard.");
   printf("\n (see STHREAD.EXE, for speed comparison)");
   printf("\n\n      FTHREAD.EXE [X], where X is 1-255\n\n");
   printf("\n\n lower case 'x' exits ...\n\n");

   if (usNumArgs > 1)
   {

     //-------------------------------------------------------
      // Insure that usNumThreads is in the range 1<<x<<MAX
     //-------------------------------------------------------
      usNumThreads = max(min(atoi(apchArgs[1]),
                             MAX_NUM_THREADS),
                         1);
   }
   else
   {
      usNumThreads = DEFAULT_NUM_THREADS;
   }                                    /* endif                */
   while (usNumThreads-- && !usReturn)
   {
      usReturn = DosCreateThread(&tidThread,
                                 (PFNTHREAD)MyThreadOne,
                                 (ULONG)NULL,
                                 DEFAULT_THREAD_FLAGS,
                                 DEFAULT_THREAD_STACK);
      if (!usReturn)
         printf(" Started Thread #%2d\n",
                tidThread-1);
      else
         printf(" DosCreateThread returned %2d\n\n",
                usReturn);
   }                                    /* endfor               */
   if (!usReturn)
   {
      printf("\n Start typing and experience ");
      printf("the speed of flags for yourself...");
      printf("\n >> lower case 'x' exits << \n\n");
```

```
      fflush(stdout);

      iCharRead = getche();

      while (iCharRead != 'x')
      {
         iCharRead = getche();
      }                                    /* endwhile              */
   }
   printf("\n\n Exiting, please wait...\n\n");

   usWaitOnFlag = FALSE;
   DosSleep(2000L);
   return  usReturn;
}

VOID APIENTRY MyThreadOne()
{
   while (usWaitOnFlag)
      ;

   /*************************************************************/
   /* VERY S L O W .. . . . .                                   */
   /*************************************************************/

}
```

**FTHREAD.DEF**

```
NAME FTHREAD WINDOWCOMPAT
DESCRIPTION 'Semaphore example
             Copyright (c) 1992-1995 by Arthur Panov.
             All rights reserved.'
STACKSIZE 16384
```

**STHREAD.C**

```
#define  INCL_DOSPROCESS
#define  INCL_DOSSEMAPHORES
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#define  DEFAULT_THREAD_FLAGS 0
#define  DEFAULT_THREAD_STACK 0x4000
#define  MY_BEGIN_SEMAPHORE "\\SEM32\\BEGIN"
#define  MAX_SEM_WAIT  - 1L
#define  DEFAULT_NUM_THREADS 10
#define  MAX_NUM_THREADS 255

HEV hevKillThread;


VOID APIENTRY MyThreadOne(void);

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{
   USHORT           usNumThreads;
   TID              tidThread;
   INT              iCharRead;
   USHORT           usReturn;

   printf("\n STHREAD.EXE demonstrates the superior processor");
```

```
   printf("\n time management by allowing a user to start a ");
   printf("\n number of threads (1-255) and have all of them");
   printf("\n wait on one semaphore, while allowing keystrokes");
   printf("\n to be entered at the keyboard.");
   printf("\n (see FTHREAD.EXE, for speed comparison)");
   printf("\n\n      STHREAD.EXE [X], where X is 1-255\n\n");
   printf("\n\n lower case 'x' exits ...\n\n");

   if (usNumArgs > 1)
   {

     //-------------------------------------------------------
      // Insure that usNumThreads is in the range 1<<x<<MAX
     //-------------------------------------------------------
      usNumThreads = max(min(atoi(apchArgs[1]),
                             MAX_NUM_THREADS),
                         1);

   }
   else
   {
      usNumThreads = DEFAULT_NUM_THREADS;
   }                                    /* endif                */
   usReturn = DosCreateEventSem(MY_BEGIN_SEMAPHORE,
                                &hevKillThread,
                                NULLHANDLE,
                                FALSE);

   while (usNumThreads-- && !usReturn)
   {
      usReturn = DosCreateThread(&tidThread,
                                 (PFNTHREAD)MyThreadOne,
                                 (ULONG)NULL,
                                 DEFAULT_THREAD_FLAGS,
                                 DEFAULT_THREAD_STACK);
      if (!usReturn)
         printf(" Started Thread #%2d\n",
                tidThread-1);
      else
         printf(" DosCreateThread returned %2d\n\n",
                usReturn);
   }                                    /* endfor               */
   if (!usReturn)
   {
      printf("\n Start typing and experience ");
      printf("the speed of semaphores for yourself...");
      printf("\n >> lower case 'x' exits << \n\n");

      fflush(stdout);

      iCharRead = getche();

      while (iCharRead != 'x')
      {
         iCharRead = getche();
      }                                 /* endwhile             */
   }
   printf("\n\n Exiting, please wait...\n\n");

   usReturn = DosPostEventSem(hevKillThread);
   DosSleep(2000L);
   usReturn = DosCloseEventSem(hevKillThread);

   return  usReturn;
}
```

```
VOID APIENTRY MyThreadOne()
{
   DosWaitEventSem(hevKillThread,
                   MAX_SEM_WAIT);
}
```

**STHREAD.DEF**

```
NAME STHREAD WINDOWCOMPAT
DESCRIPTION 'Semaphore example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

**SFTHREAD.MAK**

```
ALL:    STHREAD.EXE FTHREAD.EXE

STHREAD.EXE:                        STHREAD.OBJ
        LINK386 @<<
STHREAD
STHREAD
STHREAD
OS2386
STHREAD
<<

STHREAD.OBJ:           STHREAD.C SFTHREAD.MAK
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ STHREAD.C

FTHREAD.EXE:                        FTHREAD.OBJ
        LINK386 @<<
FTHREAD
FTHREAD
FTHREAD
OS2386
FTHREAD
<<

FTHREAD.OBJ:           FTHREAD.C SFTHREAD.MAK
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ FTHREAD.C
```

Example of usage:

FTHREAD [NUMTHREADS]
or
STHREAD [NUMTHREADS]

The first command-line argument, NUMTHREADS, should be a number in the range of 11 to 255. The default number of threads created is 10; specifying a number less than 10 is unnecessary. It is not recommended to go over 100 threads with FTHREAD.EXE. Doing so even on a superfast Pentium PC will cause the system to respond to keystrokes very slowly. For example, once the CTRL-ESC keys are pressed, it may take the system several minutes to paint the PM/WPS screen. STHREAD.EXE, on the other hand, is perfectly capable of handling 255 threads in the wait state and will still provide reasonable keyboard and display response.

# Chapter 6

# DLLs

## DLL Overview

There have been many articles written about Dynamic Link Libraries, and just as many programming books have devoted at least a chapter or two to this topic. Several of these sources are listed in the Reference section of this book. This chapter concentrates on several examples of how DLLs can be used, what to look for in selecting a particular function for a DLL inclusion, and what to avoid putting in a DLL at all costs.

As the name Dynamic Link Libraries suggests, these libraries are not linked into the .EXE file during the .EXE creation, rather they get loaded dynamically into the system memory at runtime. The overwhelming advantage of DLLs is their ability to save system resources. Once the DLL is loaded, its functions are available immediately for use to all of the system's processes. On the other hand, DLLs require complex object linking and process loading tool implementation. Overall, however, DLLs live up to their claim to fame—they save system resources and offer much more rapid successive loading of executable modules that share the common functions than do statically linked .EXEs.

Another subtle advantage of DLLs is the ability of the programmer to control the functionality available to the user. For example, a programmer writing a terminal emulation application could implement a basic set of functions and label that the base package. Later, if the user demanded more functionality, the additional features could be compiled and linked into a series of DLLs that would be available to the user at an additional cost. This way users could purchase only the functionality required, nothing less, nothing more. This particular approach yields itself very nicely to a DLL implementation. One of the DLLs, for example, may contain the Zmodem protocol while another conatins a 3270 terminal emulation filter.

So far, the discussion has centered around generic DLL functionality. Windows 3.x, OS/2, NT, and Windows 95 have implemented DLL support, but the way DLLs are loaded, unloaded, initialized, and terminated differs with each operating system. Since this book concerns itself with OS/2, the OS/2 specifics are of the most interest here. One of the peculiar OS/2 implementations is the way DLLs are loaded into memory. Theoretically OS/2 has a 4 gigabyte memory limit; practically, however, the user only has 512 MB of real memory available to applications. The limit is artificially imposed by the OS/2 process loading mechanism, which is related to the OS/2 1.x compatibility issues. In particular, the LDT tiling (this is discussed by Michael Kogan, 1990) limits the 32-bit OS/2 process address space to 512 MB. The system loader will attempt to use the upper memory area for any shared code, which includes DLLs that allow shared data, while the DLLs and .EXEs with nonshared data will be loaded in the lower memory area. Figure 6.1 depicts this process.

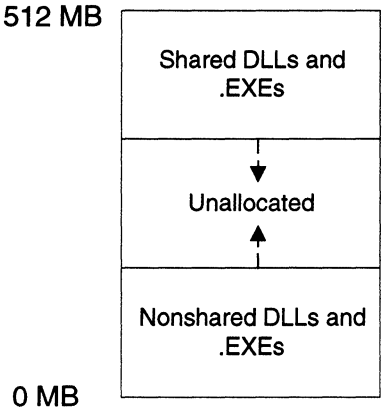



Figure 6. 1 System memory map.

# Thunking

The compatibility issues between the 32-bit and the 16-bit OS/2 modules demand a particular transition implementation called thunking. DLLs are greatly affected by this thunking mechanism. Both the 16-bit .EXE to 32-bit DLL transition, and the 32-bit .EXE to 16-bit DLL transition must be considered. The following examples explain why this is necessary.

In the 16-bit to 32-bit case, the 16-bit .EXE file may have been implented in such a way that converting it to 32-bit is tedious and unnecessary, resulting in poor performance benefits and other insignificant improvements. On the other hand, some DLLs that perform 16-bit drawing routines, for example, may benefit greatly from being converted to 32-bit modules. Also, large data structures that span 64K require careful manipulation under the 16-bit implementation; in 32-bit mode the implementation is greatly simplified.

In such cases, a developer may choose to convert the performance-sensitive sections—DLLs—of the applications to the 32-bit model, while leaving the base core as a 16-bit .EXE. The opposite transition of 32-bit to 16-bit may be required because some support libraries that the application uses are purchased 16-bit .OBJs or DLLs, and while the vendor may or may not provide the equivalent 32-bit versions of these tools, the application need not suffer a schedule slip. A 32-bit .EXE access to a 16-bit DLL can be allowed easily.

# DLL Performance

Although DLLs are designed to improve system resource usage, a few performance implications as they relate to DLL management must be understood. There are really two distinct ways to use the functions that comprise a DLL. The first and most automatic method is to create an import library, and it to resolve any references to the functions that are located inside the DLL. The system will automatically load and link the DLL functions at runtime. One thing to remember, however, is that every time a DLL function call is made, an associated address fixup must be resolved. These fixups may present somewhat of a performance impact if the memory that contains the fixup tables happens to be swapped out to disk while the call to a

DLL function is made. Before an address fixup can be resolved, the tables have to be brought back; in a resource-constrained system, this can amount to a considerable performance hit.

In order to avoid a problem with fixups Dynamic link libraries, David Reich's technique of DLL aliasing can be used. Outlined in his book *Designing OS/2 Applications*, he suggests the creation of an alias function with the same parameters as the DLL function that will be called. Then you just turn around and call the corresponding DLL function with the same parameters as the aliased one. By doing this, you are guaranteed to have only one fixup per each function in your DLL. Of course, this technique is helpful only when a particular DLL function is called numerous times throughout the .EXE. Having only a few references to a DLL function does not warrant the creation of an alias.

Portability is another good reason for aliasing some of the functions. Imagine if a developer wanted to migrate an application from one operating system to another. Sometimes using operating system-specific APIs cannot be avoided, but by aliasing some of these the migration path is much easier. The programmer is left with porting a single API reference as opposed to numerous references throughout the code.

## Simple DLL Example (32-32)

In order to preserve legacy applications' environments, the current version of OS/2 for the Intel platform allows applications to mix memory models when it comes to 16-bit and 32-bit code. It is perfectly acceptable to have a 32-bit executable call a 16-bit DLL, which in turn can call another 16-bit or 32-bit DLL. A 16-bit executable also can call a 32-bit DLL, and so forth. The only problem that may arise in doing this is memory model compatibilities. Compatibility is just a general description of pointer conversion. Both the DLL and the .EXE must know that pointer conversion must occur and take careful precautions to avoid a conversion error. Most bugs with mixed mode 16-bit/32-bit function calling are found in pointer arithmetic code. The compiler does a great job of helping the programmer convert the pointers correctly, as the following examples show. For a detailed compiler description of this thunking conversion technique, see the IBM C Set/2 *User's Guide* or IBM C/C++ *FirstStep Tools: Programming Guide*.

The most straightforward example of DLL creation and usage employs a 32-bit executable calling a 32-bit DLL. In this case, there are no memory model mixing considerations, and the programmer can freely pass values and pointers to any of the DLL functions without regard to conversion problems that are usually associated with the mixed memory environments.

The main section of the program does little more than call an externally declared function called *MyDLLFunction*, which requires two parameters. One parameter is a pointer to a function, and the other is a character pointer. Once inside the DLL, *MyDLLFunction* uses the input function pointer and passes the character pointer to that function. The user never knows how this function is implemented as it is hidden inside the DLL. At the same time, passing a function pointer to the DLL allows the DLL to call back to the .EXE if the function pointer happens to point to the function in the calling .EXE module. This, for example, may allow the DLL to "signal" the .EXE when the DLL is done with a particular task but has not completed the rest of the work yet. SIMPLE.C provides the first 32-bit to 32-bit .EXE to DLL example.

**SIMPLE.C**

```
/* Simple DLL loader                                              */

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main(VOID);

extern int MyDLLFunction(PFN,PCHAR);
extern int _System MyPrintFunctionInDLL(PCHAR,PCHAR,...);

#define  MY_CHAR_IN_VALUE "Input  character string"
#define  MY_CHAR_IN_SIZE strlen(MY_CHAR_IN_VALUE)
int main(VOID)

{

   int              rc = 0;
   PCHAR            MyCharacterPointer;

   MyCharacterPointer = (PCHAR)malloc(MY_CHAR_IN_SIZE);
   if (MyCharacterPointer)
   {
      strcpy(MyCharacterPointer,
             MY_CHAR_IN_VALUE);
      printf("\n Sending   < %s > to   DLL\n",
             MyCharacterPointer);
      rc = MyDLLFunction(MyPrintFunctionInDLL,
                         MyCharacterPointer);
      printf("\n Returned  < %s > from DLL\n",
             MyCharacterPointer);
      return (rc);
   }
   else
      return (-1);
}
```

## SIMPLE.MAK

```
ALL:    SIMPLE.EXE

SIMPLE.EXE:                         SIMPLE.OBJ
        LINK386 /NOI @<<
SIMPLE
SIMPLE
SIMPLE
OS2386+MYDLL
SIMPLE
<<


SIMPLE.OBJ:                         SIMPLE.C
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ SIMPLE.C
```

## SIMPLE.DEF

```
NAME simple WINDOWCOMPAT
DESCRIPTION 'simple 32-32 DLL example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

## MYDLL.C

```
#include <os2.h>
#include <stdio.h>
#include <string.h>
#define  MY_CHAR_OUT_VALUE "Output character string"
int MyDLLFunction(PFN,PCHAR);
```

```
int _System MyPrintFunctionInDLL(PCHAR,PCHAR,...);

int MyDLLFunction(PFN MyFunctionPointer,PCHAR MyCharacterPointer)
{
   int              rc = 0;

   rc = (MyFunctionPointer)("\n Modifying < %s > in   DLL\n",
                            MyCharacterPointer);
   return (rc);
}

int _System MyPrintFunctionInDLL(PCHAR First,PCHAR Second,...)
{
   printf(First,
          Second);
   strcpy(Second,
          MY_CHAR_OUT_VALUE);
   return (0);
}
```

**MYDLL.MAK**

```
ALL:    MYDLL.DLL MYDLL.LIB

MYDLL.LIB:                          MYDLL.DLL
        IMPLIB MYDLL.LIB MYDLL.DEF

MYDLL.DLL:                          MYDLL.OBJ
        LINK386 /NOI @<<
MYDLL
MYDLL.DLL
MYDLL
OS2386
MYDLL
<<


MYDLL.OBJ:                    MYDLL.C MYDLL.MAK MYDLL.DEF
        ICC -C+ -Ge- -Kb+ -Ss+ MYDLL.C
```

**MYDLL.DEF**

```
LIBRARY MYDLL
DESCRIPTION 'simple 32-32 DLL example
             Copyright (c) 1992-1995 by Arthur Panov.
             All rights reserved.'
DATA MULTIPLE NONSHARED
EXPORTS
       MyDLLFunction
       MyPrintFunctionInDLL
STACKSIZE 16384
```

# Creating the .EXE and the DLL

A couple of things need to be said about how this .EXE and DLL are built. First, the .EXE is compiled the same way .EXEs always are compiled. There are no special considerations. There are, however, two ways to link the .OBJs to create an .EXE that uses DLLs.

The first method employs an IMPORTS statement in the .EXE DEF file and specifies the exact DLL name and the exported function names. The second one relies on a DLL import library that is linked in just as a

static library would be. Using the import library is more of an automatic linking process, because you do not have to keep track of all of the functions called in the .EXE. From a maintenance standpoint, the import library is the preferred linking choice. The import library is created by running the IMPLIB.EXE, an OS/2 Toolkit utility, and specifying the DLL DEF file or the DLL itself as a parameter. The import library allows the linker to resolve all of the references to the DLL resident functions. Note that the import library or the DEF file with the IMPORTS keyword and functions defined is required only when the DLL resident functions are invoked automatically by the .EXE.

OS/2 provides another method of loading the DLLs at runtime and calling the DLL resident functions explicitly. In this fashion, neither the imports library nor the IMPORTS keyword and functions' specification is needed in the DEF file. An example of this loading technique is covered later in this chapter.

The DLL can be considered just as a special .EXE file, and in the earlier releases of some of the operating systems DLLs actually had .EXE extensions. The main difference is that a DLL cannot execute without a parent .EXE. In comparison with the .EXE creation, the DLL files must be compiled with a DLL flag ON (for C-Set: /Ge-). This may not be a requirement for other compilers. Next the DLL object code must be LINKed and the DLL created. The most important file for the LINK step (and again, this is for IBM C-Set/2 C/C++) is the proper use of the module definition file (DEF). The DEF file specifies how the DLL will be loaded, named, shared, and so forth. LINK386.EXE, a 32-bit linker for OS/2, recognizes the module definition keywords listed in Table 6.1.

Table 6.1 Module Definition Keywords

| Keyword | Description |
|---|---|
| BASE | Preferred load address |
| CODE | Code segments attributes |
| DATA | Data segments attributes |
| DESCRIPTION | Module description |
| EXETYPE | DLL operating system type |
| EXPORTS | Functions exported by DLL |
| IMPORTS | Functions imported by EXE/DLL |
| HEAPSIZE | Local heap size |
| LIBRARY | DLL name |
| NAME | EXE name |
| OLD | Preserve old ordinal numbers |
| PHYSICAL DEVICE | Device driver name |
| PROTMODE | Protected mode only module |
| SEGMENTS | Segments attributes |
| STACKSIZE | Local stack size |
| STUB | Prepended DOS executable module |
| VIRTUAL DEVICE | Virtual device driver name |

The definition module must specify the correct combination of keywords so that the linker can construct the DLL or .EXE file correctly.

Detailed explanation of the linker recognized keywords can be found in the online OS/2 Toolkit documentation (OS/2 Tools Reference: TOOLINFO.INF).

**Gotcha!**

IMPORTS 1mydll.MyFunction1 statement fails due to a parser. The parser of IMPORTS does not expect a number as the first character of a DLL even though the DLL name is a legal OS/2 file name.

## 16-32, 32-16 Transitions

OS/2 supports four classes of applications:

- Pure 16-bit
- Mixed 16-bit
- Pure 32-bit
- Mixed 32-bit

The pure 16-bit application development was left behind in OS/2 1.x days, and the pure 32-bit application development with DLLs is covered in the SIMPLE DLL example. This leaves only two interesting cases:

- 16-bit .EXE calling 32-bit DLL
- 32-bit .EXE calling 16-bit DLL

The most interesting item in mixed programming is the transition from one memory model to the other and back. This transition in OS/2 is achieved with the help of a mapping layer technique called thunking. A 32-16 thunk and a 16-32 thunk are possible. Thunking involves converting 32-bit pointers to 16-bit pointers, and vice versa. This thunking mechanism is a requirement for all mixed mode applications. Luckily for the programmer, the compiler generally supports the thunking transitions automatically.

The 16-bit memory model has 64K segmentation size limitations, while the 32-bit memory model does not. Therefore, if a 16-bit .EXE needed to manipulate a large data area (>64K), rewriting just the manipulation routines and composing them into a 32-bit DLL would work.

# Call a 32-Bit DLL from a 16-Bit Program

The 16-bit to 32-bit example is a simple checksum program that operates on a data area greater than 64K in size. Both the DLL and the .EXE source code are rather simple. The interesting part is the way the functions are declared in the 16-bit source and in the 32-bit source. The sizes of the arguments must match across the transition boundary. In this case, all of the parameters and the return value are of the same size in the 16-bit and the 32-bit sections of the code.

The 16-bit executable makes a call to the 32-bit DLL requesting the checksum value by passing a file name to the 32-bit DLL function. The 32-bit DLL is invoked automatically by the system. The DLL function proceeds to use the 32-bit APIs to determine the file size (*DosQueryPathInfo*), allocate the memory (*malloc* > 64K), open the file (*DosOpen*), and read the data (*DosRead*). The checksum calculation is made next, and the values are returned to the caller.

**HOWBIG.C**

```
#include <stdio.h>
int main(void);

extern unsigned long far pascal HowMany(unsigned int *,char *);
```

```
#define  FILE_NAME     "BIGFILE"
int main(void)

{
   unsigned long    ulCount = 0;
   unsigned int     usCheckSum = 0;

   printf("\n Now inside 16-bit %s",
          __FILE__);
   printf("\n size of usCheckSum (USHORT * )  = %d",
          sizeof(int *));
   printf("\n size of FILE_NAME  (char *   )  = %d",
          sizeof(char *));

   printf(
      "\n About to call the 32-bit DLL function automatically\n")
      ;

   ulCount = HowMany(&usCheckSum,
                     FILE_NAME);

   if (ulCount == 0)
      printf("\n Could not calculate checksum, sorry!\n\n");
   else
      printf("\n File: %s, checksum: %04X, Count: %ld\n\n",
             FILE_NAME,
             usCheckSum,
             ulCount);
   return (0);
}
```

## HOWBIG.MAK

```
ALL:    HOWBIG.EXE

HOWBIG.EXE:                         HOWBIG.OBJ
        LINK @<<
HOWBIG
HOWBIG.EXE
HOWBIG
COUNT
HOWBIG
<<


HOWBIG.OBJ:                  HOWBIG.C HOWBIG.MAK HOWBIG.DEF
        cl -c -AL HOWBIG.C
```

## HOWBIG.DEF

```
NAME HOWBIG WINDOWCOMPAT
DESCRIPTION 'simple 16-32 EXE example
             Copyright (c) 1992-1995 by Arthur Panov.
             All rights reserved.'
STACKSIZE 16384
```

## COUNT.C

```
#define  INCL_DOSFILEMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
```

```
ULONG _Far16 _Pascal HowMany(USHORT *_Seg16 usCheckSum,char
                             *_Seg16);

ULONG _Far16 _Pascal HowMany(USHORT *_Seg16 usCheckSum,char
                             *_Seg16 szFileName)
{

#define  MY_MAX_COUNT  10000
   APIRET           rc = 0;
   FILESTATUS3      fsts3Info;
   ULONG            ulCount = 0;
   UCHAR            *pBigBuffer = 0;
   HFILE            hFileHandle;
   ULONG            ulAction = 0L;
   ULONG            ulBytes = 0L;

   printf("\n Now inside 32-bit %s",
          __FILE__);

   printf("\n size of usCheckSum (USHORT * _Seg16) = %d",
          sizeof(USHORT *_Seg16));
   printf("\n size of szFileName (char * _Seg16  ) = %d",
          sizeof(char *_Seg16));

   rc = DosQueryPathInfo(szFileName,
                         FIL_STANDARD,
                         &fsts3Info,
                         sizeof(FILESTATUS3));
   if (rc)
      return (0);
   else
   {
      pBigBuffer = malloc(fsts3Info.cbFileAlloc);
      if (pBigBuffer)
      {
         rc = DosOpen((PSZ)szFileName,
                      &hFileHandle,
                      &ulAction,
                      0L,
                      FILE_READONLY,
                      OPEN_ACTION_OPEN_IF_EXISTS,
                      OPEN_FLAGS_SEQUENTIAL|
                         OPEN_SHARE_DENYREADWRITE|
                         OPEN_ACCESS_READONLY,
                      0L);
         if (rc)
         {
            printf("\n\n Could not open <BIGFILE>, rc=%d\n\n",
                   rc);
            return (0);
         }
         else
         {
            rc = DosRead(hFileHandle,
                         pBigBuffer,
                         fsts3Info.cbFile,
                         &ulBytes);
            if (rc || fsts3Info.cbFile != ulBytes)
            {
               printf("\n\n Could not read the data             \
    rc=%d, FileSize=%d, BytesRead=%d\n",
                      rc,
                      fsts3Info.cbFile,
                      ulBytes);
               return (0);
            }
            else
            {
```

```
                *usCheckSum = 0;
                do
                {
                   *usCheckSum += (USHORT)(*pBigBuffer++);
                   ulCount++;
                   fsts3Info.cbFile--;
                }  while (fsts3Info.cbFile);

             }
          }

      }
      else
      {
         printf(
      "\n\n Could not allocate enough space to read <BIGFILE>\n")
            ;
         return (0);
      }
      free(pBigBuffer);
      rc = DosClose(hFileHandle);
      printf("\n About to leave the 32-bit %s.\n",
             __FILE__);
      return (ulCount);
   }

}
```

## COUNT.MAK

```
ALL:    count.DLL count.LIB

count.LIB:                          count.DLL
        IMPLIB count.LIB count.DEF

count.DLL:                          count.OBJ
        LINK386 /NOI @<<
count
count.DLL
count
OS2386
count
<<


count.OBJ:                    count.C count.MAK count.DEF
        ICC -C+ -Ge- -Kb+ -Ss+ count.C
```

## COUNT.DEF

```
LIBRARY COUNT
DESCRIPTION 'simple 16-32 DLL example
             Copyright (c) 1992-1995 by Arthur Panov.
             All rights reserved.'
DATA MULTIPLE NONSHARED
EXPORTS
       HOWMANY
STACKSIZE 16384
```

## Pointer Declarations

When passing a pointer to a 16-bit function from a 32-bit program, the _Seg16 type qualifier should be used. For example:

```
char * _Seg16 ptrFor16Bit ;
```

declares this pointer to be a segmented pointer that is usable in 16-bit functions. It is also usable in a 32-bit program.

## Calling a 16-Bit DLL from a 32-Bit Program

A similar transition takes place when calling the 16-bit DLL from a 32-bit .EXE. The function declarations utilize the same keywords that were used in the 16-bit to 32-bit example earlier. This particular program attempts to determine whether the computer's serial ports utilize the faster buffered I/O National 16550 UARTs (Universal Asynchronous Receiver/Transmitter). In order to do this, the program employs a 16-bit I/O DLL called 16BITIO.DLL. This DLL contains two functions, *my_inp* and *my_outp*. These functions will directly input or output a single byte from or to the specified I/O port. A 16-bit DLL is used to demonstrate how quickly the presence of the National 16550 UART can be determined. The algorithm for determining the presence of the UART is trivial and is described in the *National UART Devices Data Book*.

**Gotcha!**

In order to perform direct h/w I/O the code *must* run at the RING 2 Input/Output Privilege Level (IOPL). This is why the appropriate CODE statement is found in the DEF file for the 16BITIO.DLL. Unfortunately, there is no IOPL support for the 32-bit DLLs; thus 16-bit IOPL DLLs must be used in such cases. This may change in future releases, but for now we are limited to using 16-bit code.

**AUT16550.C**

```
/*  Assume                                                          */
/*                                                                  */
/*        COM1 -> 0x3F8                                             */
/*        COM2 -> 0x2F8                                             */
/*                                                                  */
/*  One attempts to first clear the 16550 FIFO by writing a 0x00 to*/
/*  the FIFO Control register at offset 0x02.  Then one attempts to*/
/*  enable the FIFOs by setting bit0 of the FIFO Control register*/
/*  at offset 0x02.  Reading the Interrupt Identification register*/
/*  at offset 0x02 will tell one if 16550 is present.               */
/*                                                                  */
/*  Automatic Loading of DLL functions                              */
/*                                                                  */

#include <stdio.h>
#include <stdlib.h>
#include "aut16550.h"
int main(void);

#define  BIT_6_7_SET   0x00C0
int main(void)

{
   unsigned         Byte = 0;

   printf("\n\n Attempting to find 16550 UART ...");/* test
```

```
                                          COM1                  */
   my_outp(MY_COM1+MY_FIFO_CTRL,
           0x00);                         /* Clear the FIFO reg   */
   Byte = my_inp(MY_COM1+MY_INT_ID);
   Byte &= BIT_6_7_SET;
   if (!Byte)
   {

      my_outp(MY_COM1+MY_FIFO_CTRL,
              0x01);                      /* Set the FIFO reg     */
      if (my_inp(MY_COM1+MY_INT_ID)&BIT_6_7_SET)
         printf(
           "\n\n 16550 appears to be present for COM1->0x3F8.\n")
          ;
      else
         printf(
            "\n\n 16550 appears to be absent for COM1->0x3F8.\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM1->0x3F8. Exiting ... \n\n");
      return (-1);
   }                                      /* test COM2 loop? :)   */
   my_outp(MY_COM2+MY_FIFO_CTRL,
           0x00);                         /* Clear the FIFO reg   */
   if (!(Byte = (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)))
   {

      my_outp(MY_COM2+MY_FIFO_CTRL,
              0x01);                      /* Set the FIFO reg     */
      if (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)
         printf(
           "\n 16550 appears to be present for COM2->0x2F8.\n\n")
          ;
      else
         printf(
            "\n 16550 appears to be absent for COM2->0x2F8.\n\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM2->0x2F8. Exiting ... \n\n");
      return (-1);
   }

   return (0);
}
```

**AUT16550.H**

```
/* Header file for the 16-bit LIB/DLL used to perform IOPL i/o calls */
/* A. Panov 1993,1994,1995                                           */


extern unsigned short _Far16 _Pascal my_inp  (unsigned short);
extern unsigned short _Far16 _Pascal my_outp (unsigned short, unsigned short);

#define MY_COM1          0x3F8
#define MY_COM2          0x2F8
#define MY_INT_ENABLE    1
#define MY_INT_ID        2
#define MY_FIFO_CTRL     2
```

```
#define MY_LINE_CTRL    3
#define MY_MODEM_CTRL   4
#define MY_LINE_STATUS  5
#define MY_MODEM_STATUS 6
#define MY_SCRATCH      7
```

**AUT16550.MAK**

```
ALL:    AUT16550.EXE

AUT16550.EXE:                          AUT16550.OBJ
        LINK386 /NOI @<<
AUT16550
AUT16550
AUT16550
OS2386+16bitio
AUT16550
<<


AUT16550.OBJ:                          AUT16550.C AUT16550.MAK AUT16550.DEF
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ AUT16550.C
```

**AUT16550.DEF**

```
NAME AUT16550 WINDOWCOMPAT
DESCRIPTION 'AUT16550 example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

**16BITIO.C**

```
/* 16-bit I/O dll                                                    */

_acrtused = 0;

#include <conio.h>
int far _cdecl my_inp(unsigned);

int far _cdecl my_outp(unsigned,unsigned);
unsigned far _cdecl my_inpw(unsigned);
unsigned far _cdecl my_outpw(unsigned,unsigned);

int far _cdecl my_inp(unsigned usPort)
{
   return (inp(usPort));
}

int far _cdecl my_outp(unsigned usPort,unsigned usValue)
{
   return (outp(usPort,
                usValue));
}

unsigned far _cdecl my_inpw(unsigned usPort)
{
   return (inpw(usPort));
}

unsigned far _cdecl my_outpw(unsigned usPort,unsigned usValue)
```

```
{
   return (outpw(usPort,
                 usValue));
}
```

**16BITIO.MAK**

```
ALL:    16BITIO.DLL 16BITIO.LIB

16BITIO.LIB:                        16BITIO.DLL
        IMPLIB 16BITIO.LIB 16BITIO.DEF

16BITIO.DLL:                        16BITIO.OBJ
        LINK /NOI @<<
16BITIO
16BITIO.DLL
16BITIO

16BITIO
<<


16BITIO.OBJ:                  16BITIO.C
        cl  -c -AL -G2s -Fc 16BITIO.C
```

**16BITIO.DEF**

```
LIBRARY INITINSTANCE
PROTMODE
DESCRIPTION '16bitIO example
             Copyright (c) 1992-1995 by Arthur Panov.
             All rights reserved.'
DATA NONSHARED
SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL
EXPORTS
       _my_inp   1
       _my_outp  2
       _my_inpw  1
       _my_outpw 2
STACKSIZE 4096
```

# Loading/Unloading of DLLs

As was mentioned earlier, developers have two choices about loading and unloading the DLLs. They may choose to have the system do the work for them automatically, or they may decide to have complete control over how DLL functions are loaded, unloaded, and called.

The automatic loading and unloading of DLLs is the most headache-free, low-maintenance option. But it does have some drawbacks. The application cannot be started without the DLL being present in the LIBPATH. Nor can the resources used by the DLL be freed up until the application exits. If resource considerations are of great importance, the manual method of loading and unloading DLLs must be used. The benefits of manual manipulation of DLL functions are obvious: low memory usage, no initialization of DLLs at application startup time, resources can be freed when not needed, application can recover if DLL is missing or corrupted, and so on. The drawback to using the manual option is complexity.

The previous example of 32-bit to 16-bit CHK16550.EXE is used here to illustrate the manual loading, usage, and unloading of a DLL. First a call to the *DosLoadModule* is made.

```
APIRET DosLoadModule( PSZ pszName, ULONG cbName,
                      PSZ pszModuleName, PHMODULE phMod )
```

*pszName* is the address of buffer used in case of failure; on output it will contain the name of the object that caused the failure. *cbName* is the size of the *pszName* buffer. *pszModuleName* is the name of the dynamic link library, and *phMod* is a pointer that on output contains the handle for the dynamic link module.

Next, the starting address of a function is found using the *DosQueryProcAddr.*

```
APIRET DosQueryProcAddr(      HMODULE hmod, ULONG ulOrd,
                              PSZ pszName, PFN *ppfn )
```

*hmod* is the dynamic link module handle. *ulOrd* is the ordinal number of the function whose address is to be found. If this value is 0, the *pszName* argument is used to find the desired function. *pszName* contains the function name that is being referenced. *ppfn* is a pointer to a PFN that on output contains the procedure address.

Once the addresses of *my_inp* and *my_outp* are known, the program runs the same way. Last, the *DosFreeModule* is called to release the DLL and effectively unload it from CHK16500.EXE's memory space.

```
APIRET DosFreeModule( HMODULE hmod )
```

This function has only one parameter, *hmod*, which is the handle of the module that is to be freed.

**MAN16550.C**

```
/*  Assume                                                          */
/*                                                                  */
/*        COM1 -> 0x3F8                                             */
/*        COM2 -> 0x2F8                                             */
/*                                                                  */
/*One attempts to first clear the 16550 FIFO by writing a 0x00 to*/
/*the FIFO Control register at offset 0x02.  Then one attempts to*/
/*enable the FIFOs by setting bit0 of the FIFO Control register*/
/*at offset 0x02.  Reading the Interrupt Identification register*/
/*at offset 0x02 will tell one if 16550 is present.          */
/*                                                                  */

#define  INCL_DOSMODULEMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "man16550.h"
int main(void);

#define  BIT_6_7_SET   0x00C0
#define  IGNORE_ORDINAL_NUMBER 0
int main(void)

{
   unsigned          Byte = 0;

   HMODULE           hmod;
   APIRET            rc = 0;
   UCHAR             szDLLName[CCHMAXPATH];/* CCHMAXPATH ->
```

```
                                        bsedos.h via toolkit */
UCHAR           szBadName[CCHMAXPATH];/* Load the
                                        16bitio.dll          */

strcpy(szDLLName,
       ".\\16bitio.dll");           /* Look for DLL in the
                                        same directory       */
rc = DosLoadModule(szBadName,
                   CCHMAXPATH-1,
                   szDLLName,
                   &hmod);

if (rc)
{
   printf("\n Could not load %s successfully, bad %s, rc = \n"
      ,
          szDLLName,
          szBadName,
          rc);
   return (-1);
}
else
   printf("\n Loaded %s successfully\n",
          szDLLName);               /* Get the my_inp
                                        function address     */
rc = DosQueryProcAddr(hmod,
                      IGNORE_ORDINAL_NUMBER,
                      "MY_INP",
                      (PFN *)&my_inp);

if (rc)
{
   printf("\n Could not find address for my_inp, rc = \n",
          rc);
   DosFreeModule(hmod);
   return (-1);
}
else
   printf("\n Found my_inp()  function address\n");/* Get
                                        the my_outp function
                                        address              */
rc = DosQueryProcAddr(hmod,
                      IGNORE_ORDINAL_NUMBER,
                      "MY_OUTP",
                      (PFN *)&my_outp);

if (rc)
{
   printf("\n Could not find address for my_outp, rc = \n",
          rc);
   DosFreeModule(hmod);
   return (-1);
}
else
   printf("\n Found my_outp() function address\n");

printf("\n\n Attempting to find 16550 UART ...");/* test
                                        COM1                 */
my_outp(MY_COM1+MY_FIFO_CTRL,
        0x00);                      /* Clear the FIFO reg   */
Byte = my_inp(MY_COM1+MY_INT_ID);
Byte &= BIT_6_7_SET;
if (!Byte)
{

   my_outp(MY_COM1+MY_FIFO_CTRL,
           0x01);                   /* Set the FIFO reg     */
   if (my_inp(MY_COM1+MY_INT_ID)&BIT_6_7_SET)
      printf(
```

```
            "\n\n 16550 appears to be present for COM1->0x3F8.\n")
             ;
      else
         printf(
            "\n\n 16550 appears to be absent for COM1->0x3F8.\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM1->0x3F8. Exiting ... \n\n");
      DosFreeModule(hmod);
      return (-1);
   }                                    /* test COM2 loop? :)   */
   my_outp(MY_COM2+MY_FIFO_CTRL,
           0x00);                       /* Clear the FIFO reg   */
   if (!(Byte = (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)))
   {

      my_outp(MY_COM2+MY_FIFO_CTRL,
              0x01);                    /* Set the FIFO reg     */
      if (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)
         printf(
           "\n 16550 appears to be present for COM2->0x2F8.\n\n")
            ;
      else
         printf(
            "\n 16550 appears to be absent for COM2->0x2F8.\n\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM2->0x2F8. Exiting ... \n\n");
      DosFreeModule(hmod);
      return (-1);
   }

   /*************************************************************/
   /* Free the 16bitio.dll module                               */
   /*************************************************************/

   rc = DosFreeModule(hmod);

   if (rc)
   {
      printf("\n Could not free %s successfully, rc = \n",
             szDLLName,
             rc);
      return (-1);
   }
   else
      printf("\n Freed %s successfully\n",
             szDLLName);

   return (0);
}
```

**MAN16550.H**

```
/* Header file for the 16-bit LIB/DLL used to perform IOPL i/o calls */
/* A. Panov 1993,1994,1995                                           */

unsigned short (*  _Far16 _Pascal my_inp )(unsigned short);
unsigned short (*  _Far16 _Pascal my_outp)(unsigned short, unsigned short);
```

```
#define MY_COM1          0x3F8
#define MY_COM2          0x2F8
#define MY_INT_ENABLE    1
#define MY_INT_ID        2
#define MY_FIFO_CTRL     2
#define MY_LINE_CTRL     3
#define MY_MODEM_CTRL    4
#define MY_LINE_STATUS   5
#define MY_MODEM_STATUS  6
#define MY_SCRATCH       7
```

**MAN16550.MAK**

```
ALL:    MAN16550.EXE

MAN16550.EXE:                          MAN16550.OBJ
        LINK386 /NOI @<<
MAN16550
MAN16550
MAN16550
OS2386
MAN16550
<<


MAN16550.OBJ:      MAN16550.C MAN16550.MAK MAN16550.DEF MAN16550.H
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ MAN16550.C
```

**MAN16550.DEF**

```
NAME MAN16550 WINDOWCOMPAT
DESCRIPTION 'MAN16550 example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

**Gotcha!**

If using a *DosExitList* in a DLL, the DLL cannot be freed via *DosFreeModule* until the exit list function has run

## Optimizing Performance in DLLs

System performance can be improved significantly by efficient use of DLLs. These performance improvements can be gained from something as simple as combining several smaller DLLs into one larger one, or by using David Reich's "aliasing" technique in helping the fix-up problems. The following checklist lists some good DLL candidates.

1. Rarely called functions
2. Functions that add functionality to the base product
3. Functions that remove functionality from the base product
4. Functions that can be shared among applications
5. Functions with frequently changing internal implementation
6. Internationalization enabling functions
7. Help and Message type functions

# Chapter 7

# Exception Handling

OS/2 provides an opportunity for a program to interrupt system errors and handle them in their own manner. These system "errors" are known as exceptions and are not really errors, but more abnormal conditions. Some types of exceptions are guard-page exceptions, divide-by-zero exceptions, illegal instruction, and access violation (or protection violation). Most everyone has seen the black protection violation screen, which only lets the user end the program. Wouldn't it be nice to intercept that exception and either fix the problem ahead of time or at least provide an error message that was somewhat intelligible to the user? Exception handlers are the answer.

There are two kinds of exceptions generated by the operating system, asynchronous exceptions and synchronous exceptions. Asynchronous exceptions are caused by events external to a thread. Synchronous exceptions are caused by events internal to a thread. Some common synchronous exceptions include guard-page exceptions, divide-by-zero exceptions, and access violations. All the asynchronous exceptions generate one of two exception types, XCPT_SIGNAL or XCPT_ASYNC_PROCESS_TERMINATE. Asynchronous exceptions, except for the XCPT_ASYNC_PROCESS_TERMINATE exception, are also known as signal exceptions. Signal exceptions are available only to non-Presentation Manager processes.

When a synchronous exception occurs, the operating system sends an exception just to the thread causing the exception. If the operating system terminates the application, a XCPT_ASYNC_PROCESS_TERMINATE is sent to all the other threads in the process.

When an asynchronous exception occurs, the operating system sends an exception just to the main thread.

## How to Register an Exception Handler

Exception handlers are registered on a per-thread basis using the function

```
DosSetExceptionHandler( PEXCEPTIONREGISTRATIONRECORD)
```

Exception handlers can be "nested" as a chain of exception-handling functions. The operating system will call the last handler in the chain; after that function has completed, it may call the next-to-last handler, and so on. An exception handler will do its work and then return a value to the operating system that indicates whether to continue with the next exception handler registered in the chain or to dismiss the exception.

The EXCEPTIONREGISTRATIONRECORD data structure forms a linked list of exception handlers. The first element in the structure is a pointer to either the next exception handler or an end-of-list marker, and is filled in by the operating system. The second is a pointer to the exception-handling function currently being registered and should be filled in by the developer. When registering an exception handler, this

structure must be local to the procedure that contains *DosSetExceptionHandler*, as opposed to a global structure.

**Gotcha!**
Before exiting your program, make sure you call the function *DosUnsetExceptionHandler*. If you do not, you will probably see a stack overflow error.

## What an Exception Handler Looks Like

An exception handler should use the following prototype.

```
APIRET APIENTRY ExceptHandlerFn( EXCEPTIONREPORTRECORD *, EXCEPTIONREGISTRATIONRECORD *,
CONTEXTRECORD *, PVOID reserved )
```

The EXCEPTIONREPORTRECORD structure is a data structure

```
struct _EXCEPTIONREPORTRECORD
   {
   ULONG   ExceptionNum;      /* exception number */
   ULONG   fHandlerFlags;
   struct  _EXCEPTIONREPORTRECORD *NestedExceptionReportRecord;
   PVOID   ExceptionAddress;
   ULONG   cParameters;       /* Size of Exception Specific Info */
   ULONG   ExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS];
                              /* Exception Specfic Info */
   };
```

that describes the exception and includes the exception type and other exception information.

The EXCEPTIONREGISTRATIONRECORD structure

```
struct _EXCEPTIONREGISTRATIONRECORD
   {
   struct _EXCEPTIONREGISTRATIONRECORD * volatile prev_structure;
   _ERR * volatile ExceptionHandler;
   };
```

is described in the last section, "How to Register an Exception Handler."

The CONTEXTRECORD * structure

```
struct _CONTEXT
   {
   ULONG ContextFlags;
   ULONG   ctx_env[7];
   FPREG   ctx_stack[8];
   ULONG ctx_SegGs;
   ULONG ctx_SegFs;
   ULONG ctx_SegEs;
   ULONG ctx_SegDs;
   ULONG ctx_RegEdi;
   ULONG ctx_RegEsi;
   ULONG ctx_RegEax;
```

```
ULONG ctx_RegEbx;
ULONG ctx_RegEcx;
ULONG ctx_RegEdx;
ULONG ctx_RegEbp;
ULONG ctx_RegEip;
ULONG ctx_SegCs;
ULONG ctx_EFlags;
ULONG ctx_RegEsp;
ULONG ctx_SegSs;
};
```

is an input/output parameter that contains register contents at the time of the exception. If the exception handler will return XCPT_CONTINUE_EXECUTION, the structure can be modified. If it is modified without XCPT_CONTINUE_EXCEPTION being specified, very bad things will happen.

The last parameter, the DISPATCHERCONTEXT structure, is undocumented because it should never be modified.

The 486 chip uses the address at FS:0 to point to the address of the first exception registration record. Many compilers implement exception handlers by modifying this value directly, rather than using the OS/2 API, in order to improve performance.

## Signal Exceptions

Signal exceptions are special types of exceptions generated by only three events: when the user presses Ctrl+C, when the user presses Ctrl+Break, and when another process terminates the application with the *DosKillProcess* function.

In order to receive the Ctrl+C and the Ctrl+Break exceptions, the thread must call *DosSetSignalExceptionFocus*. The kill process signal is sent whether this function is used or not.

## Dos and Don'ts for Exception Handlers

- Always deregister the exception handler. Some compilers will do this for you if you use the #pragma handler. This pragma will set and unset the exception handler where necessary. If you use *DosSetExceptionHandler*, you must use *DosUnsetExceptionHandler*.
- Make sure all semaphores are released if the exceptions are not being handled over to the system default exception handler (by returning XCPT_CONTINUE_EXCEPTION).
- An exception handler needs approximately 1.5K of stack in the process to be called. The process will be terminated if there is not enough stack space.
- An error in the exception handler may generate a recursive exception condition. This creates a situation that is very difficult to debug. Life will get much easier for the developer if the exception handler is unset when a fatal error condition occurs.

## DosExitList and Exception Handlers

When all threads in a process receive the process termination exception, a process will execute the functions specified by *DosExitList*. The functions *DosCreateThread* and *DosExecPgm* should not be used in an exit list routine.

## A Guard Page Example

The following example illustrates guard-page handling. Guard pages provide an extra level of protection for two things, data and thread stacks. A guard page is like a traffic cop with a large brick wall as a stop

sign. When someone hits that brick wall, he or she is going to have some reaction, in this case, a guard-page exception. This gives the programmer a chance to clean up the problem. When a page of memory is committed, it also can be marked as a guard page. If the application writes to the edge of the guard page, top or bottom, a guard-page exception is generated. The default behavior is designed for dynamic stack growth, and stacks grow downward. Because of this, the operating system will look to see if the next *lower* page is free, and if so, commit it. However, an exception handler gives the programmer some flexibility. If the application so chooses, it can commit the next higher page in the exception handler, and then return control back to the function that generated the guard-page exception. This memory management scheme is the method used by most compilers to control thread stack growth.

**GP.C**

```
#define  INCL_DOSMEMMGR
#define  INCL_DOSEXCEPTIONS
#include <os2.h>
#include <stdio.h>
#define  NUM_PAGES      8
#define  SZ_PAGE        4096
ULONG MyExceptionHandler(PEXCEPTIONREPORTRECORD pTrap);

PBYTE            pbBase;
BOOL             bGuardUp;

INT main(USHORT usNumArgs,PCHAR apchArgs[])
{
   LONG             lIndex;
   EXCEPTIONREGISTRATIONRECORD errRegister;
   APIRET           arReturn;

   pbBase = NULL;

   if (usNumArgs > 1)
   {
      bGuardUp = TRUE;
      printf("Guarding up\n");
   }
   else
   {
      bGuardUp = FALSE;
      printf("Guarding down\n");
   }                                    /* endif                */
   errRegister.ExceptionHandler = (_ERR *)&MyExceptionHandler;
   arReturn = DosSetExceptionHandler(&errRegister);
   printf("DosSetExceptionHandler returns %ld\n",
          arReturn);

   /*************************************************************/
   /* allocate some memory                                      */
   /*************************************************************/

   arReturn = DosAllocMem((PPVOID)&pbBase,
                          NUM_PAGES        *SZ_PAGE,

                          PAG_READ|PAG_WRITE);

   printf("DosAllocMem returns %ld ( pbBase = %p ) \n",
          arReturn,
          pbBase);

   if (!bGuardUp)
   {
    //-------------------------------------------------------
      // Commit last page and set to guard page
    //-------------------------------------------------------
```

```
      arReturn = DosSetMem(pbBase+((NUM_PAGES-1)*SZ_PAGE),
                           SZ_PAGE,
                           PAG_COMMIT|PAG_READ|PAG_WRITE|
                              PAG_GUARD);
      printf("Return Code from DosSetMem, ""%ld - pbBase = %p\n",
             arReturn,
             pbBase);
    //-------------------------------------------------------------
      // Write to pages, from top to bottom
    //-------------------------------------------------------------

      for (lIndex = (NUM_PAGES *SZ_PAGE)-1L; lIndex >= 0L; lIndex
         -= 0x0010L)
      {
         printf("\rWriting to offset 0x%08lX",
                lIndex);
         pbBase[lIndex] = 1;
         printf("\rWritten to offset 0x%08lX",
                lIndex);
      }                                  /* endfor               */
   }
   else
   {
    //-------------------------------------------------------------
      // Commit first page and set to guard page
    //-------------------------------------------------------------
      arReturn = DosSetMem(pbBase,
                           SZ_PAGE,
                           PAG_COMMIT|PAG_READ|PAG_WRITE|
                              PAG_GUARD);
      printf("Return Code from DosSetMem, ""%ld - pbBase = %p\n",
             arReturn,
             pbBase);
    //-------------------------------------------------------------
      // Write to pages, from bottom to top
    //-------------------------------------------------------------

      for (lIndex = 0L; lIndex < (NUM_PAGES *SZ_PAGE); lIndex +=
         0x0010L)
      {
         printf("\rWriting to offset 0x%08lX",
                lIndex);
         pbBase[lIndex] = 1;
         printf("\rWritten to offset 0x%08lX",
                lIndex);
      }                                  /* endfor               */
   }                                     /* endif                */
   printf("\n");
 //-------------------------------------------------------------
   // Free memory area
 //-------------------------------------------------------------

   printf("Freeing pbBase = %p\n",
          pbBase);
   arReturn = DosFreeMem(pbBase);

   printf("Done\n");
   return 0;
}

ULONG MyExceptionHandler(PEXCEPTIONREPORTRECORD perrTrap)
{
   ULONG            ulReturn;
   APIRET           arReturn;
   PBYTE            pbTrap;

   ulReturn = XCPT_CONTINUE_SEARCH;

   if (perrTrap->ExceptionNum == XCPT_GUARD_PAGE_VIOLATION)
```

```
{
   DosBeep(300,
           100);
   printf("\n *** Guard exception *** \n");

   pbTrap = (PBYTE)perrTrap->ExceptionInfo[1];
   //-------------------------------------------------------
   // Check that the fault is within our memory zone, so that
   // we won't interfere with system handling of stack growth
   //-------------------------------------------------------

   if ((pbTrap >= pbBase) && (pbTrap < pbBase+NUM_PAGES
      *SZ_PAGE))
   {

      if (!bGuardUp)
      {
       //-----------------------------------------------------
         // Unguard guard page
       //-----------------------------------------------------
         arReturn = DosSetMem(pbTrap,
                              SZ_PAGE,
                              PAG_READ|PAG_WRITE);

         printf("DosSetMem returns %ld "
            "( pbTrap = 0x%08lX ) \n",
                arReturn,
                pbTrap);
       //-----------------------------------------------------
         // Commit and guard next page below
       //-----------------------------------------------------

         printf("Going down!\n");
         pbTrap -= SZ_PAGE;

         if ((pbTrap >= pbBase) && (pbTrap < pbBase+NUM_PAGES
            *SZ_PAGE))
         {
            arReturn = DosSetMem(pbTrap,
                                 SZ_PAGE,
                                 PAG_COMMIT|PAG_READ|PAG_WRITE
                                    |PAG_GUARD);

            printf("DosSetMem returns %ld "
               "( pbTrap = 0x%08lX ) \n",
                   arReturn,
                   pbTrap);
         }                            /* endif                */
       //-----------------------------------------------------
         // We can continue execution
       //-----------------------------------------------------
         ulReturn = XCPT_CONTINUE_EXECUTION;
      }
      else
      {
       //-----------------------------------------------------
         // Unguard guard page
       //-----------------------------------------------------
         arReturn = DosSetMem(pbTrap,
                              SZ_PAGE,
                              PAG_READ|PAG_WRITE);

         printf
            ("DosSetMem returns %ld ( pbTrap = 0x%08lX ) \n",
             arReturn,
             pbTrap);

         printf("Going up!\n");
         pbTrap += SZ_PAGE;
```

```
          //-----------------------------------------------------
            // Commit and guard next page above
          //-----------------------------------------------------

            if ((pbTrap >= pbBase) && (pbTrap < pbBase+NUM_PAGES
               *SZ_PAGE))
            {
               arReturn = DosSetMem(pbTrap,
                                    SZ_PAGE,
                                    PAG_COMMIT|PAG_READ|PAG_WRITE
                                       |PAG_GUARD);

               printf("DosSetMem returns %ld "
                  "( pbTrap = 0x%08lX ) \n",
                      arReturn,
                      pbTrap);
            }                              /* endif                */
          //-----------------------------------------------------
            // We can continue execution
          //-----------------------------------------------------
            ulReturn = XCPT_CONTINUE_EXECUTION;
         }                                 /* endif                */
      }                                    /* endif                */
   }                                       /* endif                */
   return  ulReturn;
}
```

**GP.MAK**

```
GP.EXE:                          GP.OBJ
        LINK386 @<<
GP
GP
GP
OS2386
GP
<<

GP.OBJ:                          GP.C
        ICC -C+ -Kb+ -Ss+ GP.C
```

**GP.DEF**

```
NAME GP WINDOWCOMPAT

DESCRIPTION 'Exception handler example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

When an exception occurs, information about the exception is placed in the EXCEPTIONREPORTRECORD structure, and a pointer to these structures is passed to the exception handler.

```
struct _EXCEPTIONREPORTRECORD
{
  ULONG ExceptionNum;
  ULONG fHandlerFlags;
  struct  _EXCEPTIONREPORTRECORD    *NestedExceptionReportRecord;
  PVOID ExceptionAddress;
  ULONG cParameters;
  ULONG ExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS];
};
```

*ExceptionNum* is the field that tells the type of exception that has occurred. In our case, we're looking for a XCPT_GUARD_PAGE_VIOLATION. If the exception is not a guard page, we pass it on through to the system exception handler by returning XCPT_CONTINUE_SEARCH. If a guard-page exception occurs, we check to see if we have enough memory to commit one more page. If the memory is available, we commit another page and set it as a guard page. The last thing we do is return XCPT_CONTINUE_EXECUTION, which tells the system to bypass the other exception handler and continue executing the program. The errant function statement will execute correctly, and the program functions as if no problems had occurred.

## Summary

Exception handlers are a flexible way to give the developer control over system errors. Exception handlers have a lot of restrictions because the process can be dying when the exception handler is executed. However, with the right amount of prudence, an exception handler provides a powerful tool for error control.

# Chapter 8

# Interfacing with OS/2 Devices

The current OS/2 architecture supports three types of device drivers:

- Virtual device drivers (VDD)
- Physical device drivers (PDD)
- Presentation drivers (PD)

VDDs are used primarily by the legacy DOS and Windows applications. The virtualization of the physical devices provides OS/2 with the ability to control the access to these devices through the Virtual Device Driver. An example of a VDD is a VMOUSE.SYS or a VCDROM.SYS. The first one provides the virtual support for the mouse pointer requirements, while the latter one makes sure the CD ROM interfaces for the DOS and Windows applications are supported correctly.

The PD concerns itself mainly with OS/2's Presentation Manager support. PDs usually run at Ring 2 or Ring 3, and enable the Presentation Manager (PM) APIs to perform all of the necessary video functions. These include all aspects of the PM windowing, messaging, and controlling requirements.

The PDDs provide the OS/2 user with the actual access to the standard I/O devices. A PDD usually has a corresponding VDD, which allows the same functionality for the DOS and Windows legacy applications. The PDDs and VDDs are loaded at system startup and remain loaded for the entire duration of an OS/2 session. PDD architecture also provides OS/2 the flexibility to add nonstandard device support just by loading the appropriate device driver at startup time. There are two kinds of PDDs: block device drivers and character device drivers.

A SCSI (Small Computer systems Interface) driver is a type of block device driver. This driver manipulates the data in blocks of a certain size, and is referred to by the system via a drive letter. A good example of a PDD is the serial I/O device driver. But many character and block device drivers make up the device driver suite for OS/2.

This chapter offers two examples of how to talk to the serial devices under OS/2's control. The first example utilizes the preferred device driver interface *DosDevIOCtl*(), while the second shows how to get to the I/O ports without having to talk to the device driver.

There are obvious advantages for using the device driver interface:

1. Serialization/synchronization controls are built into the driver.
2. All OS/2 device drivers are interrupt driven.

3. It provides a well-defined interface for upward OS/2 migration.
4. Devices can be shared by multiple users.

Generally, the OS/2 applications gain access to the devices through the IOCTL interface, while the DOS applications can perform the same I/O functions that are allowed under real DOS (not VDM). Only 16-bit OS/2 code can run at Ring 2 privilege level, which allows the code direct I/O access (IOPL — means I/O Privilege Level). Occasionally it is advantageous to use the IOPL code to perform a quick read or write from or to a particular I/O port, but it is *not* the preferred OS/2 method. For example, if an application is monitoring room temperature and displays it on the screen, writing a full-blown device driver to access a particular I/O port on some adapter just to read two bytes of data may not make sense. In this case it is easier to utilize a 16-bit I/O code segment to perform an IN (Input from Port) instruction and read the temperature data. Synchronization and serialization do not have to be worried about. On the other hand, if the program reads the temperature and then decides to adjust the environmental conditions, a device driver must provide serialization and locking controls.

## Serial Interface Example Using *DosDevIOCtl*

The first of the two serial I/O examples deals with reading the data from the keyboard and transmitting all of the keystrokes to the 0x3F8 I/O port (COM1).

In order to gain access to the COM1, DEVICE=COM.SYS must be executed correctly at system startup and COM.SYS must be loaded. Next, a DosOpen call is issued to the device driver with "$COM1" as the filename. The system is smart enough to recognize the fact that the user is looking to gain access to the COM1 I/O port; if no other program is using the device, the file handle for the COM1 device is granted. Using this file handle, the process can now issue any DosDevIOCtl call with the appropriate asynchronous parameters to gain access to the control functions of the NS 8250/16450/16550 UARTs. Issuing *DosRead* and *DosWrite* requests to the system using the same file handle results in the data being transferred between the application buffers and the hardware UART.

The program uses the main thread to perform all of the keyboard read functions. The characters read are transmitted immediately to the COM1 I/O Port via *DosWrite* function. However, a separate thread is used to read the data from COM1 and display it on the screen. Since the device driver is capable of processing both the read and the write requests simultaneously, a better-designed communications program will dedicate a thread for each major function, such as read or write.

**32_TERM.C**

```
#define  INCL_OS2
#define  INCL_KBD
#define  INCL_VIO
#define  INCL_DOSPROCESS
#define  INCL_DOSDEVICES
#define  INCL_DOSDEVIOCTL
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define  STACK_SIZE     8192
#define  BPS            9600
#define  KBD_HANDLE     0
#define  VIO_HANDLE     0
struct
{
   BYTE             dataBits;

   BYTE             parity;
```

```
   BYTE              stopBits;
} lineCtrl =
{
   8,
   0,
   0
} ;
                                        // 8,N,1

DCBINFO           dcbInfo;

HFILE             hCom;                 // COM handle
unsigned char     inBuffer[256];        // input buffer

void ComThread(void);

/*************************************************************/
/*  main                                                     */
/*************************************************************/

int main(void)
{

   APIRET            ulAction,rc = 0;
   ULONG             ulBaudRate = BPS;
   ULONG             ulParmLen = 0;
   ULONG             ulBytesWritten;
   TID               ComThreadId = 0;
   ULONG             ulKbdChar = 0;

   printf("\n\n Each keystroke is echoed to COM1, 9600,8,N,1");
   printf("\n Ctrl-C or Ctrl-Brk to exit...\n\n");

/*Open and initialize COM1                                   */

   if (DosOpen((PUCHAR)"COM1",
               &hCom,
               &ulAction,
               0L,
               0,
               1,
               0x12,
               0L))
   {
      printf("COM1 not available or COM0x.SYS not loaded\n");
      exit(1);
   }

/*Set data rate to 9600bps and line format to N81            */

   ulParmLen = sizeof(ulBaudRate);

   rc = DosDevIOCtl(hCom,
                    IOCTL_ASYNC,
                    ASYNC_SETBAUDRATE,
                    &ulBaudRate,
                    ulParmLen,
                    &ulParmLen,
                    0,
                    0,
                    0);

   ulParmLen = sizeof(lineCtrl);
```

```
   rc = DosDevIOCtl(hCom,
                    IOCTL_ASYNC,
                    ASYNC_SETLINECTRL,
                    &lineCtrl,
                    ulParmLen,
                    &ulParmLen,
                    0,
                    0,
                    0);

/*Set device control block parameters                              */

   ulParmLen = sizeof(DCBINFO);

   rc = DosDevIOCtl(hCom,
                    IOCTL_ASYNC,
                    ASYNC_GETDCBINFO,
                    0,
                    0,
                    0,
                    &dcbInfo,
                    ulParmLen,
                    &ulParmLen);

   dcbInfo.usWriteTimeout = 6000;

   /*************************************************************/
   /* 60 second write timeout                                   */
   /*************************************************************/

   dcbInfo.usReadTimeout = 6000;

   /*************************************************************/
   /* 60 second readtimeout                                     */
   /*************************************************************/

   dcbInfo.fbCtlHndShake = MODE_DTR_CONTROL;

   /*************************************************************/
   /* enable DTR                                                */
   /*************************************************************/

   dcbInfo.fbFlowReplace = MODE_RTS_CONTROL;

   /*************************************************************/
   /* enable RTS                                                */
   /*************************************************************/

   dcbInfo.fbTimeout = MODE_WAIT_READ_TIMEOUT;

   /*************************************************************/
   /* wait-for-something reads                                  */
   /*************************************************************/

   ulParmLen = sizeof(DCBINFO);

   rc = DosDevIOCtl(hCom,
                    IOCTL_ASYNC,
                    ASYNC_GETDCBINFO,
                    &dcbInfo,
                    ulParmLen,
                    &ulParmLen,
                    0,
                    0,
                    0);
```

```
/*Create a thread to monitor the serial port                    */

   rc = DosCreateThread(&ComThreadId,
                        (PFNTHREAD)&ComThread,
                        0,
                        CREATE_READY,
                        STACK_SIZE);

/*Monitor the keyboard and output typed characters
Hit Ctrl-C to exit (primitive termination)                      */

   while (!rc)
   {
      if (kbhit())
      {
         ulKbdChar = (ULONG)getche();
         rc = DosWrite(hCom,
                       &ulKbdChar,
                       1,
                       &ulBytesWritten);
      }
   }

   printf(
    "\n\n Could not write to COM1, killing the MAIN thread.\n\n")
      ;

   return (rc);
}

/*************************************************************/
/*  Thread to read characters from COM1 and write to screen    */
/*************************************************************/

void ComThread(void)
{

   ULONG            ulBytesRead = 0,i;
   APIRET           rc = 0;

   while (!rc)
   {
      rc = DosRead(hCom,
                   inBuffer,
                   1,
                   &ulBytesRead);

      if (ulBytesRead)
      {
         for (i = 0; i < ulBytesRead; i++)
            inBuffer[i] &= 0x7f;
         VioWrtTTY(inBuffer,
                   ulBytesRead,
                   VIO_HANDLE);
      }
   }
   printf("\n\n Could not read from COM1");
   printf("killing the LISTEN thread.\n\n");


}
```

**32_TERM.MAK**

```
ALL:    32_TERM.EXE

32_TERM.EXE:                          32_TERM.OBJ
        LINK386 /NOI @<<
32_TERM
32_TERM
32_TERM
OS2386
32_TERM
<<


32_TERM.OBJ:                          32_TERM.C
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+ 32_TERM.C
```

**32_TERM.DEF**

```
NAME 32_TERM WINDOWCOMPAT
DESCRIPTION '32_TERM example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

The COM.SYS expects the following to be true:

COM1 Must reside at 0x3F8 and use the interrupt level 4.
COM2 Must reside at 0x2F8 and use the interrupt level 3.

The COM.SYS driver provides support for the UART control functions and the RS232C interface only. No specific devices are supported directly by the COM.SYS driver. It is left up to the applications to create subsystems or standalone programs to support the RS232C devices (modems and the like). The COM.SYS is a fully interrupt driven driver and has support for extended hardware buffering that is offered by the NS 16550 UARTs.

The PDD utilizes a memory buffer between the operating system and the UARTs, and data is copied in and out of the buffer from and to the UART transmit/receive registers. Once the user has obtained the file handle for a particular I/O port (COM1, COM2, etc.), he or she can use this handle to issue *DosRead* and *DosWrite* requests to move the data between an application and an I/O port. Currently, the system maintains a 1,024-byte receive and a 128-byte transmit buffer for the COM1-COM4 I/O ports when the driver is in the non-DMA mode. When the driver is in the enhanced DMA mode, there are two 1,024-byte receive queues and one 255-byte transmit queue. OS/2 does not guarantee that the sizes will remain constant with each version of the operating system, and thus the sizes are subject to change. The operating system also does not guarantee packet delivery to the device drivers in the same order that they were issued by the application due to the multitasking nature of OS/2.

## Serial Interface Example Using *inp*

The second example is much simpler than the first. As was mentioned before, only 16-bit code is allowed to execute with IOPL flag enabled. Taking this into consideration we can create a very handy 16-bit DLL like 16BITIO.DLL that exports the *inp*(), *inpw*(), *outp*(), and *outpw*() calls. Any 32-bit application can link with the import 16BITIO.LIB library and allow direct I/O functionality. This particular example uses a

very simple algorithm to check for the presence of an NS 16550 UART by issuing a series of *inp*() and *outp*() calls to the particular COM1 and COM2 I/O port ranges.

**CHK16550.C**

```
/*  Assume

       COM1 -> 0x3F8
       COM2 -> 0x2F8

 One attempts to first clear the 16550 FIFO by writing a 0x00
 to the FIFO Control register at offset 0x02. Then one attempts
 to enable the FIFOs by setting bit0 of the FIFO Control
 register at offset 0x02.  Reading the Interrupt Identification
 register at offset 0x02 will tell one if 16550 is present.

*************************************************************
                                                            */

#include <stdio.h>
#include <stdlib.h>
#include "chk16550.h"
int main(void);

#define  BIT_6_7_SET   0x00C0
int main(void)

{
   unsigned         Byte = 0;

   printf("\n\n Attempting to find 16550 UART ...");/* test
                                          COM1              */
   my_outp(MY_COM1+MY_FIFO_CTRL,
           0x00);                  /* Clear the FIFO reg   */
   Byte = my_inp(MY_COM1+MY_INT_ID);
   Byte &= BIT_6_7_SET;
   if (!Byte)
   {

      my_outp(MY_COM1+MY_FIFO_CTRL,
              0x01);               /* Set the FIFO reg     */
      if (my_inp(MY_COM1+MY_INT_ID)&BIT_6_7_SET)
         printf(
           "\n\n 16550 appears to be present for COM1->0x3F8.\n")
           ;
      else
         printf(
            "\n\n 16550 appears to be absent for COM1->0x3F8.\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM1->0x3F8. Exiting ... \n\n");
      return (-1);
   }                               /* test COM2 loop? :)   */
   my_outp(MY_COM2+MY_FIFO_CTRL,
           0x00);                  /* Clear the FIFO reg   */
   if (!(Byte = (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)))
   {

      my_outp(MY_COM2+MY_FIFO_CTRL,
              0x01);               /* Set the FIFO reg     */
```

```
      if (my_inp(MY_COM2+MY_INT_ID)&BIT_6_7_SET)
         printf(
           "\n 16550 appears to be present for COM2->0x2F8.\n\n")
           ;
      else
         printf(
            "\n 16550 appears to be absent for COM2->0x2F8.\n\n")
            ;

   }
   else
   {
      printf(
         "\n\n Unknown error for COM2->0x2F8. Exiting ... \n\n");
      return (-1);
   }

   return (0);
}
```

**CHK16550.H**

```
/* Header file for the 16-bit LIB/DLL used to perform IOPL i/o calls */
/* A. Panov 1993,1994,1995                                           */


extern unsigned short _Far16 _Cdecl my_inp  (unsigned short);
extern unsigned short _Far16 _Cdecl my_outp (unsigned short, unsigned short);

#define MY_COM1          0x3F8
#define MY_COM2          0x2F8
#define MY_INT_ENABLE    1
#define MY_INT_ID        2
#define MY_FIFO_CTRL     2
#define MY_LINE_CTRL     3
#define MY_MODEM_CTRL    4
#define MY_LINE_STATUS   5
#define MY_MODEM_STATUS  6
#define MY_SCRATCH       7
```

**CHK16550.MAK**

```
ALL:    CHK16550.EXE

CHK16550.EXE:                          CHK16550.OBJ
        LINK386 /NOI @<<
CHK16550
CHK16550
CHK16550
OS2386+16bitio
CHK16550
<<


CHK16550.OBJ:                          CHK16550.C
        ICC -C+ -Gm+ -Kb+ -Sm -Ss+  CHK16550.C
```

**CHK16550.DEF**

```
NAME CHK16550 WINDOWCOMPAT
DESCRIPTION 'CHK16550 example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
STACKSIZE 16384
```

**16BITIO.C**

```
/* 16-bit I/O dll                                                    */

_acrtused = 0;

#include <conio.h>
int far _cdecl my_inp(unsigned);

int far _cdecl my_outp(unsigned,unsigned);
unsigned far _cdecl my_inpw(unsigned);
unsigned far _cdecl my_outpw(unsigned,unsigned);

int far _cdecl my_inp(unsigned usPort)
{
   return (inp(usPort));
}

int far _cdecl my_outp(unsigned usPort,unsigned usValue)
{
   return (outp(usPort,
                usValue));
}

unsigned far _cdecl my_inpw(unsigned usPort)
{
   return (inpw(usPort));
}

unsigned far _cdecl my_outpw(unsigned usPort,unsigned usValue)
{
   return (outpw(usPort,
                 usValue));
}
```

**16BITIO.MAK**

```
ALL:    16BITIO.DLL 16BITIO.LIB

16BITIO.LIB:                        16BITIO.DLL
        IMPLIB 16BITIO.LIB 16BITIO.DEF

16BITIO.DLL:                        16BITIO.OBJ
        LINK /NOI @<<
16BITIO
16BITIO.DLL
16BITIO

16BITIO
<<


16BITIO.OBJ:                  16BITIO.C
        cl  -c -AL -G2s -Fc 16BITIO.C
```

**16BITIO.DEF**

```
LIBRARY INITINSTANCE
PROTMODE
DESCRIPTION '16bitIO example
            Copyright (c) 1992-1995 by Arthur Panov.
            All rights reserved.'
DATA NONSHARED
SEGMENTS _IOSEG CLASS 'IOSEG_CODE' IOPL
EXPORTS
       _my_inp   1
       _my_outp  2
       _my_inpw  1
       _my_outpw 2
STACKSIZE 4096
```

The ASYNC PDD is covered in much greater detail in the *IBM Physical Device Driver Reference* manual (10G6266), which is part of the OS/2 Toolkit Technical Library.

# Chapter 9

# Introduction to Windows

## Introduction

The basic building block for all Presentation Manager (PM) programming is a window. Most items displayed on the screen are windows, of some shape or fashion. A window is designed to react to messages sent to it either from the system or from another window. These messages are placed into a message queue that is unique to each PM application. A message is used to signal events that happen to a window. For example, a WM_CREATE message is sent when a window is halfway through its creation process; a WM_SIZE message is sent after the user has sized the window; a WM_DESTROY message just before the destruction of the window is complete. Each window has a specific window procedure that is used to respond back to the system when a message is sent. The programmer is responsible for creating this window procedure. The window procedure is a switch statement that will filter out certain messages that are of interest to the application. The messages that are not interesting can be passed on to a default window procedure or a default dialog procedure. For instance, the programmer may want to initialize some data in the WM_CREATE message processing or free up memory when the WM_DESTROY is received.

## What Is a Window?

The first thing to understand when beginning Presentation Manager programming is the concept of a window. A window is a graphical image of a rectangle that sits on the screen and is used to provide a uniform interface with which a user can interact. (See Figure 9.1.)

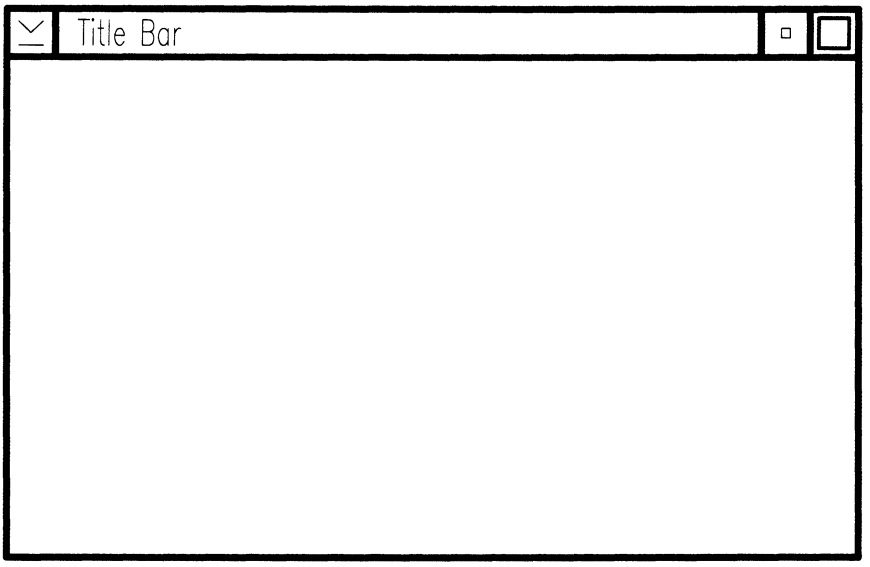



Figure 9.1 A window.

A window can be sized larger or smaller, it can be opened or closed, it can be made visible or invisible. Suffice it to say that there are a lot of things to do with a window.

Figure 9.2 looks like *one* window but, in reality, it is *five* windows:

- The frame window
- The title bar
- The system menu
- The maximize/minimize buttons
- The client window

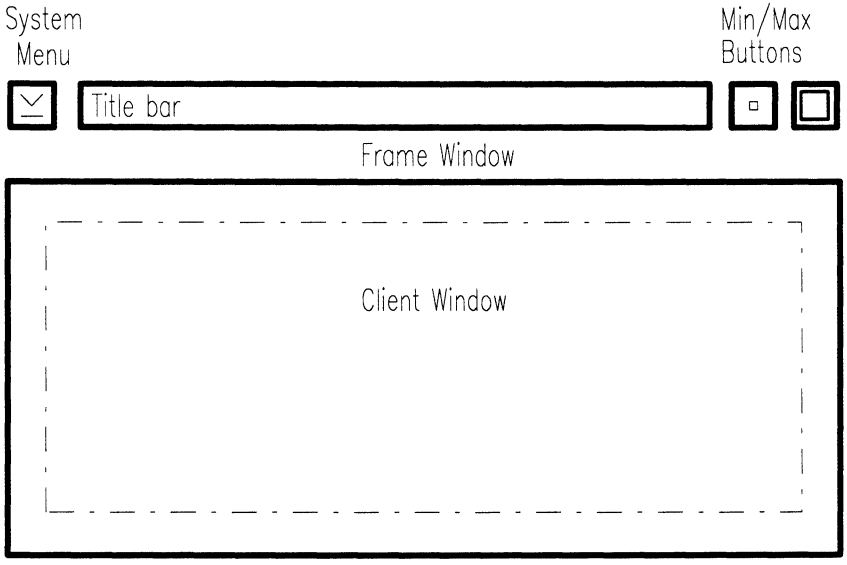



Figure 9.2 Drawing of a window's components.

Each of the five windows has a window procedure associated with it. In most cases, the programmer will be able to use the system-defined window procedures for all but the client window. The window procedure is a function that tells the window how to behave. Windows that share the same window procedure belong to the same window *class*. This is a familiar concept for those readers acquainted with object-oriented programming.

Imagine a fast food restaurant. Each item on the menu could be considered one class—a hot dog class, a hamburger class, and a pizza class. Suppose mustard, mayo, relish, or cheese could be put on a hot dog, in any combination. Each of these condiments would be a hot dog *style*.

The same is true for window classes. There are many predefined window classes, including some classes specific to pen computing and the multimedia extensions. The classes specific to Presentation Manager are:

| | |
|---|---|
| WC_FRAME | Frame control class |
| WC_COMBOBOX | Combo box control class |
| WC_BUTTON | Button control class |
| WC_MENU | Menu control class |
| WC_STATIC | Static text control class |
| WC_ENTRYFIELD | Entryfield control class |
| WC_LISTBOX | Listbox control class |

| | |
|---|---|
| WC_SCROLLBAR | Scrollbar control class |
| WC_TITLEBAR | Titlebar control class |
| WC_MLE | Multi-line edit control class |
| WC_SPINBUTTON | Spinbutton control class |
| WC_CONTAINER | Container control class |
| WC_SLIDER | Slider control class |
| WC_VALUESET | Valueset control class |
| WC_NOTEBOOK | Notebook control class |

Each window class is very different from the others. Some of these predefined classes will be covered in later chapters. The client window, which is the area inside the window frame, belongs to a user-defined class. Each window class also contains a set of window styles specific to that class. There is a set of class styles available to all classes. The styles are:

- CS_MOVENOTIFY
- CS_SIZEREDRAW
- CS_HITTEST
- CS_PUBLIC
- CS_FRAME
- CS_CLIPCHILDREN
- CS_CLIPSIBLINGS
- CS_PARENTCLIP
- CS_SAVEBITS
- CS_SYNCPAINT

These styles will be covered in more detail in the section entitled "Window Stylin'."

Once we know a little bit about the window classes the operating system offers, we can decide which are best suited for our application, or, as most of us do-it-yourselfers will do, you can create your own. So, let's do just that.

**WIN1.C**

```
#define INCL_WIN
#define INCL_GPI

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define CLS_CLIENT                "WindowClass"

MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
   ULONG ulMsg,
   MPARAM mpParm1,
   MPARAM mpParm2 ) ;

INT main ( VOID )
{
   HAB         habAnchor ;
   HMQ         hmqQueue ;
   ULONG       ulFlags ;
   HWND        hwndFrame ;
   HWND        hwndClient ;
   BOOL        bLoop ;
```

```
   QMSG        qmMsg ;

   habAnchor = WinInitialize ( 0 ) ;
   hmqQueue = WinCreateMsgQueue ( habAnchor, 0 ) ;

   WinRegisterClass ( habAnchor,
                      CLS_CLIENT,
                      ClientWndProc,
                      0,
                      0 ) ;

   ulFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER |
             FCF_MINMAX | FCF_SHELLPOSITION | FCF_TASKLIST ;

   hwndFrame = WinCreateStdWindow ( HWND_DESKTOP,
                                    WS_VISIBLE,
                                    &ulFlags,
                                    CLS_CLIENT,
                                    "Titlebar",
                                    0L,
                                    NULLHANDLE,
                                    0,
                                    &hwndClient ) ;

   if ( hwndFrame != NULLHANDLE ) {
      bLoop = WinGetMsg ( habAnchor,
                          &qmMsg,
                          NULLHANDLE,
                          0,
                          0 ) ;
      while ( bLoop ) {
         WinDispatchMsg ( habAnchor, &qmMsg ) ;
         bLoop = WinGetMsg ( habAnchor,
                             &qmMsg,
                             NULLHANDLE,
                             0,
                             0 ) ;
      } /* endwhile */

      WinDestroyWindow ( hwndFrame ) ;
   } /* endif */

   WinDestroyMsgQueue ( hmqQueue ) ;
   WinTerminate ( habAnchor ) ;
   return 0 ;
}

MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
                                 ULONG ulMsg,
                                 MPARAM mpParm1,
                                 MPARAM mpParm2 )
{
   switch ( ulMsg ) {
   case WM_ERASEBACKGROUND:
      return MRFROMSHORT ( TRUE ) ;

   default:
      return WinDefWindowProc ( hwndWnd,
                                ulMsg,
                                mpParm1,
                                mpParm2 ) ;
   } /* endswitch */

   return MRFROMSHORT ( FALSE ) ;
}
```

**WIN1.MAK**

```
WIN1.EXE:                       WIN1.OBJ
        LINK386 @<<
WIN1
WIN1
WIN1
OS2386
WIN1
<<

WIN1.OBJ:                       WIN1.C
        ICC -C+ -Kb+ -Ss+ WIN1.C
```

**WIN1.DEF**

```
NAME WIN1 WINDOWAPI

DESCRIPTION 'Simple window example
Copyright (c) 1992 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

# The INCLUDE Files

The OS/2 Toolkit provides oodles and oodles of header files. These files contain structure definitions, function prototypes, and many system-defined constants to make OS/2 programs much easier to read. The large size of these files and the tremendous amount of overhead they create make it advantageous to selectively pick and choose those parts that are applicable to a program. This is done by placing a series of #defines before the inclusion of OS2.H. In this program, we will use #define INCL_WIN.

```
#define INCL_WIN
#include <os2.h>
```

This is an all-encompassing define that will include the necessary headers for all the Win... functions. This is overkill in most cases, but for our first example we'll keep things simple.

# The Window Procedure Definition

```
MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
                                 ULONG ulMsg,
                                 MPARAM mpParm1,
                                 MPARAM mpParm2 ) ;
```

Window procedures are declared in a very special way, using the prefix MRESULT EXPENTRY. In OS2DEF.H, these expand to VOID * _System. The return type, MRESULT, gives the window procedure the freedom to return whatever it needs to by using the VOID * type. The _System tells the C-Set/2 compiler that the operating system will be calling the function. It is a good idea to use the Presentation Manager–defined data types when dealing with window procedures and messages. There is a good probability that some definitions will change when moving to other machine architectures, and by using the defined data types, we save some headaches if we need to port the application to some other version of OS/2. A more detailed explanation of window procedure is in the section "The Window Procedure Revisited."

The function's parameters are HWND *hwndWnd*, ULONG *msg*, MPARAM *mpParm1*, and MPARAM *mpParm2*. This may look very familiar to Microsoft Windows programmers. The variable *hwndWnd* is a

window handle. Each window has its own unique window handle, and most *Win...* functions will include this as a parameter. In this case, *hwndWnd* is the window to which the message is being sent. The parameter *ulMsg* is the specific message being sent to the window. We will cover messages in more detail in Chapter 11.

The last two parameters are *mpParm1* and *mpParm2,* which have the type MPARAM. These are "shape-shifter" parameters. MPARAM is really a PVOID in disguise. This gives the operating system two 32-bit spaces to insert whatever data corresponds to the message being sent. These values could be pointers or short or long integers. For example, the message WM_MOUSEMOVE is sent whenever the mouse is moved. The first message parameter, *mpParm1*, would contain two SHORTs. The second message parameter, *mpParm2*, also contains two SHORTs. Figure 9.3 provides a breakdown of a message-parameter variable.

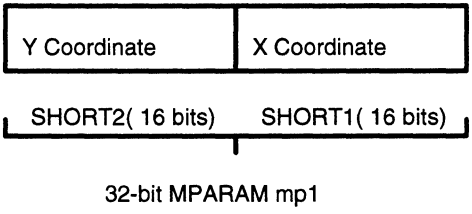



Figure 9.3 Breakdown of a message-parameter variable.

## Helper Macros

Many data-type conversions are necessary in a Presentation Manager application because of the multiple data types that can be used as an MPARAM or MRESULT. MRESULT is the value returned by the window procedure and is also a "shape-shifter." The Toolkit includes a group of helper macros to make these conversions easier.

Table 9.1 presents the macros used to convert some standard data type into a MPARAM data type that can be used when sending or posting a window message.

Table 9.1 Macros to Convert into MPARAM

| Macro | Converts into MPARAM |
|---|---|
| MPFROMVOID | 0 |
| MPFROMP | PVOID |
| MPFROMHWND | HWND |
| MPFROMCHAR | CHAR |
| MPFROMSHORT | SHORT |
| MPFROM2SHORT | 2 SHORTs |
| MPFROMSH2CH | 2 CHARs |
| MPFROMLONG | ULONG |

Table 9.2 presents the macros used to convert a MPARAM data type into a standard data type that can be used when receiving a window message.

Table 9.2 Macros to Convert from MPARAM

| Macro | Converts from MPARAM |
|---|---|
| PVOIDFROMMP | PVOID |
| HWNDFROMMP | HWND |
| CHAR1FROMMP | CHAR |
| CHAR2FROMMP | second CHAR |
| CHAR3FROMMP | third CHAR |
| CHAR4FROMMP | fourth CHAR |
| SHORT1FROMMP | low SHORT |
| SHORT2FROMMP | high SHORT |
| LONGFROMMP | ULONG |

Table 9.3 presents the macros used to convert a MRESULT data type into a standard data type that can be used to examine a return value for the window procedure.

Table 9.3 Macros to Convert from MRESULT

| Macro | Converts from MRESULT |
|---|---|
| PVOIDFROMMR | PVOID |
| SHORT1FROMMR | low SHORT |
| SHORT2FROMMR | high SHORT |
| LONGFROMMR | ULONG |

Table 9.4 presents the macros used to convert a standard data type into a MRESULT data type that can be used to construct a return value from the window procedure.

Table 9.4 Macros to Convert to MRESULT

| Macro | Converts to MRESULT |
|---|---|
| MRFROMP | PVOID |
| MRFROMSHORT | SHORT |
| MRFROM2SHORT | 2 SHORTs |
| MRFROMLONG | ULONG |

# Presentation Manager Program Initialization

```
habAnchor = WinInitialize ( 0 ) ;
hmqQueue = WinCreateMsgQueue ( habAnchor, 0 ) ;
```

The beginning of a PM program will always start with a few things. First, *WinInitialize* is called to obtain an anchor block handle, or HAB. An anchor block is specific to each thread that contains a window procedure.

```
HAB WinInitialize( ULONG flOptions)
```

The only parameter for *WinInitialize* is a ULONG that is used for initialization options. In a PM environment, this should be 0. An anchor block currently contains error information for each thread and also may be used for "future portability issues." Each Presentation Manager thread should obtain its own anchor block for two reasons: portability and also to obtain error information specific to that thread.

```
HMQ WinCreateMsgQueue( HAB hab, LONG lQueuesize)
```

*WinCreateMsgQueue* will create a message queue for the thread that called the function. The message queue is how Presentation Manager communicates back and forth with the windows. The first parameter is the anchor block handle, *habAnchor*. The second parameter is the queue size. A parameter of 0 indicates the default queue size in OS/2, which holds 10 messages. A full queue will cause the user interface to respond rather slowly and sometimes to stop responding completely. The default queue size should be fine for most applications. If a queue is getting too full, the program should be checked to see where messages are getting backlogged. (One of the requirements for a PM interface is a crisp user response. Any response that consumes more than 100 milliseconds probably should be put in a separate thread. See Chapter 30 for more information on multithreading in a PM program.)

# Creating a New Class

```
WinRegisterClass ( habAnchor,
                   CLS_CLIENT,
                   ClientWndProc,
                   0,
                   0 ) ;
```

The function *WinRegisterClass* is used to create a new class of windows, in this case CLS_CLIENT.

```
BOOL WinRegisterClass(  HAB hab,
                        PSZ pszClassName,
                        PFNWP pfnWndProc,
                        ULONG flStyle,
                        ULONG cbWindowData)
```

The first parameter is the anchor block, *habAnchor*. The next parameter is the class name. This parameter is a null-terminated string. The next parameter is the window procedure the class is assigned to, *ClientWndProc*. The fourth parameter is the class styles used for the new class. We're not going to use any class styles for now, so we put 0 here. The last parameter is the number of bytes of storage space that will be tacked on to each window belonging to this class. This piece of space is commonly referred to as "window words." This is covered in more detail later.

# Creating a Window

By now readers are probably thinking "But I just wanted to create one lousy window." Well, this is it, the function call you've been waiting for: *WinCreateStdWindow*. This function actually creates five windows as stated earlier; but only two that are of any interest to us—the frame window and the client window.

```
ulFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER |
          FCF_MINMAX | FCF_SHELLPOSITION | FCF_TASKLIST ;

hwndFrame = WinCreateStdWindow ( HWND_DESKTOP,
                                 WS_VISIBLE,
                                 &ulFlags,
                                 CLS_CLIENT,
                                 "Titlebar",
                                 0L,
                                 NULLHANDLE,
                                 0,
                                 &hwndClient ) ;
```

The function returns the frame window handle.

```
HWND WinCreateStdWindow(      HWND hwndParent,
                              ULONG flStyle,
                              PULONG pflCreateFlags,
                              PSZ pszClassClient,
                              PSZ pszTitle,
                              ULONG flStyleClient,
                              HMODULE Resource,
                              ULONG ulID,
                              PHWND phwndClient)
```

The first parameter specified is the parent of the frame window. We'll discuss parents and owners in a minute. The second parameter is the frame style. A frame can draw from two sets of styles: frame styles, because this is a frame window; and window styles, because the frame class is a subset of the window class "window." The most common window style available is WS_VISIBLE. Yep, you guessed it, this means the window is not only created but will show up as well.

The third parameter is the frame flags. Frame flags describe how the frame will look. The possible descriptors are OR'ed together. Figure 9.4 is a diagram of all the possible descriptors and the bits that correspond to them.

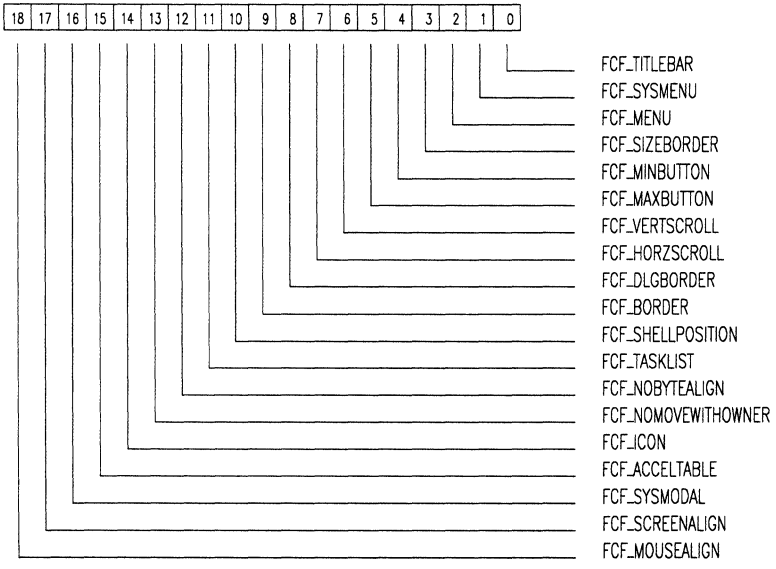



Figure 9.4 Frame creation flags.

Table 9.5 Frame Creation Flags Description

| Flag | Description |
|---|---|
| FCF_TITLEBAR | Creates a title bar on the frame. |
| FCF_SYSMENU | Creates a system menu on the frame. |
| FCF_MENU | Creates an application menu on the frame. This is loaded from the resource file or .DLL. (See Chapter 12 for more information.) |
| FCF_MINBUTTON | Creates a minimize button on the frame. |
| FCF_SIZEBORDER | Creates a sizing border on the frame. |
| FCF_MAXBUTTON | Creates a maximize button on the frame. |

| Flag | Description |
|---|---|
| FCF_MINMAX | Creates both a minimize and maximize button on the frame. |
| FCF_HORZSCROLL | Creates a horizontal scroll bar on the frame. |
| FCF_DLGBORDER | Creates the thick dialog box border on the frame. |
| FCF_VERTSCROLL | Creates a vertical scroll bar on the frame. |
| FCF_BORDER | Creates a thin border on the frame. |
| FCF_SHELLPOSITION | The system determines the initial size and placement of the frame window. |
| FCF_TASKLIST | Adds the program title to the task list and window title to the window list. |
| FCF_NOBYTEALIGN | Do not optimize window movements in 8 pel multiples. |
| FCF_NOMOVEWITHOWNER | The frame window will not move when the owner is moved. |
| FCF_ICON | An icon is added to the frame. This is loaded from the resource file or .DLL. (See Chapter 12 for more information.) |
| FCF_ACCELTABLE | An accelerator table is added to the frame. This is loaded from the resource file or .DLL. (See Chapter 12 for more information.) |
| FCF_SYSMODAL | The frame window is system modal. |
| FCF_SCREENALIGN | The frame window is positioned relative to the desktop rather than relative to the owner window. |
| FCF_MOUSEALIGN | The frame window is positioned relative to the position of the mouse rather than relative to the owner window. |
| FCF_STANDARD | FCF_TITLEBAR \| FCF_SYSMENU \| FCF_MINBUTTON \| FCF_MAXBUTTON \| FCF_SIZEBORDER \| FCF_ICON \| FCF_MENU \| FCF_ACCELTABLE \| FCF_SHELLPOSITION \| FCF_TASKLIST. |
| FCF_AUTOICON | A WM_PAINT message will not be sent to the application when the frame window is iconized. |
| FCF_HIDEBUTTON | Creates "hide" button on the frame. |
| FCF_HIDEMAX | Creates "hide" and maximize buttons on the frame. |

In this example, we'll use the following flags: FCF_TITLEBAR, FCF_SYSMENU, FCF_SIZEBORDER, FCF_TASKLIST, FCF_MINMAX, and FCF_SHELLPOSITION.

**Gotcha!**
Be sure to pass a pointer to a ULONG as this parameter.

The fourth parameter is the name of the window class that the client window will belong to; in this case, we use the string defined by CLS_CLIENT. The next parameter is the window text for the title bar. The sixth parameter is the client window style. Since we defined the parent of the client window *hwndFrame* to have the style WS_VISIBLE, the client, as a child of *hwndFrame*, will inherit the WS_VISIBLE style. This means we don't have to specify any window styles here; we'll just leave that a 0.

The next parameter is the resource ID location. The next parameter contains the resource ID for the frame window. This one resource ID will point to all the resources that are defined for the frame. This includes

the menu, icon, accelerator table, and any other items defined using the frame creation flags. For more information on resources, see Chapter 12.

The last parameter is the address of a window handle. Presentation Manager will place the client window handle into this variable upon the function's return.

If *WinCreateStdWindow* fails, NULLHANDLE is returned. Before we attempt to do anything else, it is a good idea to check the return handle to make sure it is valid; if not, the application should quit, preferably with some sort of error message.

## Message, Message, Who's Got the Message?

```
bLoop = WinGetMsg ( habAnchor,
                    &qmMsg,
                    NULLHANDLE,
                    0,
                    0 ) ;
while ( bLoop ) {
   WinDispatchMsg ( habAnchor, &qmMsg ) ;
   bLoop = WinGetMsg ( habAnchor,
                       &qmMsg,
                       NULLHANDLE,
                       0,
                       0 ) ;
} /* endwhile */
```

The two functions, *WinGetMsg* and *WinDispatchMsg*, are the keys to getting the message queue up and running. Without some form of message retrieval and dispatch, the system will respond with a "Program not responding..." error message. The secret to a well thought out Presentation Manager application is a message queue that is quick and responsive. *WinGetMsg* will retrieve the message from the message queue and place it into the variable *qmMsg*. The QMSG structure looks very similar to the variables that are passed to the window procedure. Eventually the QMSG structure will be passed on to *ClientWndProc* or to the window procedure for the window receiving the message. *WinGetMsg* and *WinDispatchMsg* form a post office for messages. They pick up the messages and then make sure that the messages are delivered to the correct window.

```
BOOL WinGetMsg(        HAB hab,
                       PQMSG pqmsgmsg,
                       HWND hwndFilter,
                       ULONG ulFirst,
                       ULONG ulLast )
```

The first parameter of *WinGetMsg* is the anchor block handle. The next one is the address of the QMSG structure that will handle the retrieved message information. The next three parameters are not used in this example. They provide a way for *WinGetMsg* to choose selectively which messages to pick out of the queue. By specifying zeroes here, *WinGetMsg* will retrieve all messages from the message queue in the order they were placed there. After the message is retrieved from the queue, it is then passed on to *WinDispatchMsg*.

```
MRESULT WinDispatchMsg(        HAB hab,
                               PQMSG pqmsgMsg )
```

It is *WinDispatchMsg*'s job to take the message from the *qmMsg* variable and send it on to the window procedure associated with the window it is addressed to. For instance, if *qmMsg.hwnd* were equal to *hwndWnd*, *WinDispatchMsg* would take *qmMsg* and send it on to *ClientWndProc*.

```
/* QMSG structure */
typedef struct _QMSG    /* qmsg */
{
   HWND    hwnd; /* window handle that msg is being sent to */
   ULONG   msg; /* the message itself */
   MPARAM  mp1; /* Message Parameter 1 */
   MPARAM  mp2; /* Message Parameter 2 */
   ULONG   time; /* Time msg was sent */
   POINTL  ptl; /* Mouse position when msg was sent */
   ULONG   reserved;
} QMSG;
typedef QMSG *PQMSG;
```

The QMSG structure contains a lot of very interesting information about the message. The first field in the structure, *hwnd*, is the window handle the message is for. The field *msg* is the constant identifying the message. Some common messages are WM_CREATE, WM_PAINT, WM_QUIT, and WM_SIZE. The next two parameters, *mp1* and *mp2*, are the message parameters. Each message has a set use for these parameters. Usually they are used to convey more information about the message. The *time* field contains the time the message was sent, and the *ptl* field is a structure that contains the mouse position when the message was sent.

## Terminating a Program

You may have noticed that *WinGetMsg* and *WinDispatchMsg* were running in a while loop. While *WinGetMsg* returns a TRUE value, this loop continues to process messages. When *WinGetMsg* receives a WM_QUIT, *WinGetMsg* returns FALSE and will fall out of the loop. At this point, the user has elected to close the application, and it's time for the final cleanup. We have created three things that need to be destroyed—the frame window *hwndFrame*, *hmqQueue*, and *habAnchor*. Each of these items has its own destroy function.

```
BOOL WinDestroyMsgQueue( HMQ hmq )
BOOL WinDestroyWindow( HWND hwnd )
BOOL WinTerminate( HAB hab )
```

By destroying *hwndFrame*, we also are destroying the client window, the title bar, and all the other windows that were children of the frame.

```
   WinDestroyWindow ( hwndFrame ) ;
} /* endif */

WinDestroyMsgQueue ( hmqQueue ) ;
WinTerminate ( habAnchor ) ;
return 0 ;
```

## The Window Procedure Revisited

You might have looked over *main* and thought, "Is this it?" Well, no. We've presented just the tip of the iceberg. The window procedure is the meat of a Presentation Manager program. A window procedure's sole purpose in life is to respond to the messages for the window that belongs to it. It is also important to

realize that multiple windows can and will access the same window procedure. Programmers must be very careful with static and global variables or flags. They can come back to haunt developers if two windows are accessing the same procedure. It is a good idea to avoid these if at all possible.

Most window procedures are nothing more than a giant *switch* statement, with a case for each message. A window procedure does not have to respond to every message; it can filter the majority of the messages through to a function, *WinDefWindowProc* or *WinDefDlgProc*. This function lets the system handle messages in a system default manner. As the creator of the window procedure, it is the programmer's job to pick out which messages will trigger a response in your program. For instance, when a WM_SIZE message is received, the programmer may wish to reflow any text on the window so that it is all visible and centered. Passing messages on to *WinDefWindowProc* or *WinDefDlgProc* is very safe.

**Gotcha!**
Be very careful about accidentally reversing *WinDefWindowProc* and *WinDefDlgProc*. Strange things can occur when calling *WinDefWindowProc* for a dialog box or using *WinDefDlgProc* for a non-dialog box window.

The default action for these messages is listed in the online reference for the Toolkit. A few messages are very important to a window procedure. These will be covered later in this chapter.

In this example the window procedure, *ClientWndProc,* is very small. It's not quite the smallest window procedure available, but it's pretty close.

```
MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
                                 ULONG ulMsg,
                                 MPARAM mpParm1,
                                 MPARAM mpParm2 )
{
   switch ( ulMsg ) {
   case WM_ERASEBACKGROUND:
      return MRFROMSHORT ( TRUE ) ;

   default:
      return WinDefWindowProc ( hwndWnd,
                                ulMsg,
                                mpParm1,
                                mpParm2 ) ;
   } /* endswitch */

   return MRFROMSHORT ( FALSE ) ;
}
```

The only message that is utilized in *ClientWndProc* is WM_ERASEBACKGROUND. This message is used to fill the client window with the system-window background color. If we let this message pass on to *WinDefWindowProc*, the background of the window would be transparent and the desktop would show through. By returning TRUE, we tell the system to paint the client window with the background color. In some cases, this message doesn't need to be processed if the painting is handled in the WM_PAINT message. In a window procedure, most messages have a default handling of returning FALSE. Programmers can save a few extra function calls by returning FALSE themselves from the handled instead of calling *WinDefWindowProc*.

## Parents and Owners

Earlier we had mentioned the concept of parents and owners. These terms are used often in Presentation Manager programming. It is important to understand each one. Every window has a parent, except for the desktop window. In some cases the parent will be the desktop, HWND_DESKTOP. In the last example, the frame window had the desktop as its parent. The frame window was the parent for the client window, the title bar window, and the other windows. What is a parent window?

A parent window performs many of the same duties that parents of human children perform. A parent window controls where the child can go. A child is "clipped" to the parent and will not be visible outside the parental boundaries. A child window can be moved outside these boundaries; however, the portion outside the parent window will not be visible. Also, a child will inherit all of the parent's styles. If a parent is visible, a child will be visible; if a parent is not visible, a child will not be visible. If a parent moves, the child moves along with it. However, unlike a human parent, if a parent window is destroyed, all of its children are destroyed as well. If a parent window has two child windows, these children are considered siblings. When a family of windows is all visible at the same time, there is a power struggle for which window will be displayed on top. A child window always will be on top of the parent window. Some surprise, huh? However, siblings, and the whole windowing system as well, use a concept known as "Z-Order" to decide who gets on top. The sibling created last usually is at the top of the "Z-Order." The programmer can change the order using the function *WinSetWindowPos*. This function lets a window be put on top or behind its other siblings. User interaction also affects the "Z-Order." When the user clicks on one of the siblings, that window will become the active window, and it will move to the top of the "Z-Order." The active window is usually the window that either is or owns the focus window. There is only one active window in the system at any given time.

The other type of window relationship is an owner window. In the last example, *hwndFrame* was also the owner of the other windows. An owner shares some of the same duties a parent shares. When an owner is hidden, destroyed, or minimized, the children are also. However, an owned window is not clipped to its owner.

The other interesting features of owners is the level of communication between owners and owned, or "control," windows. When an important event happens to an owned window, the owner is sent a WM_CONTROL message. The *mpParm1* and *mpParm2* parameters tell the owner which control sent the message and what kind of event has occurred. A window does not have to have an owner.

## Window Stylin'

When a window is created, various descriptors are used to describe how the window will look or act; these descriptors are known as window styles. There are many different kinds of styles, including window styles and class styles, and each type of control has its own styles as well. In this section we will concentrate on window styles, class styles, and frame styles. The other control styles will be covered in their respective chapters.

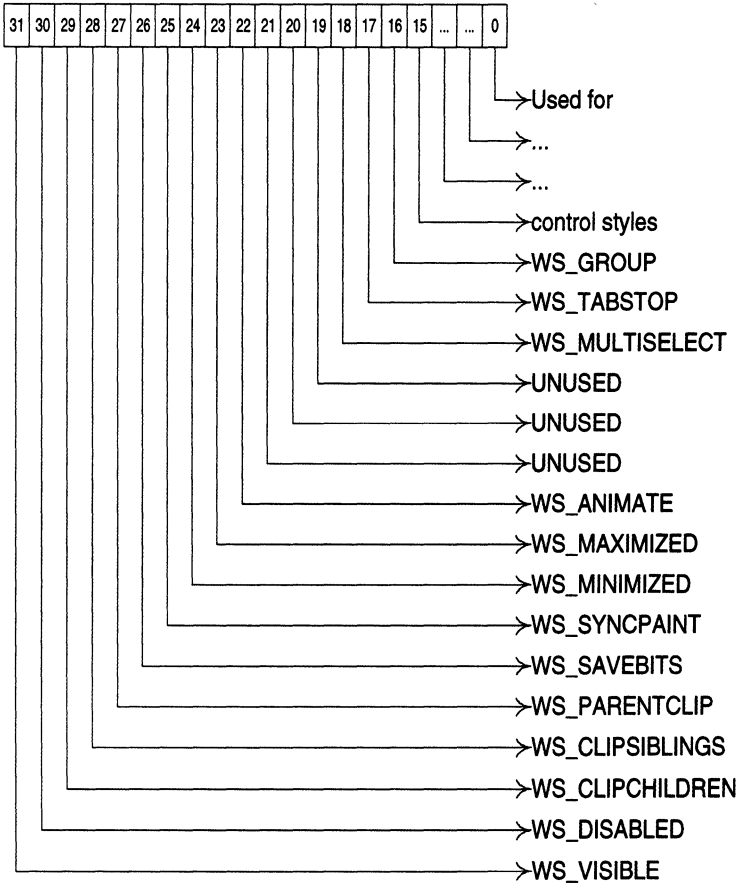



Figure 9.5 Window-style flags.

Figure 9.5 shows that the first 16 bits are used for the respective control window styles; the upper 16 bits are used for window styles. Since controls are also windows, both the control window styles and the basic window styles are designed to live together harmoniously.

Table 9.6 Window Style Descriptions

| Value | Description |
|---|---|
| WS_GROUP | Defines which items make up a group in a dialog box window. See Chapter 13. |
| WS_TABSTOP | The user can use the tab key to move to this dialog item. See Chapter 13. |
| WS_ANIMATE | Will create "exploding windows." |
| WS_MAXIMIZED | Causes a window to be created fully maximized. |
| WS_MINIMIZED | Causes a window to be created fully minimized. |
| WS_SYNCPAINT | Causes a window to have paint messages generated immediately when an area of the window needs to be repainted. |
| WS_SAVEBITS | Will save the screen area under a window and will restore the image when a covered area has been uncovered. |

| Value | Description |
|---|---|
| WS_PARENTCLIP | Will cause the parent's presentation space to be clipped to the child's presentation space, enabling the child to draw on the parent's presentation space. This can create some very interesting results, as the parent's visible presentation space usually is larger than or equal to the child's. Most often this style is not used. |
| WS_CLIPSIBLINGS | Will prevent siblings from redrawing on top of each other. |
| WS_CLIPCHILDREN | Will cause the child window area to be excluded from the drawing region; in other words, the parent cannot paint over the child. Usually this style is not necessary because if both the parent and child windows need to be repainted and also overlap, the parent will be repainted first, and then the child window is repainted. |
| WS_DISABLED | Will cause a window to be disabled upon creation. Thus this window will not respond to user input until the window is enabled. |
| WS_VISIBLE | Will make a window visible at creation time. An invisible default window will be created. |

Table 9.7 presents class styles that can be specified at class registration time.

Table 9.7 Class Style Descriptions

| Class Style | Description |
|---|---|
| CS_MOVENOTIFY | WM_MOVE messages will be sent whenever the window is moved. |
| CS_SIZEREDRAW | When a window has been sized, the window will be made completely invalid, and a WM_PAINT message will be sent. This style is useful when an application centers text on the window or sizes an image to fill the window. |
| CS_HITTEST | WM_HITTEST messages will be sent to the window whenever the mouse moves in the window. |
| CS_FRAME | Specifies a frame window class. |
| CS_CLIPCHILDREN | See above. |
| CS_CLIPSIBLINGS | See above. |
| CS_PARENTCLIP | See above. |
| CS_SAVEBITS | See above. |
| CS_SYNCPAINT | See above. |

# Another Window Example: WINDOW

The following example program illustrates some of the concepts we've talked about so far and includes some new ones also. The program, WINDOW, creates a list of all the windows that are children of the frame window and also queries the window style of each window. The information is displayed in the client area.

**WINDOW.C**

```
#define  INCL_WIN
#define  INCL_GPI
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <os2.h>
#define  CLS_CLIENT     "MyClass"
```

```
MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

typedef struct   _LINEINFO
{
   ULONG            ulCharHeight;
   ULONG            usxLeft;
   ULONG            usxRight;
} LINEINFO,*PLINEINFO;

VOID DisplayError(CHAR *pszText);
USHORT DropOneLine(PRECTL prclRect,ULONG ulCharHeight);
VOID DrawString(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                PLINEINFO pLineInfo,CHAR *pString);
VOID WriteWindowInfo(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                     PLINEINFO pLineInfo);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   HWND             hwndClient;
   BOOL             bLoop;
   QMSG             qmMsg;

   /*************************************************************/
   /* initialization                                            */
   /*************************************************************/

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
      FCF_SHELLPOSITION|FCF_TASKLIST;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Titlebar",
                                  0L,
                                  NULLHANDLE,
                                  0,
                                  &hwndClient);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
```

```
                              0);
      }                                   /* endwhile                */
      WinDestroyWindow(hwndFrame);
   }                                      /* endif                   */

   /************************************************************/
   /* clean-up                                                 */
   /************************************************************/

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{

   switch (ulMsg)
   {
      case  WM_PAINT :
         {
            HPS               hpsPaint;
            RECTL             rclRect;
            RECTL             rclWindow;
            HWND              hwndEnum;
            HWND              hwndFrame;
            HENUM             heEnum;
            FONTMETRICS       fmMetrics;
            LINEINFO          liLineInfo;

            hpsPaint = WinBeginPaint(hwndWnd,
                                     NULLHANDLE,
                                     &rclRect);

            /*****************************************************/
            /* erase the invalid region                          */
            /*****************************************************/

            WinFillRect(hpsPaint,
                        &rclRect,
                        SYSCLR_WINDOW);

            /*****************************************************/
            /* get font size and save information                */
            /*****************************************************/

            GpiQueryFontMetrics(hpsPaint,
                                sizeof(fmMetrics),
                                &fmMetrics);

            liLineInfo.ulCharHeight = fmMetrics.lMaxBaselineExt;

            /*****************************************************/
            /* calculate window size information                 */
            /*****************************************************/

            WinQueryWindowRect(hwndWnd,
                               &rclWindow);
            liLineInfo.usxLeft = (USHORT)fmMetrics.lAveCharWidth;
            liLineInfo.usxRight = rclWindow.xRight-(USHORT)
               fmMetrics.lAveCharWidth;

            /*****************************************************/
            /* move down one line                                */
            /*****************************************************/

            rclWindow.yTop = rclWindow.yTop-
               liLineInfo.ulCharHeight;
```

```
            rclWindow.yBottom = rclWindow.yTop-
               liLineInfo.ulCharHeight;

            /************************************************/
            /* start enumerating with the frame window      */
            /************************************************/

            hwndFrame = WinQueryWindow(hwndWnd,
                                       QW_PARENT);
            WriteWindowInfo(hpsPaint,
                            hwndFrame,
                            &rclWindow,
                            &liLineInfo);

            heEnum = WinBeginEnumWindows(hwndFrame);

            hwndEnum = WinGetNextWindow(heEnum);

            while (hwndEnum != NULLHANDLE)
            {
               WriteWindowInfo(hpsPaint,
                               hwndEnum,
                               &rclWindow,
                               &liLineInfo);
               hwndEnum = WinGetNextWindow(heEnum);
            }                          /* end while hwndEnum   */
            WinEndEnumWindows(heEnum);

            WinEndPaint(hpsPaint);
         }
         break;
      default  :
         return  WinDefWindowProc(hwndWnd,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

VOID WriteWindowInfo(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                     PLINEINFO pLineInfo)
{
   CHAR             achString[200],achStyle[200];
   CHAR             achClass[65];
   CHAR             achClassText[25];
   PCHAR            pchStart;
   USHORT           usIndex;
   HWND             hwndParent,hwndOwner;
   ULONG            ulStyle;
   PCHAR            apchClasses[] =
   {
      " ","WC_FRAME","WC_COMBOBOX","WC_BUTTON","WC_MENU",
      "WC_STATIC","WC_ENTRYFIELD","WC_LISTBOX","WC_SCROLLBAR",
      "WC_TITLEBAR"
   }

   ;

   /************************************************************/
   /* get window class name                                    */
   /************************************************************/

   WinQueryClassName(hwndPaint,
                     sizeof(achClass),
                     achClass);
```

```
   /*************************************************************/
   /* save start of classname                                   */
   /*************************************************************/

   pchStart = achClass;

   /*************************************************************/
   /* public classes are enumerated as #number strip off the    */
   /* pound sign to get index to class name array               */
   /*************************************************************/

   if (achClass[0] == '#')
   {
      usIndex = atoi(&achClass[1]);
      strcpy(achClassText,
             apchClasses[usIndex]);
   }
   else
   {
      strcpy(achClassText,
             pchStart);
   }                                  /* endif                  */

   /*************************************************************/
   /* get window parent and owner                               */
   /*************************************************************/

   hwndParent = WinQueryWindow(hwndPaint,
                               QW_PARENT);
   hwndOwner = WinQueryWindow(hwndPaint,
                              QW_OWNER);

   sprintf(achString,
  "Window: 0x%08lX Class: %s, Parent: 0x%08lX Owner: 0x%08lX",
           hwndPaint,
           achClassText,
           hwndParent,
           hwndOwner);

   /*************************************************************/
   /* draw the string                                           */
   /*************************************************************/

   DrawString(hpsPaint,
              hwndPaint,
              prclRect,
              pLineInfo,
              achString);

   /*************************************************************/
   /* get the window styles                                     */
   /*************************************************************/

   ulStyle = WinQueryWindowULong(hwndPaint,
                                 QWL_STYLE);
   strcpy(achStyle, "Styles: ");

   /*************************************************************/
   /* check for some of the more common styles                  */
   /*************************************************************/

   if (ulStyle&WS_VISIBLE)
      strcat(achStyle, "Visible;");
   if (ulStyle&WS_DISABLED)
      strcat(achStyle, " Disabled;");
   if (ulStyle&WS_CLIPCHILDREN)
      strcat(achStyle, " Clip Children;");
   if (ulStyle&WS_CLIPSIBLINGS)
```

```
      strcat(achStyle, " Clip Siblings;");
   if (ulStyle&WS_PARENTCLIP)
      strcat(achStyle, " Clip to Parent;");
   if (ulStyle&WS_MAXIMIZED)
      strcat(achStyle, " Maximized;");
   if (ulStyle&WS_MINIMIZED)
      strcat(achStyle, " Minimized;");
   if (ulStyle&WS_SYNCPAINT)
      strcat(achStyle, " Sync Paint;");
   if (ulStyle&WS_SAVEBITS)
      strcat(achStyle, " Save Bits;");
   if (usIndex == 1 && (ulStyle&FCF_TITLEBAR))
      strcat(achStyle," Titlebar;");
   if (usIndex == 1 && (ulStyle&FCF_SYSMENU))
      strcat(achStyle, " System Menu;");
   if (usIndex == 1 && (ulStyle&FCF_MENU))
      strcat(achStyle, " Menu;");
   if (usIndex == 1 && (ulStyle&FCF_SIZEBORDER))
      strcat(achStyle, " Size Border;");
   if (usIndex == 1 && (ulStyle&FCF_SHELLPOSITION))
      strcat(achStyle, " Shell Position;");
   if (usIndex == 1 && (ulStyle&FCF_TASKLIST))
      strcat(achStyle, " Task List;");

   DrawString(hpsPaint,
              hwndPaint,
              prclRect,
              pLineInfo,
              achStyle);
   DropOneLine(prclRect,
               pLineInfo->ulCharHeight);

   return ;
}

VOID DisplayError(CHAR *pszText)
{

   /*************************************************************/
   /* small function to display error string                    */
   /*************************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}

USHORT DropOneLine(PRECTL prclRect,ULONG ulCharHeight)
{

   /*************************************************************/
   /* small function to move down one line                      */
   /*************************************************************/

   LONG              lNewBottom;

   /*************************************************************/
   /* will new bottom go off the edge of the window?            */
   /*************************************************************/

   lNewBottom = (LONG)(prclRect->yTop-(2*ulCharHeight));
```

```
   if (lNewBottom >= 0)
   {
      prclRect->yTop = prclRect->yBottom;
      prclRect->yBottom = lNewBottom;
      return (TRUE);
   }
   else
      return (FALSE);
}

VOID DrawString(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                PLINEINFO pLineInfo,CHAR *pString)
{

   USHORT           usStringLength;
   USHORT           usNumChars;
   BOOL             bFinished;
   USHORT           usOffset;
   USHORT           usReturn;

   bFinished = FALSE;
   usStringLength = strlen(pString);

   usOffset = 0;

   while (!bFinished)
   {

      /*********************************************************/
      /* move down to next line                                */
      /*********************************************************/

      usReturn = DropOneLine(prclRect,
                             pLineInfo->ulCharHeight);

      /*********************************************************/
      /* if we can't move down any more, stop trying to write  */
      /* any more                                              */
      /*********************************************************/

      if (!usReturn)
         return ;

      /*********************************************************/
      /* set the left and right drawing coordinates            */
      /*********************************************************/

      prclRect->xLeft = pLineInfo->usxLeft;
      prclRect->xRight = pLineInfo->usxRight;

      /*********************************************************/
      /* draw text that will fit                               */
      /*********************************************************/

      usNumChars = WinDrawText(hpsPaint,
                               strlen(&pString[usOffset]),
                               &pString[usOffset],
                               prclRect,
                               0,
                               0,
                               DT_LEFT|DT_TEXTATTRS|DT_WORDBREAK)
         ;

      if (!usNumChars || (usOffset+usNumChars == usStringLength))

         /*****************************************************/
         /* if no characters were printed, or we are at the end*/
         /* of the string, quit                               */
         /*****************************************************/
```

```
        bFinished = TRUE;
      else

        /*******************************************************/
        /* offset string to new position                       */
        /*******************************************************/

        usOffset += usNumChars;
   }

   return ;

}
```

**WINDOW.MAK**

```
WINDOW.EXE:                     WINDOW.OBJ
        LINK386 @<<
WINDOW
WINDOW
WINDOW
OS2386
WINDOW
<<

WINDOW.OBJ:                     WINDOW.C
        ICC -C+ -Kb+ -Ss+ WINDOW.C
```

**WINDOW.DEF**

```
NAME WINDOW WINDOWAPI

DESCRIPTION 'Window list example
Copyright (c) 1995 by Kathleen Panov
All rights reserved.'
STACKSIZE 16384
```

Here *main* has one small difference from *main* in the previous example, WIN1.C. The class style, CS_SIZEREDRAW, is used for the client window class. With this style, Presentation Manager will invalidate the window whenever the size changes. The text on the client area is dependent on the width of the window. Because we want to ensure that all the text is nicely formatted even when the window is resized, thus we use CS_SIZEREDRAW.

# The Presentation Manager Coordinate Space

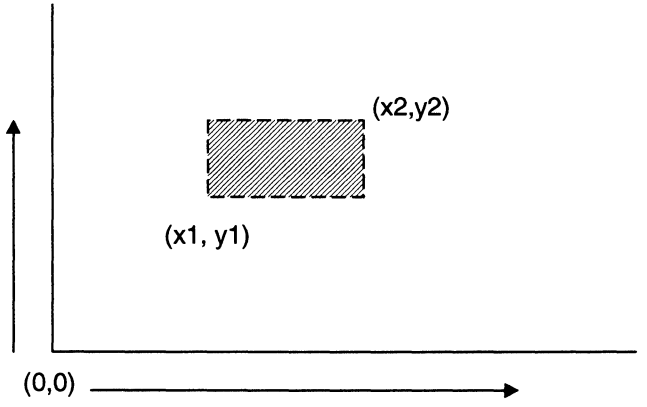



Figure 9.6 Coordinate space.

Presentation Manager windows use a different coordinate space from the one used by Microsoft Windows. (See Figure 9.6) The bottom left corner is coordinate 0,0. Most window drawing is done by specifying two sets of x,y coordinates that form the lower left and upper right corners of a "bounding rectangle." A structure RECTL contains the coordinates. It is a familiar parameter in most painting functions. The structure is defined:

```
typedef struct _RECTL
{       LONG xLeft ;
        LONG yBottom;
        LONG xRight ;
        LONG yTop;
} RECTL ;
```

# More on Window Painting

In a structured program, the application controls exactly when the screen is updated; in an event-driven environment, the system tells the application when it can update the screen. This is done by sending the application the WM_PAINT message. A Presentation Manager program should update the screen within the WM_PAINT message processing.

This message is one of the most common messages to handle. A WM_PAINT message is generated whenever some part of the client window needs to be painted. If a user moves one window on top of another window, the bottom window receives a WM_PAINT message when the covered area becomes visible again. When a portion of a window needs to be repainted, that portion is said to be "invalid." Presentation Manager can invalidate a region or a programmer can invalidate a region using *WinInvalidateRegion* or *WinInvalidateRect.*

```
BOOL  APIENTRY WinInvalidateRect(HWND hwnd,
                                 PRECTL pwrc,
                                 BOOL fIncludeChildren);

BOOL  APIENTRY WinInvalidateRegion(HWND hwnd,
```

```
BOOL  APIENTRY WinInvalidateRegion(HWND hwnd,
                                   HRGN hrgn,
                                   BOOL fIncludeChildren);
```

The first parameter for these functions is the window handle *hwnd.* The next parameter is the area that is to be invalidated. The last parameter indicates whether children are to be included in the invalid rectangle or region.

Presentation Manager is very stingy in sending WM_PAINT messages. Only that piece of the window that needs to be painted will be invalidated, *not* the entire window.

# Painting by Numbers

```
hpsPaint = WinBeginPaint ( hwndWnd,
                           NULLHANDLE,
                           &rclRect ) ;
```

Painting in this example starts with *WinBeginPaint* to obtain a presentation space.

```
HPS   APIENTRY WinBeginPaint(HWND hwnd,
                             HPS hps,
                             PRECTL prclPaint);
```

*hwndWnd* is the window the presentation space belongs to. Presentation spaces are covered in more detail later. The second parameter is used if the user already has a presentation space obtained using *WinGetPS* or some other means and wants to use that space for drawing. If a NULLHANDLE is specified, the system will provide a presentation space to be used. The last parameter is a pointer to RECTL structure. The coordinates of the invalidated region are placed in the structure. The invalidated region is the region that needs to be painted.

```
BOOL WinEndPaint(HPS hps);
```

*WinEndPaint* is used to terminate a paint procedure. There is only one parameter, *hps*, which is the presentation space returned from *WinBeginPaint.*

Once *WinEndPaint* is called, the region is validated, and any presentation space returned from *WinBeginPaint* is released.

```
WinFillRect(hpsPaint,
            &rclRect,
            SYSCLR_WINDOW);
```

This function paints the region designated by the second parameter with the specified color index. The first parameter is the presentation space to paint.

```
BOOL WinFillRect(HPS hps,
                 PRECTL prcl,
                 LONG lColor);
```

A program can use a value such as CLR_BLUE or a system value such as SYSCLR_WINDOW that will fill the rectangle with the system default window color.

The WINDOW example is designed to draw some text on the client window area; however, in a graphical user interface (GUI) environment, this is not just a call to *printf*. Instead the developer must provide the exact pixel location where the text is to be located. Before we get around actually to drawing the text, we need to find some information about the size of the font used in the client window. *GpiQueryFontMetrics* is the function to provide all the needed information about a font.

```
BOOL GpiQueryFontMetrics(HPS hps,
                         LONG lMetricsLength,
                         PFONTMETRICS pfmMetrics);


GpiQueryFontMetrics ( hpsPaint,
                      sizeof ( fmMetrics ) ,
                      &fmMetrics ) ;
```

The FONTMETRICS structure contains much data concerning the point size, face name, height, and width of the current font. The variable *lMaxBaselineExt* provides the height of the tallest character. We'll use this value as the height-of-line line of text.

```
WinQueryWindowRect(hwndWnd,
                   &rclWindow);
liLineInfo.usxLeft = (USHORT)fmMetrics.lAveCharWidth;
liLineInfo.usxRight = rclWindow.xRight -
                      (USHORT)fmMetrics.lAveCharWidth;
```

The next task is to find the current size of the window. Remember, a window can be sized by the user at any time, and a program should be able to adjust to such changes. *WinQueryWindowRect* will return the size of a window in a RECTL structure. We will define a right and left margin that is equal to the average width of one character. Very conveniently, the FONTMETRICS structure contains *lAveCharWidth*, which is exactly that. With all this, we now know the height of our lines, the x coordinate our lines will start at, and the x coordinate that is the end of the line.

To position the first line of text at the top of the page and create a one-line margin, the following math is done to move the bounding rectangle down one line.

```
rclWindow.yTop = rclWindow.yTop-
                 liLineInfo.ulCharHeight;
rclWindow.yBottom = rclWindow.yTop-
                    liLineInfo.ulCharHeight;
```

# Enumerating Windows

```
hwndFrame = WinQueryWindow(hwndWnd,
                           QW_PARENT);
```

Remember, in the window procedure, *hwndWnd* is the client window, not the frame window. *WinQueryWindow* is used to find the parent of the client window, which in our case is the frame window. This is a very simple function that will be used many times. The first parameter is the handle of the window to query, and the second parameter indicates what information will be returned.

Table 9.8 *WinQueryWindow* Flags

| Value | Description |
|---|---|
| QW_NEXT | Returns the window below the specified window. |
| QW_PREV | Returns the window above the specified window. |

| Value | Description |
|---|---|
| QW_TOP | Returns the topmost child window. |
| QW_BOTTOM | Returns the bottommost child window. |
| QW_OWNER | Returns the owner of the specified window. |
| QW_PARENT | Returns the parent of the specified window. |
| QW_NEXTTOP | Returns the next window of the owner window hierarchy. |
| QW_PREVTOP | Returns the previous window of the owner window hierarchy. |
| QW_FRAMEOWNER | Returns the owner of the specified window that also shares the same parent as the specified window. |

```
WriteWindowInfo(hpsPaint,
                hwndFrame,
                &rclWindow,
                &liLineInfo);

heEnum = WinBeginEnumWindows(hwndFrame);

hwndEnum = WinGetNextWindow(heEnum);

while (hwndEnum != NULLHANDLE)
{
   WriteWindowInfo(hpsPaint,
                   hwndEnum,
                   &rclWindow,
                   &liLineInfo);
   hwndEnum = WinGetNextWindow(heEnum);
}                          /* end while hwndEnum   */
WinEndEnumWindows(heEnum);
```

Presentation Manager lets users query all the descendants of a particular window by using the functions *WinBeginEnumWindows* and *WinGetNextWindow*. The window that is the head of the window family tree is the frame window, *hwndFrame*.

```
HENUM WinBeginEnumWindows(HWND hwnd);
HWND WinGetNextWindow(HENUM henum);
BOOL WinEndEnumWindows(HENUM henum);
```

This window handle is passed to *WinBeginEnumWindows*, which passes back an enumeration handle, *heEnum*. This is a place holder to keep track of the last window that was returned. *WinGetNextWindow* takes *heEnum* and returns the next window in the window family tree. As each window is found, our own function, *WriteWindowInfo*, is used to display information about the window. The enumeration ends with a call to *WinEndEnumWindows*.

# WriteWindowInfo

```
WinQueryClassName ( hwndPaint,
                    sizeof ( achClass ) ,
                    achClass ) ;
```

The first piece of information we'll retrieve from each window is the class name. Documentation refers to the system-defined class names as WC_FRAME and so on. However, the class name in reality, and returned by *WinQueryClassName,* is a string in the format "#1". Some help, huh? Public window class names are stored in powerful lookup tables known as *atom tables*. This format helps to check to see if a newly registered window class has the same name as one that is already registered. To convert from this cryptic format to something more readily deciphered, we define an array, *pszClassNames,* that maps the numeric class names to the documented class names. The string *pszClass*, returned from

*WinQueryClassName*, is incremented by one to strip off the "#" and leave a value that can be converted to an integer index into the array.

If *achClass* is a nonnumeric value, we assume this to be an application-defined class, and keep the string whole.

The second piece of information to retrieve is the parent and owner windows. *WinQueryWindow* is used to return the window handles of the parent and owner of the specified window. All this window detail is formatted into one string that will displayed on the client window by the user-function *DrawString*.

```
ulStyle = WinQueryWindowULong(hwndPaint,
                              QWL_STYLE);
```

The other information we will output to the client area is the window styles. This is a value that is stored in the window word. Presentation Manager stores a lot of window information in window words. The next example covers this concept in more detail. *WinQueryWindowULong* will retrieve the window styles. The first parameter is the window we're inquiring about. The next parameter is a constant used to identify which piece of the window word we're after. The value QWL_STYLE designates that the window style is the ULONG is question. The example converts these values to meaningful text string and uses the function *DrawString* to display the formatted string on the client area.

## The *DrawString* function

Two functions will draw text on a window, *WinDrawText* and *GpiCharString*. *WinDrawText* is the more powerful function, that providing such features as positioning, colored text, and word break. *GpiCharString* is much faster but leaves more work for the programmer.

```
LONG lCharDrawn = WinDrawText(      HPS hpsPaint,
                                    LONG lCount,
                                    PCH pchString,
                                    PRECTL prclRect,
                                    LONG lForeColor,
                                    LONG lBackColor,
                                    LONG flCmd
```

We use the *WinDrawText* function in this example. The first parameter is the presentation space. The second parameter is the number of characters to output. A -1 indicates that the entire length of the null-terminated string is to be used. The string to write is *pchString*. The size of the text area is defined by passing a pointer to a RECTL structure that contains the designated coordinates. The next two parameters indicate the foreground and background colors of the text. The last parameter is the formatting flag, a collection of formatting attributes that are ORed together. The attributes used in this example are DT_LEFT|DT_TEXTATTRS_DT_WORDBREAK. DT_LEFT left-aligns the text horizontally, and DT_TEXTATTRS indicates that default window foreground and background colors will be used. If this flag is specified, the two previous parameters are ignored. The DT_WORDBREAK attribute will draw only the number of *whole words* that will fit inside the bounding rectangle. The number of characters drawn is returned. By enclosing this code in a loop and incrementing the string offset by the number of characters drawn, a very powerful routine that will be used often to print formatted text is created.

```
while (!bFinished)
   {

      usReturn = DropOneLine(prclRect,
                             pLineInfo->ulCharHeight);
```

```
        if (!usReturn)
           return ;

        prclRect->xLeft = pLineInfo->usxLeft;
        prclRect->xRight = pLineInfo->usxRight;

        usNumChars = WinDrawText(hpsPaint,
                                 strlen(&pString[usOffset]),
                                 &pString[usOffset],
                                 prclRect,
                                 0,
                                 0,
                                 DT_LEFT|DT_TEXTATTRS|DT_WORDBREAK)
           ;

        if (!usNumChars || (usOffset+usNumChars == usStringLength))
          bFinished = TRUE;
        else
           usOffset += usNumChars;
    }
```

There is one last short function to explain, *DropOneLine*. This is a user function that will take a pointer to a RECTL structure and decrement the top and bottom y coordinates by the height of one line.

# Presentation Spaces

A presentation space is similar to an artist's canvas. It is the space where the application draws. However, a presentation space does not have to be a window. It could also be a printer or even some piece of memory. In reality, a presentation space is a data structure, but to the programmer it is the drawing area. There are two types of presentation spaces—a normal presentation space and a micropresentation space. A micropresentation space is designed to have output to only one source. A normal presentation space can be shared between multiple devices. For instance, to print some copy of the video display, a normal presentation space would be used. A normal presentation space uses more memory than a micropresentation space and is slower; however, it is the most powerful presentation space type available.

There are two types of micropresentation spaces—standard and cached. A microcached presentation space is used for the video display and is maintained by Presentation Manager. A microcached presentation space is faster than the other presentation spaces and uses less memory. A microstandard presentation space is used to send output to a printer or any other output device. However, it cannot send output to more than one device at a time.

Presentation Manager controls how much of a window actually belongs in the presentation space. For example, if another window is covering most of a window, who should be able to draw on the intersection of the two windows? The normal answer is the window with the highest value in the Z-order. There are a few exceptions to this rule.

- WS_CLIPCHILDREN If a window has this style, when the child window overlaps the parent, the parent window cannot draw on any part of the child's window. Normally, a child has a higher place in the Z-order than the parent, anyway, and this style is not necessary.
- WS_CLIPSIBLINGS When two windows share the same parent, this style will omit a sibling's presentation space from that of the other sibling. This style can be used to make sure one sibling always "comes out on top."
- WS_PARENTCLIP This gives a child window the ability to draw on its parent. This style can be potentially dangerous, esthetically speaking, because the parent's presentation space is larger than the child's space. However, somebody must have had a use for it somewhere.

# Window Words

Window words is a fairly simple concept that is fairly easy to implement, but it got a bad rap because it was poorly documented. Every window has a pointer to some memory that contains quite a bit of very interesting information. Such things as window ID, frame flags, window style, and much more are available through window words. Table 9.9 presents three sets of functions that are used to set and query the information.

Table 9.9 Data Type Returned From Window Word Functions

| *Function* | Data Type Returned |
|---|---|
| *WinQueryWindowUShort* | USHORT |
| *WinSetWindowUShort* | USHORT |
| *WinQueryWindowULong* | ULONG |
| *WinSetWindowULong* | ULONG |
| *WinQueryWindowPtr* | PVOID |
| *WinSetWindowPtr* | PVOID |

Four bytes of space are reserved in the window word for the programmer. These four bytes can contain any data type that will fit in the space. If more space is needed, the programmer should create his or her own structure and pass a pointer to the structure in the window word.

Specific information from the window word is obtained using QWL_*, QWS_*, and QWP_* values. These values are constants that represent the offset into the window word. The L, S, and P indicate the data type that resides at that offset. The programmer-defined data space resides at offset QWL_USER. One note here: The following control windows contain the programmer-defined data area:

- Frames
- Dialog boxes
- Combo boxes
- Buttons
- Menus
- Static text
- Entryfields
- Listboxes
- Scrollbars
- Titlebars
- MLEs
- Spin buttons
- Containers
- Sliders
- Value set
- Notebooks

The following example modifies the WINDOW program to use a window word to save the window handles and prevent multiple window enumerations in the WM_PAINT processing.

**WINWORD.C**

```
#define  INCL_WIN
#define  INCL_GPI
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <os2.h>
#define  CLS_CLIENT    "MyClass"
MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

typedef struct   _WININFO
{
   ULONG            ulStructSize;
   BOOL             bStructInit;
   SHORT            sNumWindows;
   HWND             ahwndWindows[10];

} WININFO,*PWININFO;

typedef struct   _LINEINFO
{
   ULONG            ulCharHeight;
   ULONG            usxLeft;
   ULONG            usxRight;
} LINEINFO,*PLINEINFO;

VOID DisplayError(CHAR *pszText);
USHORT DropOneLine(PRECTL prclRect,ULONG ulCharHeight);
VOID DrawString(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                PLINEINFO pLineInfo,CHAR *pString);

VOID WriteWindowInfo(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                     PLINEINFO pLineInfo);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   HWND             hwndClient;
   BOOL             bLoop;
   QMSG             qmMsg;

   /*************************************************************/
   /* initialization                                            */
   /*************************************************************/

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
      FCF_SHELLPOSITION|FCF_TASKLIST;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Titlebar",
                                  0L,
```

```
                                NULLHANDLE,
                                0,
                                &hwndClient);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                 /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                    /* endif                 */

   /*************************************************************/
   /* clean-up                                                  */
   /*************************************************************/

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PWININFO         pWinInfo;

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
            pWinInfo = (PWININFO)calloc(1,
                                        sizeof(WININFO));
            if (pWinInfo)
               WinSetWindowPtr(hwndWnd,
                               QWL_USER,
                               pWinInfo);
            else
               DisplayError("No memory allocated for pWinInfo");
            break;
         }
      case  WM_DESTROY :
         {

            pWinInfo = (PWININFO)WinQueryWindowPtr(hwndWnd,
                                                   QWL_USER);
            if (pWinInfo)
               free(pWinInfo);

            break;
         }

      case  WM_PAINT :
         {
            HPS               hpsPaint;
            RECTL             rclRect;
            RECTL             rclWindow;
```

```
HWND            hwndEnum;
HENUM           heEnum;
FONTMETRICS     fmMetrics;
SHORT           i;
LINEINFO        liLineInfo;

hpsPaint = WinBeginPaint(hwndWnd,
                         NULLHANDLE,
                         &rclRect);

/*************************************************/
/* erase the invalid region                      */
/*************************************************/

WinFillRect(hpsPaint,
            &rclRect,
            SYSCLR_WINDOW);

/*************************************************/
/* get font size and save information            */
/*************************************************/

GpiQueryFontMetrics(hpsPaint,
                    sizeof(fmMetrics),
                    &fmMetrics);

liLineInfo.ulCharHeight = fmMetrics.lMaxBaselineExt;

/*************************************************/
/* calculate window size information             */
/*************************************************/

WinQueryWindowRect(hwndWnd,
                   &rclWindow);
liLineInfo.usxLeft = (USHORT)fmMetrics.lAveCharWidth;
liLineInfo.usxRight = rclWindow.xRight-(USHORT)
   fmMetrics.lAveCharWidth;

/*************************************************/
/* move down one line                            */
/*************************************************/

rclWindow.yTop = rclWindow.yTop-
   liLineInfo.ulCharHeight;
rclWindow.yBottom = rclWindow.yTop-
   liLineInfo.ulCharHeight;

/*************************************************/
/* retrieve window word                          */
/*************************************************/

pWinInfo = (PWININFO)WinQueryWindowPtr(hwndWnd,
                                       QWL_USER);
if (!pWinInfo)

   /*********************************************/
   /* error                                     */
   /*********************************************/

   DisplayError("No pWinInfo retrieved in WM_PAINT");
else
{

   /*********************************************/
   /* first time through initialize structures and */
   /* enumerate windows                         */
   /*********************************************/
```

```
if (!pWinInfo->bStructInit)
{

   SHORT             sNumWindows = 0;
   HWND              hwndFrame;

   /*******************************************/
   /* start enumerating with the frame window    */
   /*******************************************/

   hwndFrame = WinQueryWindow(hwndWnd,
                              QW_PARENT);
   WriteWindowInfo(hpsPaint,
                   hwndFrame,
                   &rclWindow,
                   &liLineInfo);

   heEnum = WinBeginEnumWindows(hwndFrame);

   /*******************************************/
   /* save the window handles in window word    */
   /*******************************************/

   pWinInfo->ahwndWindows[sNumWindows] = hwndFrame
      ;
   hwndEnum = WinGetNextWindow(heEnum);

   while (hwndEnum != NULLHANDLE)
   {
      WriteWindowInfo(hpsPaint,
                      hwndEnum,
                      &rclWindow,
                      &liLineInfo);
      sNumWindows++;
      pWinInfo->ahwndWindows[sNumWindows] =
         hwndEnum;
      hwndEnum = WinGetNextWindow(heEnum);
   }                        /* end while hwndEnum   */
   WinEndEnumWindows(heEnum);

   /*******************************************/
   /* keep track of how many windows we have    */
   /*******************************************/

   pWinInfo->sNumWindows = sNumWindows;

   /*******************************************/
   /* set flag so we know structure is          */
   /* initialized                               */
   /*******************************************/

   pWinInfo->bStructInit = TRUE;
}                          /* end if structure is
                              NOT initialized      */
else
{

   /*******************************************/
   /* if structure has been initialized, just   */
   /* loop through the window handles and write */
   /* info to the window                        */
   /*******************************************/

   for (i = 0; i <= pWinInfo->sNumWindows; i++)
   {
      WriteWindowInfo(hpsPaint,
                      pWinInfo->ahwndWindows[i],
                      &rclWindow,
                      &liLineInfo);
```

```
                }                          /* end for                */
             }                             /* end else struct is
                                              initialized            */
          }                                /* end else               */
          WinEndPaint(hpsPaint);
       }
       break;
    default  :
       return  WinDefWindowProc(hwndWnd,
                                ulMsg,
                                mpParm1,

                                mpParm2);
                                     /* endswitch                    */
  }
  return  MRFROMSHORT(FALSE);
}

VOID WriteWindowInfo(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                     PLINEINFO pLineInfo)
{
  CHAR             achString[200],achStyle[200];
  CHAR             achClass[65];
  CHAR             achClassText[25];
  PCHAR            pchStart;
  USHORT           usIndex;
  HWND             hwndParent,hwndOwner;
  ULONG            ulStyle;
  PCHAR            apchClasses[] =
  {
     " ","WC_FRAME","WC_COMBOBOX","WC_BUTTON","WC_MENU",
     "WC_STATIC","WC_ENTRYFIELD","WC_LISTBOX","WC_SCROLLBAR",
     "WC_TITLEBAR"
  }

  ;

  /*************************************************************/
  /* get window class name                                     */
  /*************************************************************/

  WinQueryClassName(hwndPaint,
                    sizeof(achClass),
                    achClass);

  /*************************************************************/
  /* save start of classname                                   */
  /*************************************************************/

  pchStart = achClass;

  /*************************************************************/
  /* public classes are enumerated as #number strip off the    */
  /* pound sign to get index to class name array               */
  /*************************************************************/

  if (achClass[0] == '#')
  {
     usIndex = atoi(&achClass[1]);
     strcpy(achClassText,
            apchClasses[usIndex]);
  }
  else
  {
     strcpy(achClassText,
            pchStart);
  }                                  /* endif                    */
```

```
/*************************************************************/
/* get window parent and owner                               */
/*************************************************************/

hwndParent = WinQueryWindow(hwndPaint,
                            QW_PARENT);
hwndOwner = WinQueryWindow(hwndPaint,
                           QW_OWNER);

sprintf(achString,
"Window: 0x%08lX Class: %s, Parent: 0x%08lX Owner: 0x%08lX",
        hwndPaint,
        achClassText,
        hwndParent,
        hwndOwner);

/*************************************************************/
/* draw the string                                           */
/*************************************************************/

DrawString(hpsPaint,
           hwndPaint,
           prclRect,
           pLineInfo,
           achString);

/*************************************************************/
/* get the window styles                                     */
/*************************************************************/

ulStyle = WinQueryWindowULong(hwndPaint,
                              QWL_STYLE);
strcpy(achStyle,
       "Styles: ");

/*************************************************************/
/* check for some of the more common styles                  */
/*************************************************************/

if (ulStyle&WS_VISIBLE)
   strcat(achStyle,
          "Visible;");
if (ulStyle&WS_DISABLED)
   strcat(achStyle,
          " Disabled;");
if (ulStyle&WS_CLIPCHILDREN)
   strcat(achStyle,
          " Clip Children;");
if (ulStyle&WS_CLIPSIBLINGS)
   strcat(achStyle,
          " Clip Siblings;");
if (ulStyle&WS_PARENTCLIP)
   strcat(achStyle,
          " Clip to Parent;");
if (ulStyle&WS_MAXIMIZED)
   strcat(achStyle,
          " Maximized;");
if (ulStyle&WS_MINIMIZED)
   strcat(achStyle,
          " Minimized;");
if (ulStyle&WS_SYNCPAINT)
   strcat(achStyle,
          " Sync Paint;");
if (ulStyle&WS_SAVEBITS)
   strcat(achStyle,
          " Save Bits;");
if (usIndex == 1 && (ulStyle&FCF_TITLEBAR))
   strcat(achStyle,
          " Titlebar;");
```

```
   if (usIndex == 1 && (ulStyle&FCF_SYSMENU))
      strcat(achStyle,
             " System Menu;");
   if (usIndex == 1 && (ulStyle&FCF_MENU))
      strcat(achStyle,
             " Menu;");
   if (usIndex == 1 && (ulStyle&FCF_SIZEBORDER))
      strcat(achStyle,
             " Size Border;");
   if (usIndex == 1 && (ulStyle&FCF_SHELLPOSITION))
      strcat(achStyle,
             " Shell Position;");
   if (usIndex == 1 && (ulStyle&FCF_TASKLIST))
      strcat(achStyle,
             " Task List;");

   DrawString(hpsPaint,
              hwndPaint,
              prclRect,
              pLineInfo,
              achStyle);
   DropOneLine(prclRect,
               pLineInfo->ulCharHeight);

   return ;
}

VOID DisplayError(CHAR *pszText)
{

   /*************************************************************/
   /* small function to display error string                    */
   /*************************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}

USHORT DropOneLine(PRECTL prclRect,ULONG ulCharHeight)
{

   /*************************************************************/
   /* small function to move down one line                      */
   /*************************************************************/

   LONG              lNewBottom;

   /*************************************************************/
   /* will new bottom go off the edge of the window?            */
   /*************************************************************/

   lNewBottom = (LONG)(prclRect->yTop-(2*ulCharHeight));
   if (lNewBottom >= 0)
   {
      prclRect->yTop = prclRect->yBottom;
      prclRect->yBottom = lNewBottom;
      return (TRUE);
   }
   else
      return (FALSE);
```

```
}

VOID DrawString(HPS hpsPaint,HWND hwndPaint,PRECTL prclRect,
                PLINEINFO pLineInfo,CHAR *pString)
{

   USHORT           usStringLength;
   USHORT           usNumChars;
   BOOL             bFinished;
   USHORT           usOffset;
   USHORT           usReturn;

   bFinished = FALSE;
   usStringLength = strlen(pString);

   usOffset = 0;

   while (!bFinished)
   {

      /*********************************************************/
      /* move down to next line                                */
      /*********************************************************/

      usReturn = DropOneLine(prclRect,
                             pLineInfo->ulCharHeight);

      /*********************************************************/
      /* if we can't move down any more, stop trying to write  */
      /* any more                                              */
      /*********************************************************/

      if (!usReturn)
         return ;

      /*********************************************************/
      /* set the left and right drawing coordinates            */
      /*********************************************************/

      prclRect->xLeft = pLineInfo->usxLeft;
      prclRect->xRight = pLineInfo->usxRight;

      /*********************************************************/
      /* draw text that will fit                               */
      /*********************************************************/

      usNumChars = WinDrawText(hpsPaint,
                               strlen(&pString[usOffset]),
                               &pString[usOffset],
                               prclRect,
                               0,
                               0,
                               DT_LEFT|DT_TEXTATTRS|DT_WORDBREAK)
         ;

      if (!usNumChars || (usOffset+usNumChars == usStringLength))

         /*****************************************************/
         /* if no characters were printed, or we are at the end*/
         /* of the string, quit                               */
         /*****************************************************/

         bFinished = TRUE;
      else

         /*****************************************************/
         /* offset string to new position                     */
         /*****************************************************/
```

```
        usOffset += usNumChars;
    }

    return ;

}
```

**WINWORD.MAK**

```
WINWORD.EXE:                        WINWORD.OBJ
        LINK386 @<<
WINWORD
WINWORD
WINWORD
OS2386
WINWORD
<<

WINWORD.OBJ:                        WINWORD.C
        ICC -C+ -Kb+ -Ss+ WINWORD.C
```

**WINWORD.DEF**

```
NAME WINWORD WINDOWAPI

DESCRIPTION 'Window word example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

The structure that contains the window information is defined as follows.

```
typedef struct   _WININFO
{
   ULONG            ulStructSize;
   BOOL             bStructInit;
   SHORT            sNumWindows;
   HWND             ahwndWindows[10];

} WININFO,*PWININFO
```

The first element a window word structure is the size of the structure. The operating system uses this first element if running under OS/2 2.1 or lower. The windowing functions in these versions are 16-bit, and the operating system must "thunk" the 32-bit memory pointers so the 16-bit parts of the operating system can understand the address. The operating system uses the *ulStructSize* to see if the memory chunk will straddle a 64K boundary. If a boundary is straddled, the memory chunk is placed into a new 64K segment. This requirement goes away in OS/2 Warp, but programmers must be careful if their code will run on prior OS/2 versions.

Most of the functions in this program should look familiar. The first difference to emerge is *WinRegisterClass*.

```
WinRegisterClass(habAnchor,
                 CLS_CLIENT,
                 ClientWndProc,
                 CS_SIZEREDRAW,
                 sizeof(PVOID));
```

The last parameter specifics the amount of space to set aside in the user-defined window word each time a window of this class is created. In most cases, the programmer will want to allocate a pointer to a structure that contains all the information to be carried around with the window.

Some initialization is necessary in order to utilize this space, and the best place for initialization is in the WM_CREATE message. This is the first message that will be sent to a window.

```
pWinInfo = (PWININFO)calloc(1,
                    sizeof(WININFO));
if (pWinInfo)
        WinSetWindowPtr(hwndWnd,
                        QWL_USER,
                        pWinInfo);
else
        DisplayError("No memory allocated for pWinInfo");
break;
```

The memory for the WININFO structure is allocated, and *WinSetWindowPtr* places the *pWinInfo* pointer at the window word location QWL_USER (otherwise known as offset 0).

```
pWinInfo = (PWININFO)WinQueryWindowPtr(hwndWnd,
                                       QWL_USER);
```

Instead of enumerating all the windows each time we receive a WM_PAINT message, we perform this action only the first time through and set the Boolean initialized flag, *bStructInit* in the WININFO structure, to TRUE. The next time a WM_PAINT message is received, this flag is checked; if it indicates that the initialization has been performed already, the array of window handles in the WININFO structure are used.

# Control Windows

At the heart of data input in a Presentation Manager program are many different styles of reusable controls. A control window is a window within a window designed to perform some useful behavior in a consistent manner. The controls available are listed on page 125.

# Presentation Parameters

Presentation Manager provides pretty fancy ways to set the color and font of a window—descriptors are called presentation parameters. *WinSetPresParam* and *WinQueryPresParam* are used to set and query the presentation parameters respectively.

```
BOOL WinSetPresParam(HWND hwnd,
                     ULONG id,
                     ULONG cbParam,
                     PVOID pbParam);
```

*hwnd* is the window for which to set the presentation parameters. *id* is a constant used to indicate which presentation parameter to set. These values are listed below. *cbParam* is the size of the presentation parameter data, and *pbParam* is the actual presentation parameter data. For examples setting presentation parameters, see Chapter 25, Sliders, and Chapter 26, Font and File Dialogs.

```
ULONG WinQueryPresParam(HWND hwnd,
                        ULONG id1,
                        ULONG id2,
                        PULONG pulId,
                        ULONG cbBuf,
                        PVOID pbBuf,
                        ULONG fs);
```

Again, *hwnd* is the window for which to query the presentation parameters. *id1* is the first of the presentation parameter attribute to be queried. *id2* is the second of the presentation parameter attribute to be queried. If a window contains both presentation parameter attributes, only the data for *id1* is returned. *pulId* is used on output to indicate which presentation parameter attribute was found. *cbBuf* is the size of the buffer used to hold the presentation parameter data, and *pbBuf* is the actual buffer itself. *fs* is a collection of possible query options which are OR'ed together. Table 9.10 lists the possible values.

Table 9.10 Options For Presentation Parameter Attribute Queries

| Value | Description |
|---|---|
| QPF_NOINHERIT | Presentation parameters are not inherited from the owner of window hwnd. By default, the presentation parameters are inherited. |
| QPF_ID1COLORINDEX | Indicates *id1* is a color index presentation parameter attribute, which needs to be converted to RGB before being passed back in *pbBuf*. |
| QPF_ID2COLORINDEX | Indicates *id2* is a color index presentation parameter attribute, which needs to be converted to RGB before being passed back in *pbBuf*. |
| QPF_PURERGBCOLOR | Specifies that either or both *id1* and *id2* reference an RGB color, and that these must be pure colors. |

For an example using *WinQueryPresParam*, see Chapter 26 .

```
BOOL WinRemovePresParam(HWND hwnd,
                        ULONG id);
```

*WinRemovePresParam* is used to remove a presentation parameter attribute. *hwnd* is the window to remove the presentation parameter attribute from. *id* is the id of the presentation parameter to remove. The function returns TRUE upon successful completion.

Presentation parameters can also be passed through *WinCreateWindow*. A presentation parameter has an attribute type (PP_*) and a value for the specified attribute. Table 9.11 presents valid attribute types.

Table 9.11 Attribute Types

| Attribute Type | Description | Data Type |
|---|---|---|
| PP_FOREGROUNDCOLOR | Foreground window color | RGB |
| PP_BACKGROUNDCOLOR | Background window color | RGB |
| PP_FOREGROUNDCOLORINDEX | Foreground window color | COLOR (LONG) |
| PP_BACKGROUNDCOLORINDEX | Background window color | COLOR (LONG) |
| PP_HILITEFOREGROUNDCOLOR | Highlighted foreground window color | RGB |
| PP_HILITEBACKGROUNDCOLOR | Highlighted background window color | RGB |

| Attribute Type | Description | Data Type |
|---|---|---|
| PP_FOREGROUNDCOLOR | Foreground window color | RGB |
| PP_BACKGROUNDCOLORd | Background window color | RGB |
| PP_HILITEFOREGROUNDCOLORINDEX | Highlighted foreground window color | COLOR (LONG) |
| PP_HILITEBACKGROUNDCOLORINDEX | Highlighted background window color | COLOR (LONG) |
| PP_DISABLEDFOREGROUNDCOLOR | Disabled foreground window color | RGB |
| PP_DISABLEDBACKGROUNDCOLOR | Disabled background window color | RGB |
| PP_DISABLEDFOREGROUNDCOLORINDEX | Disabled foreground window color | COLOR (LONG) |
| PP_DISABLEDBACKGROUNDCOLORINDEX | Disabled background window color | COLOR (LONG) |
| PP_BORDERCOLOR | Window border color | RGB |
| PP_BORDERCOLORINDEX | Window border color | COLOR (LONG) |
| PP_FONTNAMESIZE | Window font name and point size | PSZ |
| PP_ACTIVECOLOR | Active frame window title bar color | RGB |
| PP_ACTIVECOLORINDEX | Active frame window title bar color | COLOR (LONG) |
| PP_INACTIVECOLOR | Inactive frame window title bar color | RGB |
| PP_INACTIVECOLORINDEX | Inactive frame window title bar color | COLOR (LONG |
| PP_ACTIVETEXTFGNDCOLOR | Active frame window title bar text foreground color | RGB |
| PP_ACTIVETEXTFGNDCOLORINDEX | Active frame window title bar text foreground color | COLOR (LONG) |
| PP_ACTIVETEXTBGNDCOLOR | Active frame window title bar text background color | RGB |
| PP_ACTIVETEXTBGNDCOLORINDEX | Active frame window title bar text background color | COLOR (LONG) |
| PP_INACTIVETEXTFGNDCOLOR | Inactive frame window title bar text foreground color | RGB |
| PP_INACTIVETEXTFGNDCOLORINDEX | Inactive frame window title bar text foreground color | COLOR (LONG) |
| PP_INACTIVETEXTBGNDCOLOR | Inactive frame window title bar text background color | RGB |
| PP_INACTIVETEXTBGNDCOLORINDEX | Inactive frame window title bar text background color | COLOR (LONG) |
| PP_SHADOW | Color for shadow of certain controls | COLOR (LONG) |
| PP_MENUFOREGROUNDCOLOR | Color for menu foreground | RGB |
| PP_MENUFOREGROUNDCOLORINDEX | Color for menu foreground | COLOR (LONG) |
| PP_MENUBACKGROUNDCOLOR | Color for menu background | RGB |

| Attribute Type | Description | Data Type |
|---|---|---|
| PP_FOREGROUNDCOLOR | Foreground window color | RGB |
| PP_BACKGROUNDCOLORd | Background window color | RGB |
| PP_MENUBACKGROUNDCOLORINDEX | Color for menu background | COLOR (LONG) |
| PP_MENUHILITEFOREGROUNDCOLOR | Color for highlighted menu foreground | RGB |
| PP_MENUHILITEFOREGROUNDCOLORINDEX | Color for highlighted menu foreground | COLOR (LONG) |
| PP_MENUHILITEBACKGROUNDCOLOR | Color for highlighted menu background | RGB |
| PP_MENUHILITEBACKGROUNDCOLORINDEX | Color for highlighted menu background | COLOR (LONG) |
| PP_MENUDISABLEDFOREGROUNDCOLOR | Color for disabled menu foreground | RGB |
| PP_MENUDISABLEDFOREGROUNDCOLORINDEX | Color for disabled menu foreground | COLOR (LONG) |
| PP_MENUDISABLEDBACKGROUNDCOLOR | Color for disabled menu background | RGB |
| PP_MENUDISABLEDBACKGROUNDCOLORINDEX | Color for disabled menu background | COLOR (LONG) |

# Chapter 10

# Window Management

A window has many physical characteristics that are controlled both by the user by the programmer. These characteristics include size, visiblity, position, and order. A user can size a window by dragging the sizing border of the window; likewise, the programmer also can size the window by using a function call. A good application will not hinder the user from arranging the windows on the desktop in whatever manner he or she sees fit; however, an application also can provide the user with visual clues as to what actions can and cannot be performed. For example, a "Save" menu item may be disabled when the file is unchanged from its previous state, or a window may be inactive until the user has logged on successfully.

This chapter covers the following window characteristics:

- Visibility/invisible
- Active/inactive
- Sizing
- Z-order

The programming interfaces to change these characteristics are explained and two example programs are included: WINSAVE, a program designed to save the window characteristics at the time the application is closed, WINTRACK, a program that will maintain a minimum and maximum size requirement.

## Visible, Invisible, Enabled, and Disabled Windows

Presentation Manager supports the idea of a "messy desktop" window arrangement. This means that several windows can be stacked upon each other similar to pieces of paper on a desk. A window that is visible is one that is currently visible on the deskop or that can be uncovered by moving a window that is on top of it. An invisible window is one with the WS_VISIBLE bit not set; the programmer must make it visible before it can be seen. *WinShowWindow* can be used to make an invisible window visible.

```
BOOL WinShowWindow(HWND hwnd,
                   BOOL fShow);
```

The first parameter is the window to be made visible or invisible. A value of TRUE for the next parameter indicates the window is to be made visible. FALSE indicates the window is to be made invisible.

A window that is enabled is one that can respond to user input. An application can disable a window by using *WinEnableWindow*. Items on a dialog box can be disabled from being chosen if the choices are no longer applicable.

```
BOOL WinEnableWindow(HWND hwnd,
                     BOOL fEnable);
```

The first parameter is the window to be enabled or disabled. A value of TRUE for the next parameter indicates the window is to be enabled. FALSE indicates the window is to be disabled.

## Window Sizing

The CUA (Common User Access) guidelines recommend that a frame window let the user size and position the window to his or her own specifications. These guidelines are used to help maintain a consistent "look and feel" across all Presentation Manager applications. The CUA specifications are published by IBM and can help a user adapt more easily to a new OS/2 application.

Conveniently enough, Presentation Manager can handle most of this frame manipulation automagically. The frame control flag, FCF_SIZEBORDER, gives the frame window a "sizing border." The user can shape and size the window to his or her heart's content, and the programmer can kick back, relax, and let Presentation Manager do all the work. But (there's always a but) the programmer should make sure that the WM_PAINT message processing adapts for the change in window real estate. There are a few ways to keep track of the window size.

- In the WM_PAINT processing, callto return the RECTL structure containing the window size.
- Keep track of the window size by processing the WM_SIZE messages, and store these values in a structure pointed to by a window word.

Suppose a client area contained a graphic that the programmer wanted to be visible at all times. One option is to resize automatically the window if the user sizes the window to a smaller size. A less clumsy option is to restrict the size when the user is adjusting the size border. The following example shows just how to do this.

**WINSIZE.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  CLS_CLIENT    "MyClass"

typedef struct _FRAMEINFO
{
   LONG              lWidth;
   LONG              lHeight;
   PFNWP             pfnNormalFrameProc;
} FRAMEINFO,*PFRAMEINFO;

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

MRESULT EXPENTRY SubclassFrameProc(HWND hwndWnd,ULONG ulMsg,
                                   MPARAM mpParm1,MPARAM mpParm2);

VOID DisplayError(CHAR *pszText);

INT main(VOID)
{

   HAB               habAnchor;
```

```
HMQ             hmqQueue;
ULONG           ulFlags;
HWND            hwndFrame;
BOOL            bLoop;
QMSG            qmMsg;
PFRAMEINFO      pFrameInfo;
PFNWP           pfnNormalFrameProc;
LONG            lHeight,lWidth;

habAnchor = WinInitialize(0);
hmqQueue = WinCreateMsgQueue(habAnchor,
                             0);

WinRegisterClass(habAnchor,
                 CLS_CLIENT,
                 ClientWndProc,
                 0,
                 sizeof(PVOID));

ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
   FCF_TASKLIST;

/************************************************************/
/* create frame window                                      */
/************************************************************/

hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                               0,
                               &ulFlags,
                               CLS_CLIENT,
                               "Frame Window",
                               0,
                               NULLHANDLE,
                               0,
                               NULL);

/************************************************************/
/* subclass the frame window and point to subclass frame    */
/* proc                                                     */
/************************************************************/

pfnNormalFrameProc = WinSubclassWindow(hwndFrame,
                                       SubclassFrameProc);

/************************************************************/
/* get screen height and width                              */
/************************************************************/

lWidth = WinQuerySysValue(HWND_DESKTOP,
                          SV_CXSCREEN);

lHeight = WinQuerySysValue(HWND_DESKTOP,
                           SV_CYSCREEN);

/************************************************************/
/* if failed, display error, and set to default value       */
/************************************************************/

if (!lWidth || !lHeight)
{
   DisplayError("WinQuerySysValue failed");
   lWidth = 640;
   lHeight = 480;
}

if (hwndFrame != NULLHANDLE)
{
```

```
      /*******************************************************/
      /* allocate memory for window word                     */
      /*******************************************************/

      pFrameInfo = calloc(1,
                          sizeof(FRAMEINFO));
      if (!pFrameInfo)
      {

         /*******************************************************/
         /* error handling                                      */
         /*******************************************************/

         DisplayError("No memory allocated for frame info");
         WinDestroyWindow(hwndFrame);
         WinDestroyMsgQueue(hmqQueue);
         WinTerminate(habAnchor);
         return 0;
      }
      WinSetWindowPtr(hwndFrame,
                      QWL_USER,
                      pFrameInfo);
      pFrameInfo->lWidth = lWidth;
      pFrameInfo->lHeight = lHeight;
      pFrameInfo->pfnNormalFrameProc = pfnNormalFrameProc;

      /*******************************************************/
      /* set window position                                 */
      /*******************************************************/

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      lWidth/4,
                      lHeight/4,
                      lWidth/2,
                      lHeight/2,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile            */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif               */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {

      case  WM_ERASEBACKGROUND :
         return  MRFROMSHORT(TRUE);
```

```
      default   :
         return  WinDefWindowProc(hwndWnd,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);
   }                                   /* endswitch              */
}

MRESULT EXPENTRY SubclassFrameProc(HWND hwndFrame,ULONG ulMsg,
                                   MPARAM mpParm1,MPARAM mpParm2)
{

   PFRAMEINFO       pFrameInfo;
   PFNWP            pfnNormalFrameProc;
   CLASSINFO        classInfo;
   HAB              hab;

   /*********************************************************/
   /* retrieve frame information from frame window word     */
   /*********************************************************/

   pFrameInfo = WinQueryWindowPtr(hwndFrame,
                                  QWL_USER);
   if (!pFrameInfo)
   {

      /*******************************************************/
      /* if cannot find window word, use the default window  */
      /* proc                                                */
      /*******************************************************/

      hab = WinQueryAnchorBlock(hwndFrame);
      WinQueryClassInfo(hab,
                        WC_FRAME,
                        &classInfo);
      pfnNormalFrameProc = classInfo.pfnWindowProc;
      return ((*pfnNormalFrameProc)(hwndFrame,
                                    ulMsg,
                                    mpParm1,
                                    mpParm2));
   }
   pfnNormalFrameProc = pFrameInfo->pfnNormalFrameProc;

   switch (ulMsg)
   {
      case  WM_QUERYTRACKINFO :
         {
            PTRACKINFO       pTrackInfo;
            MRESULT          mrReply;

            /*************************************************/
            /* call default procedure first to get TRACKINFO */
            /* structure                                     */
            /*************************************************/

            mrReply = (*pfnNormalFrameProc)(hwndFrame,
                                            ulMsg,
                                            mpParm1,
                                            mpParm2);
            pTrackInfo = (PTRACKINFO)PVOIDFROMMP(mpParm2);

            /*************************************************/
            /* set limits at 1/2 the height and width of screen*/
            /*************************************************/

            pTrackInfo->ptlMinTrackSize.x = pFrameInfo->lWidth/2;
            pTrackInfo->ptlMinTrackSize.y = pFrameInfo->lHeight/2
               ;
```

```
            /*************************************************/
            /* put limits in TRACKINFO structure             */
            /*************************************************/

            return  mrReply;

         }
      case  WM_DESTROY :
         {

            /*************************************************/
            /* clean up                                      */
            /*************************************************/

            if (pFrameInfo)
               free(pFrameInfo);
            break;
         }
      default  :
         break;

   }                                    /* endswitch         */

   /*************************************************************/
   /* return normal frame window procedure                      */
   /*************************************************************/

   return ((*pfnNormalFrameProc)(hwndFrame,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2));

}

VOID DisplayError(CHAR *pszText)
{

   /*************************************************************/
   /* small function to display error string                    */
   /*************************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}
```

**WINSIZE.MAK**

```
WINSIZE.EXE:                    WINSIZE.OBJ
        LINK386 @<<
WINSIZE
WINSIZE
WINSIZE
OS2386
WINSIZE
<<

WINSIZE.OBJ:                    WINSIZE.C
        ICC -C+ -Kb+ -Ss+ WINSIZE.C
```

**WINSIZE.DEF**

```
NAME WINSIZE WINDOWAPI

DESCRIPTION 'Window sizing example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

## Device Independence, Almost

One new feature will be added to *main* in this example—a mini form of device independence. SVGA is very popular, and supporting both 1024 x 768 and 640 x 480 screen resolutions in your programs can be quite painful. Unfortunately, Presentation Manager does not guarantee that your programs will be dimensioned proportionally at both resolutions. The best way to make your program look great at any resolution is to size your windows according to the screen size. "But how will I know how big the screen is?" you may ask. The answer: Presentation Manager knows all, and you just have to know which questions to ask. *WinQuerySysValue* is used for exactly that reason.

```
LONG WinQuerySysValue(HWND hwndDesktop,
                      LONG iSysValue);
```

*hwndDesktop* is the desktop window handle, and *iSysValue* is a constant used to query a specific value. The constants available are too numerous to list here, but are listed in the documentation for *WinQuerySysValue*.

```
lHeight = WinQuerySysValue( HWND_DESKTOP, SV_CYSCREEN );
lWidth  = WinQuerySysValue( HWND_DESKTOP, SV_CXSCREEN );
```

This function will provide lots of information about the dimensions of various system components. The values we are interested in are the height and width of the screen, SV_CXSCREEN and SV_CYSCREEN. The value returned from the function is the answer to your query.

Once we know the screen height and width, we use *WinSetWindowPos* to size and position the window accordingly.

## Subclassing the Frame Window

```
pfnNormalFrameProc = WinSubclassWindow(hwndFrame,
                                       SubclassFrameProc);
WinSetWindowPtr(hwndFrame,
                QWL_USER,
                pFrameInfo);
pFrameInfo->lWidth = lWidth;
pFrameInfo->lHeight = lHeight;
pFrameInfo->pfnNormalFrameProc = pfnNormalFrameProc;
```

The frame window must be subclassed in order to alter the default frame window behavior. For more information on subclassing, see Chapter 27. In this program, the old (in this case, the default) frame window procedure is saved, along with the minimum height and width, in the frame window word. Window words were covered in Chapter 9. Remember, some of the system control windows, such as the frame windows, have reserved space for a user-defined window word.

## In Case of Error, Use the Class Default

```
hab = WinQueryAnchorBlock(hwndFrame);
WinQueryClassInfo(hab,
                  WC_FRAME,
                  &classInfo);
pfnNormalFrameProc = classInfo.pfnWindowProc;
return ((*pfnNormalFrameProc)(hwndFrame,
                              ulMsg,
                              mpParm1,
                              mpParm2));
```

In case the window pointer is not found, the frame resorts back to its old window procedure. The path to the old window procedure is found by using two very useful functions, *WinQueryAnchorBlock* and *WinQueryClassInfo*.

```
HAB WinQueryAnchorBlock(HWND hwnd);
```

*WinQueryAnchorBlock* has only one parameter, the window handle of the window for which to retrieve the anchor block handle. The function returns the handle to the anchor block.

```
BOOL WinQueryClassInfo(HAB hab, PSZ pszClassName,
                       PCLASSINFO pClassInfo);
```

This function has three parameters. *hab* is the anchor block handle, *pszClassName* is the name of the class for which to retrieve the information, and *pClassInfo* is a pointer to a CLASSINFO structure.

```
typedef struct _CLASSINFO
   {
      ULONG   flClassStyle;
      PFNWP   pfnWindowProc;
      ULONG   cbWindowData;
   } CLASSINFO;
   typedef CLASSINFO *PCLASSINFO;
```

The structure contains the class style flags, *flClassStyle*. A pointer to the window procedure, *pfnWindowProc*, and also the number of additional window words, *cbWindowData*.

*WinQueryAnchorBlock* is used to retrieve the anchor block for our message queue. Once we have the anchor block handle, *WinQueryClassInfo* is called to retrieve the default window procedure for the frame class. Then, this window procedure is executed rather than the subclassed frame window procedure.

## Tracking the Frame

The WM_TRACKFRAME message controls the sizing of the frame. This message is sent from the title bar to the frame window. When the frame window receives this message, it sends a WM_QUERYTRACKINFO message to itself to query the TRACKINFO structure, which is used to define the boundaries of the tracking (moving or sizing) operation. What the example program does is intercept the WM_QUERYTRACKINFO message, fill in the TRACKINFO structure, modify the tracking values that we want to limit, and return TRUE to let the tracking operation continue. The TRACKINFO structure looks like this.

```
typedef struct _TRACKINFO    /* ti */
   {
      LONG    cxBorder;
      LONG    cyBorder;
      LONG    cxGrid;
      LONG    cyGrid;
      LONG    cxKeyboard;
      LONG    cyKeyboard;
      RECTL   rclTrack;
      RECTL   rclBoundary;
      POINTL  ptlMinTrackSize;
      POINTL  ptlMaxTrackSize;
      ULONG   fs;
   } TRACKINFO;
   typedef TRACKINFO *PTRACKINFO;


mrReply = (*pfnNormalFrameProc)(hwndFrame,
                                ulMsg,
                                mpParm1,
                                mpParm2);
pTrackInfo = (PTRACKINFO)PVOIDFROMMP(mpParm2);
```

The default frame window procedure is called in order to get a TRACKINFO structure that is already filled in.

```
pTrackInfo->ptlMinTrackSize.x = pFrameInfo->lWidth/2;
pTrackInfo->ptlMinTrackSize.y = pFrameInfo->lHeight/2;
```

Once we have this structure, we modify the *ptlMinTrackSize.x* and *ptlMinTrackSize.y* values. We use one-half the screen width and one-half the screen height as the new minimum tracking sizes. The last step is to return *mrReply* which will be TRUE in all cases, except for errors.

## Saving Window Settings

Now we're ready to expand a little beyond the basic Presentation Manager program. When the user closes down an application, it is only polite to remember all the changes he or she has made to the frame window. In OS/2 2.0, the developers added two new functions to make it super-easy really to impress your customers—*WinStoreWindowPos* and *WinRestoreWindowPos*. These functions store the window size, position, and presentation parameters in the OS2.INI file and then retrieve them on demand.

**WINSAVE.C**

```
#define INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define CLS_CLIENT              "MyClass"

#define SAVE_NAME               "WINSAVE"
#define SAVE_KEY                "WINDOWPOS"

MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
                                 ULONG ulMsg,
                                 MPARAM mpParm1,
                                 MPARAM mpParm2 ) ;

INT main ( VOID )
{
   HAB         habAnchor ;
```

```
HMQ         hmqQueue ;
ULONG       ulFlags ;
HWND        hwndFrame ;
HWND        hwndClient ;
BOOL        bReturn ;
BOOL        bLoop ;
QMSG        qmMsg ;

habAnchor = WinInitialize ( 0 ) ;
hmqQueue = WinCreateMsgQueue ( habAnchor, 0 ) ;

WinRegisterClass ( habAnchor,
                   CLS_CLIENT,
                   ClientWndProc,
                   0,
                   0 ) ;

ulFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER |
          FCF_MINMAX | FCF_TASKLIST ;

hwndFrame = WinCreateStdWindow ( HWND_DESKTOP,
                                 WS_VISIBLE,
                                 &ulFlags,
                                 CLS_CLIENT,
                                 "Titlebar",
                                 0L,
                                 NULLHANDLE,
                                 0,
                                 &hwndClient ) ;

if ( hwndFrame != NULLHANDLE ) {
   bReturn = WinRestoreWindowPos ( SAVE_NAME,
                                   SAVE_KEY,
                                   hwndFrame ) ;

   if ( bReturn ) {
      WinSetWindowPos ( hwndFrame,
                        HWND_TOP,
                        0,
                        0,
                        0,
                        0,
                        SWP_ACTIVATE | SWP_SHOW ) ;
   } else {
      WinSetWindowPos ( hwndFrame,
                        NULLHANDLE,
                        10,
                        10,
                        400,
                        300,
                        SWP_ACTIVATE |
                        SWP_MOVE |
                        SWP_SIZE |
                        SWP_SHOW ) ;
   } /* endif */

   bLoop = WinGetMsg ( habAnchor,
                       &qmMsg,
                       NULLHANDLE,
                       0,
                       0 ) ;

   while ( bLoop ) {
      WinDispatchMsg ( habAnchor, &qmMsg ) ;
      bLoop = WinGetMsg ( habAnchor,
                          &qmMsg,
                          NULLHANDLE,
                          0,
                          0 ) ;
```

```
      } /* endwhile */

      WinDestroyWindow ( hwndFrame ) ;
   } /* endif */

   WinDestroyMsgQueue ( hmqQueue ) ;
   WinTerminate ( habAnchor ) ;
   return 0 ;
}

MRESULT EXPENTRY ClientWndProc ( HWND hwndWnd,
                                 ULONG ulMsg,
                                 MPARAM mpParm1,
                                 MPARAM mpParm2 )
{
   switch ( ulMsg ) {

   case WM_ERASEBACKGROUND:
      return MRFROMSHORT ( TRUE ) ;

   case WM_SAVEAPPLICATION:
      WinStoreWindowPos ( SAVE_NAME,
                          SAVE_KEY,
                          WinQueryWindow ( hwndWnd, QW_PARENT )) ;
      break ;

   default:
      return WinDefWindowProc ( hwndWnd,
                                ulMsg,
                                mpParm1,
                                mpParm2 ) ;
   } /* endswitch */

   return MRFROMSHORT ( FALSE ) ;
}
```

**WINSAVE.MAK**

```
WINSAVE.EXE:                    WINSAVE.OBJ
        LINK386 @<<
WINSAVE
WINSAVE
WINSAVE
OS2386
WINSAVE
<<

WINSAVE.OBJ:                    WINSAVE.C
        ICC -C+ -Kb+ -Ss+ WINSAVE.C
```

**WINSAVE.DEF**

```
NAME WINSAVE WINDOWAPI

DESCRIPTION 'WinRestoreWindowPos example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

## *WinRestoreWindowPos*

```
      bReturn = WinRestoreWindowPos ( SAVE_NAME,
                                      SAVE_KEY,
                                      hwndFrame ) ;
```

*WinRestoreWindowPos* is called right after the frame window is created. This enables the saved changes to be visible right when the window is created.

```
BOOL APIENTRY WinRestoreWindowPos(PSZ pszAppName,
                                  PSZ pszKeyName,
                                  HWND hwnd);
```

The first parameter is the application name, placed in the .INI file. The second is the keyword used in conjunction with the application name. The last parameter is the window to apply the changes to.

If the call completes successfully, *WinSetWindowPos* will make the window visible and make it the active window.

```
BOOL WinSetWindowPos(HWND hwnd,
                     HWND hwndInsertBehind,
                     LONG x,
                     LONG y,
                     LONG cx,
                     LONG cy,
                     ULONG fl);
```

*WinSetWindowPos* is a very handy function. It is used to position, size, activate, deactivate, maximize, minimize, hide, or restore a window. One of the nice aspects of *WinSetWindowPos* is its ability to consolidate several function calls into one.

```
WinSetWindowPos ( hwndFrame,
                  HWND_TOP,
                  0,
                  0,
                  0,
                  0,
                  SWP_ACTIVATE | SWP_SHOW ) ;
```

*hwndFrame* is the window to adjust. The next parameter, HWND_TOP, indicates the position in the Z-order for the window. We've mentioned Z-order before; it's time for a little more detail.

# X,Y,Z-Order

Presentation Manager supports a concept of piling windows (visually) one on top of another, known as Z-order. The active window and its children are always at the top of the Z-order. Children are ahead of their parents in their position in the Z-order. The window that is at the top of the Z-order is one in which the user inputs keystrokes and mouse moves.

The next four parameters of *WinSetWindowPos* are the x coordinate, y coordinate, width, and height of the window. The last parameter is the value of the action flags OR'ed together. If SWP_MOVE is specified, the x, y coordinates are used to move the window to the requested position; if not, these two parameters are ignored. If SWP_SIZE is used, the window is resized to the new height and width; if not, these two parameters are ignored. We'll use SWP_ACTIVATE and SWP_SHOW to show the window, and also to make the frame window the active one.

Readers may wonder why they call these flags SWP_. The reason is that a structure used in window positioning is a SWP (or "set window position") structure. The structure is as follows.

```
typedef struct _SWP
{
   ULONG    fl;
   LONG     cy;
   LONG     cx;
   LONG     y;
   LONG     x;
   HWND     hwndInsertBehind;
   HWND     hwnd;
   ULONG    ulReserved1;
   ULONG    ulReserved2;
} SWP;
typedef SWP *PSWP;
```

After calling *WinRestoreWindowPos*, either *WinShowWindow* or *WinSetWindowPos* with the SWP_SHOW flag should be called.

## Saving State

```
case WM_SAVEAPPLICATION:
   WinStoreWindowPos ( SAVE_NAME,
                       SAVE_KEY,
                       WinQueryWindow ( hwndWnd, QW_PARENT )) ;
   break ;
```

Presentation Manager sends a special message at application shutdown time for the sole purpose of giving the programmer a chance to save the options and settings the user has customized to reflect his or her preferences. This is the WM_SAVEAPPLICATION message. Catchy name. This is the time to call *WinStoreWindowPos*.

```
BOOL APIENTRY WinStoreWindowPos(PSZ pszAppName,
                                PSZ pszKeyName,
                                HWND hwnd);
```

The parameters for this function are *exactly* the same as *WinRestoreWindowPos*.

**Gotcha!**
One little note here: The settings for the frame window, not the client, are the ones to be retrieved.

# Chapter 11

# Window Messages and Queues

Presentation Manager windows communicate using a queue message processing system. All windows in a Presentation Manager thread share a single message queue for processing messages; however, all message queues are descendants of the Presentation Manager system message queue. This is the reason that one poorly designed Presentation Manager application can freeze up the entire system The queueing mechanism is a very important concept to understand.

Once a window has a message queue, it can communicate with any other window in the entire system. All it needs is the window, or message queue, handle to send the message to.

A window can send or receive messages. Each message is used to signal some sort of event. Each time a mouse is moved, a window is resized, or a menu item is selected, messages are sent to a window. A window procedure operates like a massive sieve, filtering the messages of interest and passing through those messages that are unimportant. It is important to realize that all messages must be processed and replied to, either through your own window procedure or by passing the message to *WinDefWindowProc* or *WinDefDlgProc*. This facility of using events to control the programming flow is known as "event-driven programming." This style is common not only to Presentation Manager programming, but to other GUI programming environments as well.

## Message Ordering

It is not a good idea to count on messages arriving in your message queue in a certain order; the purpose of event-driven programming is to be flexible and dynamic and respond only when asked; however, there are obviously times when it is important to understand the flow of messages the system sends to your queue.

The first message you can count on being sent to your client window is the WM_CREATE message. At the time this message arrives, the window handle exists, but has no size and is not visible. The WM_CREATE message can be used to do some application-specific initialization, for instance, allocating memory for window words; however, any queries specific to size or focus should be done after creation. One way to accomplish this is by posting a user- defined message to the client window in the WM_CREATE processing. The size and focus messages the system places in the queue are sent messages, and will be processed before a posted message. When you process the user-defined message, you will have a client area that has both size and focus, and this information can be used in any initialization that needs to be done.

The standard way to set size or focus is by using the respective API's directly after the *WinCreateStdWindow* or *WinCreateWindow* call. If you would like to change the size or position of a window, there are two ways to do this. First, create the client window as not visible, and use the function *WinSetWindowPos* to size and show the window. The second method is to intercept the WM_ADJUSTWINDOWPOS message. This message is sent before a window has been sized or moved. This gives the application a chance to override the new size and position with a size and position of its own choosing. If modifications are made, the application should return TRUE instead of FALSE, and the new coordinates are used.

## Focus Messages

When a window is gaining or losing focus, there are several messages that are sent by the system. It is not advisable to process any of these messages yourself, but it is useful to understand how Presentation Manager handles changing a windows focus.

When a user clicks the mouse on another window, the system first sends a set of messages to the window that is losing the focus. A set of WM_QUERYFOCUSCHAIN messages are sent to the frame window and its children to help the system decide which windows will be involved in this focus change operation. Next, a WM_FOCUSCHANGE message is sent to both the frame and its children to indicate they are all losing focus. The next message sent is the WM_SETFOCUS message. This message indicates the window is either about to lose or about to gain the input focus. In this case, it would be losing input focus. Next, the WM_SETSELECTION message is sent This message is used to unhighlight or highlight any selected items in the window. The client area does not do much with this message, but it is at this time that the titlebar window changes from a highlighted titlebar to an unhighlighted titlebar. The last message sent when a window is losing focus is the WM_ACTIVATE message. The message actually takes away the focus from the active window.

When a window is gaining focus, the messages are sent in a similar fashion. First, the system queries the windows with the WM_QUERYFOCUSCHAIN. Then, a WM_FOCUSCHANGE message is sent to the frame and its children to indicate they are gaining focus. Next, the focus change operations are actually performed, with a WM_SETFOCUS being sent first, then the WM_SETSELECTION, and lastly, the WM_ACTIVATE.

## Size and Paint Messages

An application receives three messages when a window is sized, WM_CALCVALIDRECTS, WM_SIZE, and then WM_PAINT. The message WM_CALCVALIDRECTS is used to communicate the new window size and coordinates after the sizing operation. The WM_CALCVALIDRECTS is used only when CS_SIZEREDRAW style is not specified, as the whole window will be invalidated when a sizing operation is done on a window with this style.

The next message is the WM_SIZE message. This message gives the application a chance to reposition any other window that may be dependent on the newly sized windows position. The last message passed, if the style CS_SIZEREDRAW is set, is the WM_PAINT. If the WS_SYNCPAINT style is set, the message will be sent, otherwise the message will be posted. The system will pass the rectangular coordinates that contain the area to be redrawn as a parameter in the WM_PAINT message.

## The Last Messages a Window Receives

When a WM_CLOSE message is posted to a window (when the user selects CLOSE from the system menu), first a WM_SYSCOMMAND message is posted with the SC_CLOSE ID. Next, a WM_QUIT message is posted to the message queue. This is a very special message, because when *WinGetMsg* receives this message, the function returns FALSE, causing the *WinGetMsg / WinDispatchMsg* loop to terminate. A WM_SAVEAPPLICATION message is posted next. This gives the application a chance to prompt the user for any last minute clean-up work; for instance, saving a file, or disconnecting a communication line. When *WinDestroyWindow* is used to destroy the frame window, the system will send the focus change messages to indicate this frame window and all its children will be losing focus. The last message a window will receive is the WM_DESTROY. This is the place to control any application-specific cleanup. For example, freeing memory should be done in the WM_DESTROY processing.

When the user has selected "Shutdown" from the desktop menu or the Warp Launchpad, there is a little change in the messages that arrive in the queue. The system bypasses the WM_CLOSE message, and sends two messages to each thread that contains a message queue. The first is the WM_SAVEAPPLICATION. The next message issued is the WM_QUIT message. An application will usually not process the WM_QUIT message; however, in the case when it needs to interrupt or halt system shutdown, it must process WM_QUIT. If the application wants to cancel the shutdown, it can call *WinCancelShutdown*. If the application would like to do something else before shutting down, it can perform its closing work, and then call *WinDestroyMsgQueue*. After processing, make sure you return FALSE implicitely, and do not call *WinDefWindowProc* as the default window procedure does not know how to handle a WM_QUIT message .

**Gotcha!**
For each thread that contains a message queue, make absolutely sure that you issue a *WinCancelShutdown* soon after the thread is created if you do not want to process the WM_QUIT, or else be prepared to process the WM_QUIT message and destroy the message queue. A thread with a never-ending message queue can prevent the entire system from shutting down properly. Also, there is no guarantee that a secondary thread will execute all function calls and return before the primary thread (and thus the application) exits. It is up to the developer to make sure all clean-up in secondary threads is complete before the application exits.

## Sending Messages

When a message is sent, it is usually directed to a particular window. For instance, a WM_CHAR message, indicating a key had been pressed, would be sent to the window that was currently active and had the keyboard focus. There are two ways a message can be dispatched. They can either be sent, using *WinSendMsg*, or posted, using *WinPostMsg*. There is a very subtle difference between these two dispatch methods, and this could cause you problems somewhere down the road. When a message is sent, it is not put in a window's message queue; it is processed the next time *WinGetMsg* is called, or immediately executed, if no message is currently being processed. The thread containing *WinSendMsg* blocks, and control is switched over to the thread containing the receiving message's window procedure.

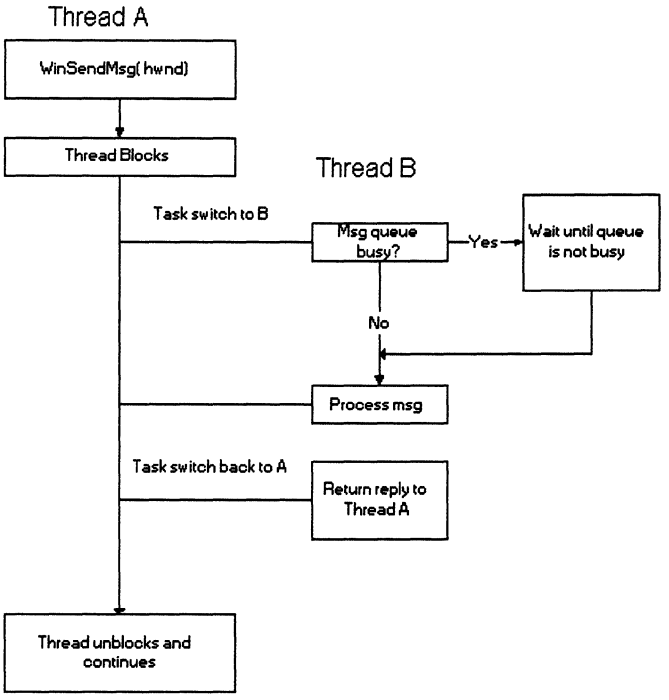



Figure 11.1 *WinSendMsg* in a multithreaded application.

A message should be sent when it absolutely, positively, has to be there right now. A good example of this is passing pointers in messages when there is no guarantee that the pointer will point to something valid when the message is up for processing. *WinSendMsg* should be used in this situation.

One little bit of information about *WinSendMsg*: this function will not return until that message has been processed. Yup, that's right. If you send a message from your window procedure to a window procedure that's asleep at the wheel, or even just a little slow to respond, your window procedure will sit there and wait until it gets some response back from the other window procedure. If you send a message to some window that the system controls the window procedure for, you can pretty much guarantee a zippy response; however, be very careful when using this function to send messages to either your own window procedure or to some other application's window procedure. *WinPostMsg* is a much safer method of transmitting messages; however, the message is placed into the receiving window's message queue. It will be processed when that window gets around to it. *WinPostMsg* should be used when you want to communicate some information and do not care about a reply. *WinSendMsg* should be used when it is imperative that you gain some piece of information and have to respond to it now.

# Broadcasting Messages

A window can communicate one to one with another window directly, or it can broadcast a message to several windows at once. The function

```
WinBroadcastMsg( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2, ULONG ulCmd )
```

can be used to send or post a message to the windows specified in the *ulCmd* parameter. This command contains two parts: who to communicate with, and what form of communication to use. These flags are

then ORed together. The default communication form is BMSG_POST. You can specify BMSG_POST, BMSG_SEND, or BMSG_POSTQUEUE. The POSTQUEUE flag will post a message to all threads in the system that have a message queue, and the *hwnd* parameter will be ignored. Only one of these three flags can be specified. The second part of the *ulCmd* parameter indicates who to communicate with. The choices are BMSG_DESCENDANTS, or BMSG_FRAMEONLY. DESCENDANTS will communicate with *hwnd*, and all of its descendants. FRAMEONLY will broadcast a message to all frame windows that are descendants of *hwnd*. To broadcast to all frames in the system use HWND_DESKTOP for *hwnd*.

## Peeking into the Message Queue

There are many instances when you do not want to retrieve a message from the message queue, instead you would rather just "peek" into the queue, and see if a message is waiting. The function:

```
WinPeekMsg( HAB hab, PQMSG pqmsg, HWND hwnd, ULONG ulFirst, ULONG ulLast, ULONG
ulOptions)
```

inspects the message queue and returns back information about the queue. hwnd narrows the search to a specific window or its children. The *ulFirst* and *ulLast* parameters let you narrow the search even further to a numerical range. If both these parameters are null, all messages are included in the search. The *ulOptions* flag indicates whether the message is removed from the queue, or not. The default is to not remove the message from the queue. The return from this function indicates whether the search was successful, or not.

## Finding More Message Queue Information

There are several functions to query information from the message queue. The following are these query functions:

| Function | Description |
|---|---|
| *WinQueryMsgPos* | Returns the pointer position when the last message retrieved from the queue was posted. This is the *ptl* parameter in the QMSG structure. |
| *WinQueryMsgTime* | Returns the time in milliseconds when the last message retrieved from the queue was posted. This is the *time* parameter in the QMSG structure. |
| *WinQueryQueueInfo* | Returns the MQINFO structure. This structure includes the process ID, thread ID, and message count |
| *WinQueryQueueStatus* | Returns information about what types of messages are in the queue |

## Message Priorities

When messages are retrieved from the message queue, they are not necessarily retrieved on a "first in, first out" basis. Instead, messages are retrieved on the basis of priority, similar to threads. The following is a list of messages in the order they will be retrieved:

- Sent messages
- WM_SEM1
- All other posted messages
- Keyboard or mouse messages

- WM_SEM2
- WM_PAINT
- WM_SEM3
- WM_TIMER
- WM_SEM4

Figure 11.2 represents a flow chart of how messages are retrieved.

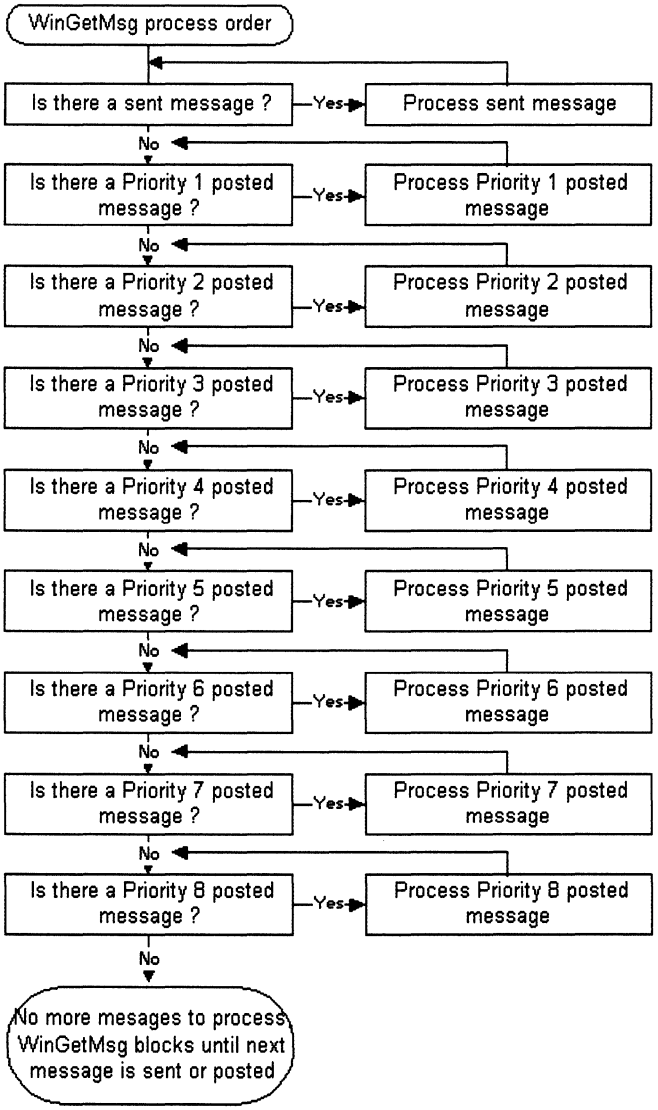



Figure 11.2 WinGetMsg message processing order

You may be wondering, "What are these WM_SEM messages, and the WM_TIMER message?" Well, on to the next topic... WM_PAINT messages are fairly low on the message priority totem pole. The default window style causes Presentation Manager to "group" invalidated regions together and generate one WM_PAINT message. The window style, WS_SYNCPAINT, or the class style CS_SYNCPAINT, will stop Presentation Manager from behaving in this independent manner, and each time a region is

invalidated, Presentation Manager will very obediently call the WM_PAINT processing immediately by sending the WM_PAINT message. The system does not post this messages, it jumps to the WM_PAINT processing, and then, when painting is completed, jumps back to the call following the region invalidation.

## Messages and Synchronization of Events

Often an application wants to know when some event has occurred. One way to do this is the use of the WM_SEM1,2,3,4 messages. These messages are totally for your application use. If these messages are passed to *WinDefWindowProc* or *WinDefDlgProc*, it has no effect on the system. For example, suppose you have a worker thread that has finished processing. That thread could post a WM_SEM2 to the main thread to indicate that the thread has finished its work. WM_SEM1 messages should really be reserved for very important, time-critical events.

A way to keep track of an event that is dependent on some function of time is to use the functions *WinStartTimer* and *WinStopTimer*. *WinStartTimer* starts an alarm clock that is set to go off after some application- defined amount of time, in milliseconds. When the timer goes off, the system sends a WM_TIMER message back to your window procedure.

You might consider using semaphores in a window procedure. DON'T!! Instead, think of using *WinRequestMutexSem*, *WinWaitEventSem*, or *WinWaitMuxWaitSem*. Waiting on a semaphore using the regular *DosWait...Sem* functions can bring a window procedure to a screeching halt. Even the most well-behaved semaphore synchronization can develop a mind of its own every now and then. The special set of window semaphore functions were created to provide the same functionality as the *DosWait...Sem* functions, but not to interrupt the flow of your window procedure completely. The system will appear to wait in the message processing that this function is called from, but messages **sent** to the message queue will be processed synchronously. When the semaphore has been posted or the function times out, the message processing resumes where the *WinWait...*call was executed. Note that messages that are posted will remain in the message queue until after the *WinWait...* call has completed.

## User-Defined Messages

Presentation Manager also gives you the flexibility to add your own messages to the system. These are called user-defined messages, and are numerically represented by the range 0x1000 through 0xBFFF. There are some system-defined messages that fall into this range, WM_USER+40 through WM_USER+55. This is an area that may change in the future, so its a good idea to search through the Toolkit header files to see if there are any new messages defined that fall into this range. Several examples in this book use user-defined messages.

# Chapter 12

# Resources

Although *resources* such as CPU time and memory in the traditional sense are viewed as "things" that need to be shared, the term has a different meaning in a GUI environment. In a Presentation Manager environment resources are viewed as items that are necessary for the user interface of an application but not part of the application code itself.

So, why does this book contain a chapter dedicated to resources if they aren't code-related? The operative phrase in the preceding paragraph is "necessary for the user interface." Resources are not something that can be done without. Instead, programmers will spend a large amount of time on "developing" resources, since they define the look of the resulting application (though not its operation).

This chapter discusses the following types of resources, what they are, and how they are used within an application: pointers, icons, bitmaps, string tables, accelerator tables, and application-defined resources. Help tables, dialog boxes, and menus are also resources that are discussed briefly, with cross-references to chapters on these topics provided. Fonts, which are the other resource type defined, will not be discussed because their use requires a detailed look at the Graphics Programming Interface (Gpi), which is a book in itself.

## More About Resources, I Would Know

In an orchestra, there are the musicians, the conductor, and the seating arrangement, which allows the conductor to know exactly where everything can be found. In this chapter, we will look the analagous parts in a PM application:

- The resources (musicians)
- The application (the conductor)
- The resource file (the seating arrangement)

The resources are the actual user interface items that are used by the application—pointers, menus, and so on; as in an orchestra, without the resources themselves, the rest is pointless. The application coordinates the use of the resources to get a meaningful result; it wouldn't make sense, for example, to show an "Open" dialog when the user requested that the document should be printed. The resource file is where the compiler is instructed which resources the application will use; these resources, as we will show, are appended to the executable in a separate area (called *resource segments*), which are analogous to the seats in the orchestra section.

Table 12.1 shows the types of resources, defined by OS/2, that we look at in this chapter.

Table 12.1 Resource Types

| Resource | Description |
|---|---|
| Pointer | Pointer or icon data |
| Bitmap | Bitmap data |
| String table | Table of strings |
| Accelerator table | Table of "shortcut" keys |
| Menu | Menu description |
| Dialog | Dialog description |
| Font | Font description |
| Help table | Table of frame windows and dialogs for which online help is to be provided |
| Help subtable | Table of windows within a frame window or dialog for which online help is to be provided |
| User data | Data in an application-specific format |

All resources are defined using *resource identifiers*, numeric constants that, together with the type of the resource being referenced, uniquely identify each resource in an application. A resource is said to be *loaded* when an application needs to use it for the first time; this loading of the resource results in a *handle* that the application uses when it calls a Presentation Manager function.

## Resource Files

But before we can look at the resources themselves, we must first look at the place in which they are specified and the compiler used to append them to the executable. The resource file usually has a main file with the extension .RC, and this file usually includes one or more dialog definition files with the extension .DLG. The resource file can include C header files using the *#include* keyword and also can include comments according to the C++ standard (i.e., using "/*" and "*/" or using "//"). Where a construct requires a BEGIN and END keywords, the symbols "{" and "}" also may be used.

Dialog files are included in a funny manner: The main file uses the keyword *DLGINCLUDE* to specify that a dialog file is to be included.

```
DLGINCLUDE resid filename
```

*resid* specifies the resource identifier of the file (!) and *filename* is the name of the file to be included. The original intent was that each dialog definition would go in a separate file and all of the files would be included by the main file.

**Gotcha!**

Because the original purpose of the dialog file is as described, each *DLGINCLUDE* statement must have a unique resource identifier. It is not necessary, however, to limit each dialog file to having a single dialog box definition.

As if that weren't enough trouble, each dialog file also must use the *RCINCLUDE* statement to specify what the main file is to which it is being attached. This is to allow the dialog file to access the symbolic definitions (i.e., *#defines*).

As was said, dialog files are included in a funny manner, and the logic is rather illogical. This process was not followed in every PM sample presentated in this book; instead, all of the dialog definitions were moved from the dialog files into the main file to eliminate the confusion of resource identifiers for files.

## Using the Resource Compiler

Now that a resource file is defined, it needs to be compiled into a .RES file. This is accomplished using the *resource compiler* (RC.EXE). The compiler comes with the base operating system and can be found in the \OS2 directory. It also comes with the Programmer's Toolkit. It supports the command-line options listed in Table 12.2.

Table 12.2 Resource Compiler Switches

| Option | Description |
|---|---|
| -D*name*[=*value*] | Defines a macro and optionally a value |
| -I*path* | Specifies a path to include when searching for #include files |
| -R | Do not attach the compiled .RES file to the .EXE or .DLL |
| -P | Do not allow resources to cross 64K boundaries |
| -K*codepage* | DBCS code page number |
| -X(*1*\|*2*) | Compress resources using one of two algorithms |
| -C*countrycode* | Specifies the country code |
| -H | Displays help |

To compile a resource file, MYAPP.RC, to a .RES file without attaching MYAPP.RES to MYAPP.EXE, compressing resources, and using "." as a directory to search, the following code would be entered:

```
RC -R -X1 -I. MYAPP.RC
```

## Pointers and Icons

Pointers and icons are defined and accessed in the same manner. This isn't coincidence; with the exception of the first two bytes in the file containing the actual data, the two are identical. Both resources are defined in the resource file in the following manner:

```
POINTER resid filename
```

*resid* is the resource identifier of the pointer or icon, and *filename* is the name of the file containing the pointer or icon data. These files are created using the "icon editor" utility (ICONEDIT.EXE), which is provided by OS/2 and also can be found as part of the Programmer's Toolkit. For help on using the icon editor, programmers should refer to the online documentation.

In a program, both are loaded using the *WinLoadPointer* function.

```
HPOINTER WinLoadPointer(HWND hwndDesktop,
                        HMODULE hmDll,
                        ULONG ulId);
```

*hwndDesktop* is the desktop window handle, for which HWND_DESKTOP can be specified. *hmDll* is the handle to a DLL that was loaded with *DosLoadModule* or *WinLoadLibrary* to which the resource is attached. If the resources are appended to the executable, then NULLHANDLE should be used for this

parameter. *ulId* is the resource identifier of the pointer or icon to be loaded. This function returns a handle to the pointer or icon that was loaded, which is used in subsequent functions that act upon pointers or icons.

Once a pointer or icon is loaded, it can be drawn in a window with the *WinDrawPointer* function.

```
BOOL WinDrawPointer(HPS hpsWnd,
                    LONG lX,
                    LONG lY,
                    HPOINTER hpPointer,
                    ULONG ulFlags);
```

*hpsWnd* is a handle to the presentation space in which the pointer or icon is to be drawn. *lX* and *lY* specify the position within the presentation space where the pointer or icon is to be drawn. *hpPointer* specifies the handle of the pointer or icon that is to be drawn. *ulFlags* specifies how the pointer or icon is to be drawn, and is one of the constants listed in Table 12.3.

Table 12.3 Values for *ulFlags*

| Constant | Description |
|---|---|
| DP_NORMAL | Draw the pointer or icon in the "normal" manner. |
| DP_HALFTONED | Draw the pointer or icon in a halftone manner. |
| DP_INVERTED | Draw the pointer or icon in color-inverted state. |

This function returns a flag indicating success or failure.

The *WinDrawPointer* function is useful for drawing an icon in a window, but it cannot be used to set the mouse pointer to anything. To accomplish this, we instead need the *WinSetPointer* function.

```
BOOL WinSetPointer(HWND hwndDesktop,
                   HPOINTER hpNew);
```

*hwndDesktop* is the handle to the desktop; again, the HWND_DESKTOP constant for this can be specified. *hpNew* is the handle to the pointer to which one wishes the mouse pointer to change. This function also returns a flag indicating success or failure.

**Gotcha!**

Just because the mouse is set to a specified pointer doesn't mean that something else cannot set it to something else. In fact, *WinDefWindowProc* will set the pointer to the arrow pointer within its processing for the WM_MOUSEMOVE message. Typically, the application would intercept the WM_MOUSEMOVE message and call *WinSetPointer* at that point to change the mouse pointer and not call *WinDefWindowProc*.

In addition to any user-drawn pointers or icons, Presentation Manager defines a number of "system pointers;" the arrow pointer, the waiting pointer, and some icons that have been discussed come from here. These pointers and icons can be accessed or reloaded using the *WinQuerySysPointer* function.

```
HPOINTER WinQuerySysPointer(HWND hwndDesktop,
                            LONG lPtr,
                            BOOL bLoad);
```

*hwndDesktop* is the desktop handle (HWND_DESKTOP). *lPtr* specifies which system pointer or icon one wishes to access or load. It is one of the constants found in Table 12.4.

Table 12.4 System Pointers

| Constant | Description |
|---|---|
| SPTR_APPICON | Default icon for a PM application |
| SPTR_ARROW | Arrow pointer |
| SPTR_FILE | File icon |
| SPTR_FOLDER | Folder icon |
| SPTR_ICONERROR | Error icon |
| SPTR_ICONINFORMATION | Information icon |
| SPTR_ICONQUESTION | Query icon |
| SPTR_ICONWARNING | Warning icon |
| SPTR_ILLEGAL | Illegal action icon |
| SPTR_MOVE | Move icon |
| SPTR_MULTFILE | Multiple object icon |
| SPTR_PROGRAM | Executable object icon |
| SPTR_SIZE | Sizing pointer |
| SPTR_SIZENESW | Sizing pointer from upper right to lower left |
| SPTR_SIZENWSE | Sizing pointer from upper left to lower right |
| SPTR_SIZENS | Vertical sizing pointer |
| SPTR_SIZEWE | Horizontal sizing pointer |
| SPTR_TEXT | Text "I-beam" pointer |
| SPTR_WAIT | Waiting pointer |

*bLoad* specifies whether the handle to the pointer that the system loaded during its initialization should be returned or whether the pointer should be loaded again and a new handle returned. To make modifications to the pointer for use within your application, *bLoad* should be specified TRUE. This function returns a handle to the specified pointer or to a copy of the specified pointer, depending on the value of *bLoad*.

Pointers and icons that were loaded explicitly by an application are destroyed using the *WinDestroyPointer* function.

```
BOOL WinDestroyPointer(HPOINTER hpPointer);
```

*hpPointer* specifies the handle of the pointer or icon to be destroyed. This function returns a flag indicating success or failure.

## Bitmaps

Bitmaps are similar to their cousins, pointers and icons. However, pointers and icons are of a fixed size, defined by Presentation Manager and cannot be any bigger or smaller. Bitmaps do not have this restriction; they do not have a "transparency" color, though, which is something that pointers and icons do have. Bitmaps in general have many uses—no blanket statement describes their usual purpose in an application.

The manner in which a bitmap is specified within a resource file is like that of the pointer and icon.

```
BITMAP resid filename
```

This causes the bitmap file with the specified name, *filename*, to be included in the resource tables and be assigned the specified resource id, *resid.*

Bitmaps are loaded with the *GpiLoadBitmap* function.

```
HBITMAP GpiLoadBitmap(HPS hpsWnd,
                      HMODULE hmDll,
                      ULONG ulId,
                      LONG lWidth,
                      LONG lHeight);
```

*hpsWnd* is a handle to the presentation space that is used to load the bitmap; this parameter is complex and will not be discussed. *hmDll* is a handle to a DLL that contains the resources, if this is the case. Again, if the resource is appended to the executable, NULLHANDLE should be specified. *ulId* is the resource identifier of the bitmap to be loaded. *lWidth* and *lHeight* are the width and height to which the bitmap should be stretched, if this is desired. Specifying 0 for both of these parameters specifies that the bitmap should be kept at its original size. This function returns a handle to the bitmap loaded.

Drawing a bitmap is accomplished in one of many ways. We will look at the simplest of these, which is to use the *WinDrawBitmap* function. Like *WinDrawPointer*, this will draw a bitmap into a presentation space that is associated with a window.

```
BOOL WinDrawBitmap(HPS hpsWnd,
                   HBITMAP hbmBitmap,
                   PRECTL prclSrc,
                   PPOINTL pptlDest,
                   LONG lForeClr,
                   LONG lBackClr,
                   ULONG ulFlags);
```

*hpsWnd* is, again, a handle to a presentation space in which the bitmap will be drawn. *hbmBitmap* is a handle to the bitmap to be drawn. *prclSrc* points to a RECTL structure that defines the portion of the bitmap to be drawn. If NULLHANDLE is specified, the entire bitmap is drawn. *pptlDest* specifies the point corresponding to where the lower left corner of the bitmap is to be in the presentation space. *lForeClr* and *lBackClr* are the foreground and background colors and are used for monochrome bitmaps only. *ulFlags* specifies how the bitmap is to be drawn and can be one of the constants depicted in Table 12.5.

Table 12.5 Values for *ulFlags*

| Constant | Description |
|---|---|
| DBM_NORMAL | Draw the bitmap in a "normal" fashion. |
| DBM_INVERT | Draw the bitmap in a color-inverted state. |
| DBM_HALFTONE | Draw the bitmap in a halftone manner. |
| DBM_STRETCH | Draw the bitmap stretched to fit *prclSrc.* |
| DBM_IMAGEATTRS | Draw the (monochrome) bitmap using the current foreground and background colors of the presentation space. *lForeClr* and *lBackClr* are ignored if this is specified. |

This function returns a flag indicating its success or failure.

We've used the word "monochrome" twice, so it is helpful to be able to determine what the parameters are that were used to create the bitmap. This is done with the *GpiQueryBitmapInfoHeader* function.

```
BOOL GpiQueryBitmapInfoHeader(HBITMAP hbmBitmap,
                              PBITMAPINFOHEADER2 pbihInfo);
```

*hbmBitmap* is a handle to the bitmap in which the programmer is interested. *pbihInfo* points to a very interesting structure—BITMAPINFOHEADER2.

```
typedef struct _BITMAPINFOHEADER2 {
   ULONG  cbFix;
   ULONG  cx;
   ULONG  cy;
   USHORT cPlanes;
   USHORT cBitCount;
   ULONG  ulCompression;
   ULONG  cbImage;
   ULONG  cxResolution;
   ULONG  cyResolution;
   ULONG  cclrUsed;
   ULONG  cclrImportant;
   USHORT usUnits;
   USHORT usReserved;
   USHORT usRecording;
   USHORT usRendering;
   ULONG  cSize1;
   ULONG  cSize2;
   ULONG  ulColorEncoding;
   ULONG  ulIdentifier;
} BITMAPINFOHEADER2, *PBITMAPINFOHEADER2;
```

The *GpiQueryBitmapInfoHeader* function returns a flag indicating success or failure of the function.

In OS/2 versions 1.x, this structure was called BITMAPINFOHEADER and contained only the first five fields. In the current structure, PM developers have enabled programmers to have much more control over the creation of a bitmap (or, in this situation, much more information about a bitmap). However, they also realized that programmers probably still will use only the first five fields. So, the Gpi requires only that programmers initialize all fields up to the last one they are interested in and that they specify the number of bytes initialized in the *cbFix* field; and if the parameters of an existing bitmap are being queried, only *cbFix* needs to be initialized to specify how many bytes need to be returned. Thus, if *cbFix* has the value 16, only the first five fields (sizeof(*cbFix*) + sizeof(*cx*) + sizeof(*cy*) + sizeof(*cPlanes*) + sizeof(*cBitCount*) = 16) would be provided, but any value that makes sense, up to the size of the structure, can be specified. Before *GpiQueryBitmapInfoHeader* is called, *cbFix* should be initialized to specify how much information should be returned.

**Gotcha!**

Initializing *cbFix* to the proper value is a must when calling the *GpiQueryBitmapInfoHeader* function, or unpredictable information will be returned.

The fields of the structure are explained in Table 12.6.

Table 12.6 BITMAPINFOHEADER2 Structure Details

| Field | Description |
|---|---|
| *cbFix* | Length of structure |
| *cx* | Bitmap width in pels |

| Field | Description |
|---|---|
| *cy* | Bitmap height in pels |
| *cPlanes* | Number of bit planes |
| *cBitCount* | Number of bits per pel within a plane |
| *ulCompression* | Compression scheme used to store the bitmap |
| *cbImage* | Length of bitmap storage data in bytes |
| *cxResolution* | Horizontal resolution of the intended target device |
| *cyResolution* | Vertical resolution of the intended target device |
| *cclrUsed* | Number of color indices used |
| *cclrImportant* | Number of important color indices used |
| *usUnits* | Units of measure |
| *usReserved* | Reserved |
| *usRecording* | Recording algorithm used |
| *usRendering* | Halftoning algorithm used |
| *cSize1* | Size value 1 |
| *cSize2* | Size value 2 |
| *ulColorEncoding* | Color encoding used |
| *ulIdentifier* | Reserved for application use |

*cx* and *cy* specify the width and height of the bitmap. *cPlanes* specifies the number of color planes used by the bitmap; while OS/2 supports multiplane bitmaps, the APIs to draw bitmaps support only single-plane bitmaps. *cBitCount* specifies the number of bits it takes to represent one pel in the bitmap and can have the value 1, 2, 4, 8, or 24; if the value is 1, it is a monochrome bitmap, since it can have only $2^1$ colors. *ulCompression* specifies the compression scheme used to compress the bitmap in memory and can be one of the values listed in Table 12.7.

Table 12.7 Values for *ulCompression*

| Constant | Description |
|---|---|
| BCA_UNCOMP | Uncompressed |
| BCA_HUFFMAN1D | Huffman encoding scheme |
| BCA_RLE4 | Run-length encoding for 4 bit-per-pel (BPP) bitmaps |
| BCA_RLE8 | Run-length encoding for 8 BPP bitmaps |
| BCA_RLE24 | Run-length encoding for 24 BPP bitmaps |

*cbImage* specifies how much memory is needed to store the bitmap data. *cxResolution* and *cyResolution* specify the resolution of the device for which the bitmap was intended to be displayed upon. This does not prohibit the bitmap from being displayed on another display type; it merely indicates the display type for which the bitmap was drawn. *cclrUsed*, *cclrImportant*, *ulRecording*, *ulRendering*, *cSize1*, *cSize2*, and *ulColorEncoding* all specify additional data as described in Table 12.6 and are beyond the scope of this text.

Bitmaps are destroyed using the *GpiDeleteBitmap* function.

```
BOOL GpiDeleteBitmap(HBITMAP hbmBitmap);
```

*hbmBitmap* specifies the handle to the bitmap to be deleted. This function returns a flag indicating the success or failure of the function.

# String Tables

String tables are very simple in concept and implementation: They are lookup tables where the application provides the resource identifier of a string and Presentation Manager provides the corresponding text that was defined in the resource file. The purpose of a string table is to allow easy translation of an application to other languages, providing all of the "user-readable" text is placed into a string table. "User-readable" in this sense means text that the user sees; window class names would not be included in this group, but messages would be.

Unlike all other resources, string tables do not have a resource identifier explicitly assigned to them by the programmer. Instead, the resource compiler breaks up the string table into groups of 16 strings and automatically assigns an identifier to each 16-string group. A string table has the following form in a resource file.

```
STRINGTABLE
{
   resid1, "string1"
   resid2, "string2"
   resid3, "string3"
}
```

As was stated earlier and is now obvious, a string table is simply that—a table of strings. Each string has a unique identifier associated with it, which is specified on the call to *WinLoadString*, which loads a string from the string table.

```
LONG WinLoadString(HAB habAnchor,
                   HMODULE hmDll,
                   ULONG ulId,
                   LONG lSzBuffer,
                   PCHAR pchBuffer);
```

*habAnchor* is the handle to the anchor block of the calling thread. *hmDll* is the handle to the DLL where the string table resides, or NULLHANDLE if it resides in the executable's resource tables. *ulId* is the identifier of the string to be loaded. *lSzBuffer* specifies the size of the buffer pointed to by *pchBuffer*. This function returns the number of characters loaded from the string table, up to a maximum of *lSzBuffer* - 1.

That's all there is to it!

# Accelerators

Accelerators are "shortcut" keys that accelerate the rate at which a user is able to complete certain tasks within an application. The accelerator table defines a translation from a keystroke, modified by the Alt, Ctrl, or Shift keys if specified, to a numeric identifier that is sent to the application via the WM_COMMAND message.

The accelerator table has the following form.

```
ACCELTABLE resid
{
   key, cmd_id, type [, modifiers]
   key, cmd_id, type [, modifiers]
   key, cmd_id, type [, modifiers]
}
```

*resid* is the resource identifier for the accelerator table. *key* is the base key for the accelerator and can be a VK_ constant (e.g. VK_F1) or a character in quotes. *cmd_id* is the numeric identifier to be sent as SHORT1FROMMP(*mpParm1*) in the WM_COMMAND message. *type* is the type of character and must be CHAR or VIRTUALKEY. *modifiers* are optional and can be one or more of those listed in Table 12.8, separated by commas.

Table 12.8 Values for *modifiers*

| Modifier | Description |
|---|---|
| CONTROL | Ctrl key must be pressed. |
| ALT | Alt key must be pressed. |
| SHIFT | Shift key must be pressed. |

**Gotcha!**

If a character (instead of a virtual key) is specified for an accelerator, it is case-sensitive, so two entries must be provided to cover both possibilities of the shift key state (unless each case should have different meanings, of course).

If the sole modifier of a character accelerator is the control key, the CONTROL modifier may be omitted and the key prefixed with a caret symbol, "^". Also, keys that are not virtual keys must be specified in quotes.

```
ACCELTABLE RES_CLIENT
{
   "^O", MI_OPEN
   "^o", MI_OPEN
}
```

Accelerator tables usually are associated with standard windows through the use of the FCF_ACCELTABLE frame control flag. However, an accelerator table can be loaded explicitly with the *WinLoadAccelTable* function.

```
HACCEL WinLoadAccelTable(HAB habAnchor,
                         HMODULE hmDll,
                         ULONG ulId);
```

*habAnchor* is the handle to the anchor block of the calling thread. *hmDll* is the handle to the DLL if the accelerator table resides there, or to NULLHANDLE if it is in the executable's resource tables. *ulId* is the resource identifier of the accelerator table. This function returns a handle to the loaded accelerator table.

After an accelerator table is loaded, it can be made active with the *WinSetAccelTable* function.

```
BOOL WinSetAccelTable(HAB habAnchor,
                      HACCEL haAccel,
                      HWND hwndFrame);
```

*habAnchor* is the handle to the anchor block of the calling thread. *haAccel* is the handle to the accelerator table to be made active. *hwndFrame* is the handle to the frame window to which the accelerator table is attached. This function returns a flag indicating success or failure.

For each message queue, there are certain "standard" accelerators that are defined, such as Alt+F4 to close a frame window. These are called "queue accelerators," since they are in effect for the entire message queue and are independent of the active window. If *hwndFrame* in the call to *WinSetAccelTable* is NULLHANDLE, the accelerator table replaces the queue accelerator table.

Accelerator tables are destroyed with the *WinDestroyAccelTable* function.

```
BOOL WinDestroyAccelTable(HACCEL haAccel);
```

This function destroys the accelerator table whose handle is specified in *haAccel* and returns a flag indicating success or failure.

# Dialog Boxes

Dialog boxes are complicated beasts, but their use is simplified greatly through the use of the "dialog box editor" (DLGEDIT.EXE). A dialog box is described in a resource file using the *dialog template*. This template consists of three parts:

- The DLGTEMPLATE statement
- The DIALOG statement
- One or more child window definitions

The nice thing is that the dialog box editor will create the template for the programmer; all he or she needs to do is build the dialog box using its WYSIWYG interface. When the work is saved in the dialog box editor, a dialog file (.DLG) is generated, containing the dialog templates corresponding to the dialog boxes that the programmer designed.

However, it is nice to know how to make minor adjustments manually, so let us look briefly at the format of the dialog template.

```
DLGTEMPLATE resid
{
   DIALOG "title text", resid, x, y, cx, cy, style, flags
   [CTLDATA controldata]
   [PRESPARAM presparam]
{
      CONTROL "text", id, x, y, cx, cy, class, style
      [CTLDATA controldata]
      [PRESPARAM presparam]
   }
}
```

**Gotcha!**

The *resid* on the DLGTEMPLATE and DIALOG statements must match, or the dialog will fail to load. Why the same constant must be specified twice is beyond our understanding.

*x*, *y*, *cx*, and *cy* are the coordinates of the lower left corner and the size of the dialog or window, respectively. *style* is one or more style flags; since a dialog is really nothing more than a subclassed frame window, it can use the FS_ constants in addition to the WS_ constants. The child windows (CONTROL statement) can use the WS_ constants as well as the constants specific to their window class. *class* can be a

WC_ constant or an application-defined class—registered prior to the loading of the dialog with *WinRegisterClass*—in double quotes.

The *control data* (CTLDATA statement) is used to initialize the dialog or the child window, as will be shown in later chapters. The *presentation parameters* (PRESPARAM statement) define the appearance, such as the font used, the foreground and background colors, and so on. See Chapter 9 for more information on setting presentation parameters.

It should be noted that the coordinates and size of the dialog and the child windows are based on a different coordinate system; the units are *dialog units*, which are based on the average character width of the system font for the resolution of the display. The concept-went-awry is that dialog units are supposed to be "display independent," meaning that the dialog will occupy the same amount of physical space on different resolutions; however, most monitors do not report their pel densities properly, so this rarely works. *WinMapDlgPoints* can be used to convert between dialog units and pels.

```
BOOL WinMapDlgPoints(HWND hwndDlg,
                     PPOINTL pptlPoints,
                     ULONG ulNumPoints,
                     BOOL bCalcWindow);
```

*hwndDlg* is the handle to the dialog window. *pptlPoints* points to one or more POINTL structures to convert. *ulNumPoints* specifies how many structures *pptlPoints* points to. *bCalcWindow* is TRUE if the programmer wants to convert to window coordinates from dialog coordinates or FALSE if the opposite is desired.

# Menus

Menus are a familiar user-interface component to anyone who has used a MacIntosh, Windows, OS/2, or some other GUI. Their definition in a resource file is also quite simple, for there are only three different parts: the main "MENU" keyword, submenu definitions, and menu item definitions.

```
MENU resid
{
   SUBMENU "Text", submenu_id [, styles]
   {
      MENUITEM "Text", menuitem_id [, attributes]
      MENUITEM "Text", menuitem_id [, attributes]
   }
}
```

*resid* is the resource identifier of the menu. *submenu_id* and *menuitem_id* are unique identifiers of the submenus and menu items, respectively. They are used when communicating with the menu via the MM_ messages. *styles* are one or more MIS_ constants that affect the entire submenu. *attributes* are one or more MIA_ constants that affect a specific menu item. Both *styles* and *attributes* are optional.

See Chapter 14 for more information on using menus.

# Help Tables

Help tables are used to provide a linkage between the application's child windows (including menu items, which are child windows in an odd way) and the help panels which are defined by a help developer. As you will see in Chapter 29, there are two parts to this linkage: the HELPTABLE and the various HELPSUBTABLES. See that chapter for information on the resource file syntax and how online help is provided by an application.

# Application-defined Data

Application-defined data is the general case for all resources. In fact, all of the APIs discussed in this chapter for loading resources follow these instructions in the bowels of the Presentation Manager code. The OS/2 kernel provides two APIs for resource management that are used to load and unload a specific resource—*DosGetResouce* and *DosFreeResource*.

```
APIRET DosGetResource(HMODULE hmDll,
                      ULONG ulType,
                      ULONG ulId,
                      PPVOID ppvData);

APIRET DosFreeResource(PVOID pvData);
```

*hmDll* is the handle to the DLL where the resource resides, or is NULLHANDLE if it is found in the executable's resource tables. *ulType* is an RT_ constant that specifies the type of the resource.

Table 12.9 Resource Type Constants

| Constant | Description |
|---|---|
| RT_POINTER | Pointer data |
| RT_BITMAP | Bitmap data |
| RT_MENU | Menu template |
| RT_DIALOG | Dialog template |
| RT_STRING | String table |
| RT_FONTDIR | Font directory |
| RT_FONT | Font data |
| RT_ACCELTABLE | Accelerator table |
| RT_RCDATA | Binary data |
| RT_MESSAGE | Error message |
| RT_DLGINCLUDE | File name for the DLGINCLUDE statement |
| RT_HELPTABLE | Help table for Help Manager |
| RT_HELPSUBTABLE | Help subtable for Help Manager |

*ulId* is the resource identifier to be loaded. *ppvData* is a pointer to a pointer that is initialized by OS/2 to point to the beginning of the resource data. This pointer is specified on the call to *DosFreeResource* to return the memory consumed to the system, since OS/2 allocates the memory for the programmer when *DosGetResource* is called.

In the resource file, application-defined data must reside in a separate file and is included via the RESOUCE keyword.

```
RESOURCE type resid filename
```

*type* and *resid* correspond to their definitions as described earlier, and *filename* is the name of the file where the resource data resides. It should be noted that application-defined resources must have a value for *type* of 256 or greater.

# Chapter 13

# Dialog Boxes

Dialog boxes are designed to gather specific pieces of information from the user. Dialogs contain a mix and match of child control windows. A window that pops up and contains such fields as "Name:," "Address," "Phone," "City," and "State," is a good example of a dialog box.

There are three ways to create a dialog box and its child controls—by using a resource file, by physically calling the *WinCreateWindow* for the dialog box and each of its controls, or by using *WinCreateDlg*. The resource file is the easiest way to create a dialog box. The Dialog Box Editor shipped with the Toolkit is designed to help facilitate this creation process.

Dialog boxes come in two styles—modal and modeless. A modeless dialog box lets the user interact with all the other windows and controls belonging to the same process. A modal dialog box is more restrictive of the user's input. A user cannot interact with the other windows and controls that are children of the owner of the dialog box, including the owner. A modal dialog box is designed to be used when the user is required to enter some information before proceeding on to the next step in the application.

The following sample program is designed to introduce dialog box programming and to display the difference between modal and modeless dialog boxes.

**DIALOG.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "dialog.h"
#define  CLS_CLIENT    "MyClass"
MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

MRESULT EXPENTRY DlgProc(HWND hwndWnd,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);

typedef struct   _DLGINFO
{
  ULONG            ulStructSize;

  BOOL             bModal;
  HWND             hwndModeless;
  HWND             hwndClient;

} DLGINFO,*PDLGINFO;

VOID DisplayError(CHAR *pString);
```

```
INT main(VOID)
{
  HAB              habAnchor;
  HMQ              hmqQueue;
  ULONG            ulFlags;
  HWND             hwndFrame;
  HWND             hwndClient;
  QMSG             qmMsg;
  BOOL             bLoop;
  LONG             lHeight;
  LONG             lWidth;

  habAnchor = WinInitialize(0);
  hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

  /*************************************************************/
  /* register class with space for window word                 */
  /*************************************************************/

  WinRegisterClass(habAnchor,
                   CLS_CLIENT,
                   ClientWndProc,
                   CS_SIZEREDRAW,
                   sizeof(PVOID));

  ulFlags = FCF_STANDARD&~FCF_SHELLPOSITION&~FCF_ACCELTABLE&~
     FCF_TASKLIST;

  /*************************************************************/
  /* create the frame, client, etc.                            */
  /*************************************************************/

  hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                 0,
                                 &ulFlags,
                                 CLS_CLIENT,
                                 "Dialog Box Example",
                                 0,
                                 NULLHANDLE,
                                 IDR_CLIENT,
                                 &hwndClient);

  /*************************************************************/
  /* find desktop dimensions                                   */
  /*************************************************************/

  lHeight = WinQuerySysValue(HWND_DESKTOP,
                             SV_CYSCREEN);
  lWidth = WinQuerySysValue(HWND_DESKTOP,
                            SV_CXSCREEN);

  /*************************************************************/
  /* center frame window on desktop                            */
  /*************************************************************/

  if (hwndFrame != 0)
  {
    WinSetWindowPos(hwndFrame,
                    NULLHANDLE,
                    lWidth/8,
                    lHeight/8,
                    lWidth/8*6,
                    lHeight/8*6,
                    SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);
```

```
    /*********************************************************/
    /* message loop processing                               */
    /*********************************************************/

    bLoop = WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
    while (bLoop)
    {
      WinDispatchMsg(habAnchor,
                     &qmMsg);
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
    }                                   /* endwhile              */
    WinDestroyWindow(hwndFrame);
  }                                     /* endif                 */
  WinDestroyMsgQueue(hmqQueue);
  WinTerminate(habAnchor);
  return 0;
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)
{
  switch (ulMsg)
  {

    case  WM_INITDLG :
      {
        BOOL             bModal;
        CHAR             achMessage[64];
        PDLGINFO         pDlgInfo;
        RECTL            rclClient;
        POINTL           ptPoints;
        LONG             lHeight,lWidth;

        /*****************************************************/
        /* dialog create information is stored in mpParm2    */
        /*****************************************************/

        pDlgInfo = PVOIDFROMMP(mpParm2);

        if (!pDlgInfo)
        {
          DisplayError("No create information");
          return  MRFROMSHORT(TRUE);
        }

        /*****************************************************/
        /* the bModal flags says whether we're modal or      */
        /* modeless                                          */
        /*****************************************************/

        bModal = pDlgInfo->bModal;

        /*****************************************************/
        /* do the sizing calculations, so dialogs look right */
        /*****************************************************/
```

```
      WinQueryWindowRect(pDlgInfo->hwndClient,
                         &rclClient);

      lHeight = rclClient.yTop-rclClient.yBottom;
      lWidth = rclClient.xRight-rclClient.xLeft;

      /*******************************************************/
      /* since parent of dialogs are desktop, they will be   */
      /* positioned relative to the desktop, but we want them*/
      /* centered on the client start at 1/8th of the client */
      /*******************************************************/

      ptPoints.x = lWidth/8;

      /*******************************************************/
      /* start at 1/19th of the height of the client and     */
      /* 10/19th for the modeless dialog                     */
      /*******************************************************/

      ptPoints.y = bModal?lHeight/19:lHeight/19*10;

      /*******************************************************/
      /* find out where these points are relative to the     */
      /* desktop                                             */
      /*******************************************************/

      WinMapWindowPoints(pDlgInfo->hwndClient,
                         HWND_DESKTOP,
                         &ptPoints,
                         1);

      WinSetWindowPos(hwndDlg,
                      NULLHANDLE,
                      ptPoints.x,
                      ptPoints.y,
                      lWidth/8*6,
                      lHeight/19*8,
                      SWP_MOVE|SWP_SIZE);

      sprintf(achMessage, "I'm a %s dialog box", (bModal?
         ("modal"):("modeless")));

      WinSetDlgItemText(hwndDlg,
                        IDT_DIALOGNAME,
                        achMessage);

      if (bModal)
      {
        strcpy(achMessage, "Try to click on the main window");
      }
      else
      {
        strcpy(achMessage, "Click on the main window");
      }                                 /* endif               */
      WinSetDlgItemText(hwndDlg,
                        IDT_CLICK,
                        achMessage);
    }
    break;
  case  WM_COMMAND :

    switch (SHORT1FROMMP(mpParm1))
    {
      case  DID_OK :
      case  DID_CANCEL :
```

```
        /*****************************************************/
        /* for the modeless dialog, this function hides the  */
        /* dialog; for the modal dialog, this function       */
        /* destroys the dialog                               */
        /*****************************************************/

        WinDismissDlg(hwndDlg,
                      FALSE);
        break;

      default  :
        return  WinDefDlgProc(hwndDlg,
                              ulMsg,
                              mpParm1,

                              mpParm2);
    }                                 /* endswitch           */
    break;

  default  :
    return  WinDefDlgProc(hwndDlg,
                          ulMsg,
                          mpParm1,
                          mpParm2);
}                                     /* endswitch           */
return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
  switch (ulMsg)
  {

    case  WM_ERASEBACKGROUND :
      return  MRFROMSHORT(TRUE);

    case  WM_CREATE :
      {
        PDLGINFO         pDlgInfo;

        pDlgInfo = calloc(1, sizeof(DLGINFO));
        if (!pDlgInfo)
        {

          /*****************************************************/
          /* if error, display message and halt creation       */
          /*****************************************************/

          DisplayError("Cannot allocate memory");
          return  MRFROMSHORT(TRUE);

        }

        pDlgInfo->ulStructSize = sizeof(DLGINFO);
        pDlgInfo->hwndClient = hwndWnd;

        /*****************************************************/
        /* associate pDlgInfo with window word               */
        /*****************************************************/

        WinSetWindowPtr(hwndWnd,
                        QWL_USER,
                        pDlgInfo);
        break;
      }                               /* end WM_CREATE       */
    case  WM_DESTROY :
```

```
    {
      PDLGINFO         pDlgInfo;

      pDlgInfo = WinQueryWindowPtr(hwndWnd,
                                   QWL_USER);

      /*****************************************************/
      /* clean up                                          */
      /*****************************************************/

      if (pDlgInfo)
      {

        /*****************************************************/
        /* if the modelesss dialog box is still around,      */
        /* destroy it                                        */
        /*****************************************************/

        if (pDlgInfo->hwndModeless)
          WinDestroyWindow(pDlgInfo->hwndModeless);
        free(pDlgInfo);
      }
      break;
    }

  case  WM_COMMAND :

    switch (SHORT1FROMMP(mpParm1))
    {
      case  IDM_MODELESS :
        {
          PDLGINFO         pDlgInfo;

          pDlgInfo = WinQueryWindowPtr(hwndWnd,
                                       QWL_USER);
          if (!pDlgInfo)
          {
            DisplayError("Missing Window Word");
            break;
          }

          pDlgInfo->bModal = FALSE;

          /*****************************************************/
          /* if the dialog has not been created, call the      */
          /* loader, else, just show the dialog window         */
          /*****************************************************/

          if (!pDlgInfo->hwndModeless)
            pDlgInfo->hwndModeless = WinLoadDlg(HWND_DESKTOP,
                                                hwndWnd,
                                                DlgProc,
                                                NULLHANDLE,
                                                IDD_DIALOG,
                                                pDlgInfo);
          else
            WinSetWindowPos(pDlgInfo->hwndModeless,
                            HWND_TOP,
                            0,
                            0,
                            0,
                            0,
                            SWP_SHOW|SWP_ACTIVATE);
        }
        break;

      case  IDM_MODAL :
        {
          PDLGINFO         pDlgInfo;
```

```
            pDlgInfo = WinQueryWindowPtr(hwndWnd,
                                         QWL_USER);
            if (!pDlgInfo)
            {
              DisplayError("Missing Window Word");
              break;
            }

            pDlgInfo->bModal = TRUE;

            WinDlgBox(HWND_DESKTOP,
                      hwndWnd,
                      DlgProc,
                      NULLHANDLE,
                      IDD_DIALOG,
                      pDlgInfo);
          }
          break;

        case  IDM_EXIT :
          WinPostMsg(hwndWnd,
                     WM_CLOSE,
                     0,
                     0);
          break;

        default  :
          return  WinDefWindowProc(hwndWnd,
                                   ulMsg,
                                   mpParm1,

                                   mpParm2);
      }                                 /* endswitch              */
      break;
    default  :
      return  WinDefWindowProc(hwndWnd,
                               ulMsg,
                               mpParm1,

                               mpParm2);
  }                                     /* endswitch              */
  return  MRFROMSHORT(FALSE);
}

/* small function to beep and display error message              */

VOID DisplayError(CHAR *pString)
{

  WinAlarm(HWND_DESKTOP,
           WA_ERROR);
  WinMessageBox(HWND_DESKTOP,
                HWND_DESKTOP,
                pString,
                "Error",
                0,
                MB_ICONEXCLAMATION|MB_OK);

  return ;
}
```

## DIALOG.RC

```
#include <os2.h>
#include "dialog.h"

ICON IDR_CLIENT DIALOG.ICO

MENU IDR_CLIENT
{
   SUBMENU "~Dialog", IDM_DIALOG
   {
      MENUITEM "~Modeless dialog...", IDM_MODELESS
      MENUITEM "Modal ~dialog...", IDM_MODAL
      MENUITEM SEPARATOR
      MENUITEM "E~xit", IDM_EXIT
   }
}

DLGTEMPLATE IDD_DIALOG LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG  "Dialog example", IDD_DIALOG, 53, 28, 260, 55,
      WS_VISIBLE,
      FCF_SYSMENU | FCF_TITLEBAR
   {
      LTEXT "?", IDT_DIALOGNAME, 10, 40, 150, 8
      LTEXT "?", IDT_CLICK, 10, 30, 150, 8
      DEFPUSHBUTTON "OK", DID_OK, 10, 10, 50, 13
   }
}
```

## DIALOG.H

```
#define IDR_CLIENT                  256
#define IDD_DIALOG                  257

#define IDM_DIALOG                  320
#define IDM_MODELESS                321
#define IDM_MODAL                   322
#define IDM_EXIT                    323

#define IDT_DIALOGNAME              512
#define IDT_CLICK                   513
```

## DIALOG.MAK

```
DIALOG.EXE:                     DIALOG.OBJ \
                                DIALOG.RES
        LINK386 @<<
DIALOG
DIALOG
DIALOG
OS2386
DIALOG
<<
        RC DIALOG.RES DIALOG.EXE

DIALOG.RES:                     DIALOG.RC \
                                DIALOG.H
        RC -r DIALOG.RC DIALOG.RES

DIALOG.OBJ:                     DIALOG.C \
                                DIALOG.H
        ICC -C+ -Kb+ -Ss+ DIALOG.C
```

**DIALOG.DEF**

```
NAME DIALOG WINDOWAPI
DESCRIPTION 'Dialog example.
            Copyright (c) 1992-1995 by Kathleen Panov.
            All rights reserved.'


STACKSIZE 16384
```

The resource file, DIALOG.RC, is the starting point for the sample program. Two items are defined in the file, a menu and the dialog box. The resource file for the window shows the menu that we would like displayed in our client window. For more information on resources, see Chapter 12.

# The Dialog Box Template

The following is the resource definition to create the dialog boxes used in the DIALOG.C program.

```
DLGTEMPLATE IDD_DIALOG LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG  "Dialog example", IDD_DIALOG, 53, 28, 260, 55,
      WS_VISIBLE,
      FCF_SYSMENU | FCF_TITLEBAR
   {
      LTEXT "?", IDT_DIALOGNAME, 10, 40, 150, 8
      LTEXT "?", IDT_CLICK, 10, 30, 150, 8
      DEFPUSHBUTTON "OK", DID_OK, 10, 10, 50, 13
   }
}
```

The dialog IDD_DIALOG is created in the resource file as visible, with a system menu and title bar.

The next step is to define the controls that are to appear on the dialog box. In this example only an "OK" pushbutton and some static text will be used. The IDT_CLICK text will be used to communicate some instructions to the user. The IDT_DIALOGNAME is used to specify whether this is a modal or modeless dialog box.

# The Client Window Procedure

The client window procedure, *ClientWndProc*, is not very big. A window word is used to store some information that we will need later in the dialog procedure. This information is stored in a DLGINFO structure. The structure includes the structure size, a BOOL variable to indicate whether the user selected modal or modeless from the menu, the handle of the modeless dialog box, and the handle of the client window. This structure is allocated in the WM_CREATE processing, and cleanup is done in the WM_DESTROY processing.

The programmatic differences between a modal and nonmodal dialog box exist in the processing of the WM_COMMAND message.

In our WM_COMMAND processing, we first find out who is sending us the WM_COMMAND message. The resource ID for the sender is located in *mpParm1*. If the user selected "Modal Dialog Box," IDM_MODAL is returned in *mpParm1*. A Boolean variable, *pDlgInfo->bModal*, is used to indicate to the *DlgProc* whether the user selected a modal or modeless dialog box.

# Creating a Modal Dialog Box

The function *WinDlgBox* is used to create a modal dialog box.

```
ULONG  WinDlgBox(HWND hwndParent,
                 HWND hwndOwner,
                 PFNWP pfnDlgProc,
                 HMODULE hmod,
                 ULONG idDlg,
                 PVOID pCreateParams);
```

When *WinDlgBox* is used to create a dialog box, a message queue is created for that dialog. User interaction with the other message queue (and the client window associated with it) is held up until the dialog box is dismissed and the message queue is destroyed.

```
pDlgInfo->bModal = TRUE;

WinDlgBox(HWND_DESKTOP,
          hwndWnd,
          DlgProc,
          NULLHANDLE,
          IDD_DIALOG,
          pDlgInfo);
```

The first parameter is the parent, HWND_DESKTOP, and the second parameter is the owner window, *hwndWnd*. The programmer almost always will want to specify the desktop as the parent of a modal dialog, and the client window as the owner. If the frame or client was specified as the parent of the dialog, the frame window would still be active, thus preventing the whole purpose of using a modal dialog. The third parameter is the pointer to the dialog process function, in this case *DlgProc*. NULLHANDLE tells the system that the resources for the dialog process, *DlgProc*, are located in the .EXE file. IDD_DIALOG is the resource ID for the dialog. The last parameter is the data area. This is used to pass programmer-defined data of type PVOID into the dialog procedure. In this area we will pass a pointer to our dialog information structure, *pDlgInfo*. *WinDlgBox* is actually a combination of four functions, *WinLoadDlg*, *WinProcessDlg*, *WinDestroyWindow*, and *return*.

**Gotcha!**

The last parameter to *WinDlgBox must* be a pointer. This parameter undergoes a procedure called "thunking" that converts a 32-bit pointer into a pointer that is readable by 16-bit code. The application will trap if the value is not a pointer and the system attempts to thunk it. The dialog box functions are 16-bit in OS/2 2.1, and must try and thunk this value. The dialog box functions in Warp are 32-bit, so no thunking will be done; however, if previous versions of the operating system must be supported, it is best to be prepared for thunking.

## Creating a Modeless Dialog Box

```
pDlgInfo->bModal = FALSE;

if (!pDlgInfo->hwndModeless)
  pDlgInfo->hwndModeless = WinLoadDlg(HWND_DESKTOP,
                                      hwndWnd,
                                      DlgProc,
                                      NULLHANDLE,
                                      IDD_DIALOG,
                                      pDlgInfo);
else
  WinSetWindowPos(pDlgInfo->hwndModeless,
                  HWND_TOP,
                  0,
                  0,
                  0,
                  0,
                  SWP_SHOW|SWP_ACTIVATE);
```

In this example, we first set the *bModal* variable to FALSE to indicate that this will be a modeless dialog box.

**Gotcha!**

A modeless dialog is not destroyed by *WinDismissDlg*, only hidden. In order to destroy the dialogs loaded by *WinLoadDlg*, *WinDestroyWindow* must be called implicitly for each modeless dialog that has been created.

If the user selects the modeless option from the menu multiple times, we do not create the same dialog over and over; instead, we just check to see if its already exists. If the window handle is there, *WinSetWindowPos* is used to show the dialog and make it the active window.

*WinLoadDlg* is used to create a modeless dialog box, and this function returns immediately after creating it. *WinDlgBox* waits until it finishes its processing before returning. This is why a modeless dialog box permits user interaction with the other windows and a modal dialog box does not. The parameter list for *WinLoadDlg* is exactly the same as for *WinDlgBox*.

## The Dialog Procedure *DlgProc*

The dialog procedure, in this case *DlgProc*, is fairly similar to a window procedure. Our program can use the same dialog process for both the modal and modeless dialog boxes.

**Gotcha!**

One difference between a dialog procedure and a window procedure is the default procedure function. A dialog procedure must call *WinDefDlgProc* instead of *WinDefWindowProc*. If a dialog procedure behaves irrationally, it should be checked to see if it includes *WinDefDlgProc*. These two functions often get interchanged.

One of the other differences between dialog and window procedures is the appearance in the former of the WM_INITDLG message *instead of* the usual WM_CREATE. This message is provided to give the programmer a place to put the initialization code for the dialog box.

```
pDlgInfo = PVOIDFROMMP(mpParm2);
```

The first thing we do is retrieve the information sent to us through the *WinLoadDlg* or *WinDlgBox* function. Both these functions will send this information in the message parameter 2 of the WM_INITDLG message.

```
WinQueryWindowRect(pDlgInfo->hwndClient,
                   &rclClient);

lHeight = rclClient.yTop-rclClient.yBottom;
lWidth = rclClient.xRight-rclClient.xLeft;
```

In order to make our dialog program prettier, we'll position the two dialogs directly on the client window. However, the parent of the dialogs is the desktop, and remember, the children will be positioned relative to the parent. So we do some math. First, we find the height and width of the client area, and use these dimensions to see where the dialogs should be placed relative to the client. We'll start the dialogs at the x coordinate that is 1/8th of the client area width. The y coordinate will differ depending on whether the dialog is the modal dialog or the modeless dialog.

```
ptPoints.x = lWidth/8;
ptPoints.y = bModal?lHeight/19:lHeight/19*10;
```

Now that we know where we would put our dialogs if they were placed relative to the client window's coordinate system, all we have to do is find where these coordinates are on the desktop window. And yes, Presentation Manager has a function that will do this for us: *WinMapWindowPoints.*

```
WinMapWindowPoints(   HWND hwndSource, HWND hwndDestination,
       PPOINTL aptlPoints, LONG lCount ) ;
```

*hwndSource* is the handle of the window to map the coordinate space from. *hwndDestination* is the handle of the window to map the coordinate space to. *aptlPoints* is a point to an array (one or more) of POINTL structures that on input contain the coordinates to map and on output contain the new coordinates relative to *hwndDestination*. *lCount* is the number of structures in the *aptlPoints* array

In our case, the function looks like this.

```
WinMapWindowPoints(pDlgInfo->hwndClient,
                   HWND_DESKTOP,
                   &ptPoints,
                   1);
```

On the function's return, *ptPoints* will contain the new x and y coordinates relative to the desktop. We use these coordinates as the basis for the *WinSetWindowPos* function to adjust the size and position of the dialog.

```
WinSetWindowPos(hwndDlg,
                NULLHANDLE,
                ptPoints.x,
                ptPoints.y,
                lWidth/8*6,
                lHeight/19*8,
                SWP_MOVE|SWP_SIZE);
```

The WM_COMMAND processing is just like the WM_COMMAND processing for the client window. If the user presses the OK pushbutton, the dialog box is canceled with *WinDismissDlg*.

## WM_COMMAND and Dialogs

Some "features" (actually they really can be nice) can cause problems in the future if programmers are unaware of the way *WinDefDlgProc* handles WM_COMMAND messages. A dialog will be dismissed if a WM_COMMAND message is passed to *WinDefDlgProc*. In some cases, this makes sense. For instance, if the user presses the OK or CANCEL pushbuttons, it would be perfectly logical for the dialog box to go away. However, if other pushbuttons exist, and the programmer does not want the dialog box to be dismissed, WM_COMMAND processing must be intercepted and return FALSE, instead of letting the message processing fall through to *WinDefDlgProc*. This also means that *WinDismissDlg* must be called when the programmer is ready for the dialog to disappear and *WinDestroyWindow* when he or she is ready to destroy the dialog box.

*WinDismissDlg* is also called if a WM_QUIT message is sent to the dialog.

## Summary

Dialogs will become an integral part of most of a programmer's Presentation Manager programs. They are easy to use and provide a clean user interface. The main drawback to dialogs is the lack of true device-independent dialog coordinates. Currently, a set of multiple dialogs must be created for different screen resolutions. Perhaps someday.

# Chapter 14

# Menus

The menu is a control that provides a list of choices to the user. There are four types of menus: the menu bar, pull-down menus, cascaded menus, and pop-up menus. A menu uses a small amount of screen real estate and can be very valuable complex applications by providing visual clues to the user.

A menu bar is displayed in the area between the title bar and the client area of a window. A menu bar is almost always visible, and contains either specified choices or a description of the choices that the pull-down menu contains.

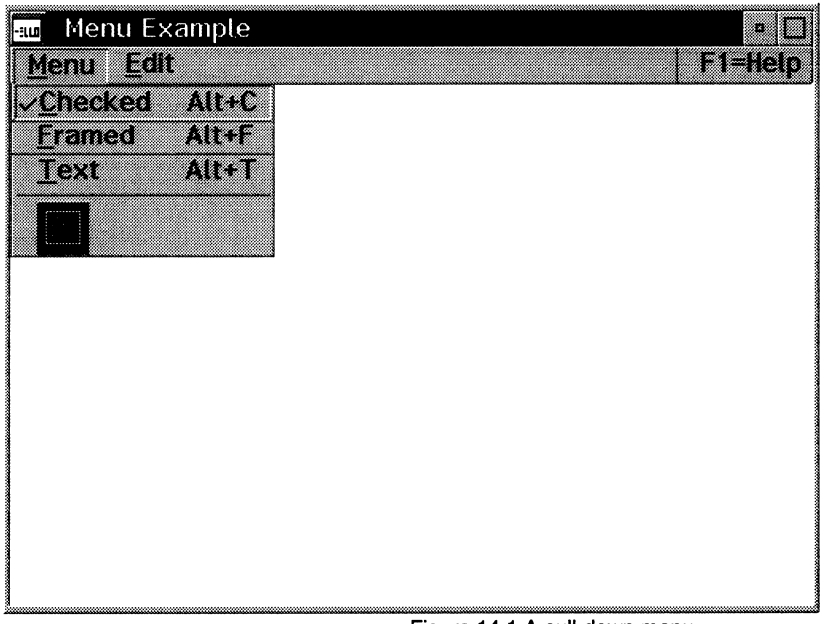



Figure 14.1 A pull-down menu.

Most users are familiar with the traditional *pull-down* menus. (See Figure 14.1.) This interface is common throughout many GUI environments. A pull-down menu should contain related choices. These choices extend from the menu bar when a particular menu bar choice is selected.

A *cascaded* menu is one that extends from a selected choice in a pull-down menu—kind of a tag-along pull-down menu. Cascaded menus can help to shorten long menus. Presentation Manager indicates the presence of a cascaded menu by a right arrow along the right edge of the pull-down menu.

A *pop-up* menu (see Figure 14.2) is a menu that pops up a list of choices for an object when some action is performed to trigger the menu. Pop-up menus are very common in 32-bit OS/2 and are an integral part of the object-oriented workplace shell. Pop-up menus normally are placed to the right of the object they pertain to, unless space does not permit; in such a case, the menu is placed wherever space permits.

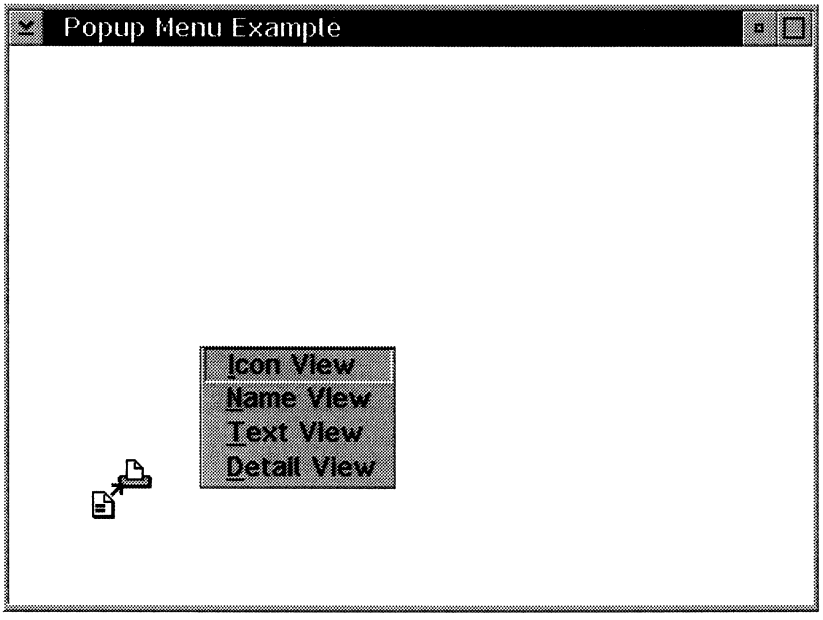



Figure 14.2 A pop-up menu.

## Menus: The Keyboard and the Mouse

Menus are no good to the user unless they are easy to understand and easy to get to. The mouse provides the easiest interaction with a menu. The user just selects the item by clicking the mouse on any item. If a pull-down menu is available, it will become visible.

The keys specified in Table 14.1 are important keystrokes to access menus.

Table 14.1 Menu Keystrokes

| Key | Action |
|---|---|
| ALT | Toggles the focus on the menu action bar. |
| Shift + ESC, Alt + spacebar | Causes the system menu to become visible. |
| F10 | Jumps to the next higher menu level. |
| ↑ (up arrow) | If the pull-down menu is not visible, causes it to become visible; if the pull-down menu is visible, will move to the previous menu item. |
| ↓ (down arrow) | If the pull-down menu is not visible, causes it to become visible; if the pull-down menu is visible, will move to the next menu item. |
| ←(left arrow) | Will move to the next item on the action bar; the system menu is included in the items this key will cycle through. |

| Key | Action |
|---|---|
| →(right arrow) | Will move to the previous item on the action bar; the system menu is included in the items this key will cycle through. |
| Enter | Selects the current item; if the item is on the action bar, the pull-down menu will become visible. |
| Character keys | Moves to the menu item that has corresponding mnemonic key. |

## Mnemosyne's Mnemonics

A mnemonic key is similar to an accelerator key, only not quite as powerful. A mnemonic will select the first menu item with the specified character as its mnemonic key. If the item has a pull-down menu associated with it, the pull-down menu will become visible. A mnemonic key usually corresponds to a character in the menu item text. The first letter is used if possible; otherwise, some meaningful character in the text is used. A mnemonic is indicated by an underlined character. The tilde character (~) in a menu template in the resource file indicates that the character to follow is a mnemonic key. No other definitions are necessary in the program; the menu control processing will handle the action of the mnemonics.

## Menu Styles

Table 14.2 Menu Styles

| Styles | Description |
|---|---|
| MS_ACTIONBAR | Creates a menu bar. |
| MS_CONDITIONALCASCADE | Creates a cascaded menu that will become visible only when the arrow to the right of the menu item is selected. |
| MS_TITLEBUTTON | Creates a push button along the menu bar. |
| MS_VERTICALFLIP | Causes a pull-down menu to be placed above the action bar, space permitting; if space is not available, the menu is placed below the action bar. |

The choices available in a menu are known as menu items. These menu items are not really a window, but they do have a special set of styles associated with them. Table 14.2 lists these styles.

## Menu Item Styles

Table 14.3 Menu Item Styles

| Item Styles | Description |
|---|---|
| MIS_SUBMENU | Creates a submenu. |
| MIS_SEPARATOR | Inserts a horizontal bar in the menu; a separator is a dummy item and cannot be selected, enabled, or disabled. |
| MIS_BITMAP | A bitmap instead of text. |
| MIS_TEXT | A text string. |
| MIS_BUTTONSEPARATOR | Creates a menu item that is separate from the other menus. Is placed on the far right on a menu bar and as the last item in a pull-down menu. A vertical separator is drawn between this item and the previous items. |
| MIS_BREAK | Creates a new row (on a menu bar) or a new column (on a pull-down menu). |
| MIS_BREAKSEPARATOR | Just like MIS_BREAK, except that a line is drawn between the new row or column. |

| Item Styles | Description |
|---|---|
| MIS_SYSCOMMAND | Notifies the owner through a WM_SYSCOMMAND message rather than a WM_COMMAND message. |
| MIS_OWNERDRAW | Creates an owner-drawn menu item; WM_DRAWITEM messages are sent whenever the menu item is to be drawn. |
| MIS_HELP | Sends a WM_HELP message to its owner, rather than a WM_COMMAND message. |
| MIS_STATIC | Creates an unselectable menu item that should be used for information purposes only. |

The following example program shows how to create a pull-down menu. When the menu item is selected, a message box is displayed containing information about the selected item.

**MENU.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "menu.h"
#define  CLS_CLIENT     "MyClass"
VOID displayMenuInfo(HWND hwndMenu,USHORT usMenuItem,HWND
                     hwndClient);

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{

   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   BOOL              bLoop;
   QMSG              qmMsg;
   LONG              lWidth,lHeight;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    0,
                    0);

   ulFlags = FCF_STANDARD&~FCF_SHELLPOSITION;

   /*********************************************************/
   /* create frame window                                   */
   /*********************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Menu Example",
                                  0,
                                  NULLHANDLE,
                                  RES_FRAME,
                                  NULL);
```

```
   /************************************************************/
   /* get screen height and width                              */
   /************************************************************/

   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /************************************************************/
   /* if failed, display error, and set to default value       */
   /************************************************************/

   if (!lWidth || !lHeight)
   {
      lWidth = 640;
      lHeight = 480;
   }

   if (hwndFrame != NULLHANDLE)
   {

      /************************************************************/
      /* set window position                                      */
      /************************************************************/

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      lWidth/8,
                      lHeight/8,
                      lWidth/8*6,
                      lHeight/8*6,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
            HPS               hpsWnd;
            HBITMAP           hbmBitmap;
```

```
        MENUITEM        miItem;
        HWND            hwndMenu;

        hpsWnd = WinGetPS(hwndClient);

        /************************************************/
        /* load bitmap from resource file               */
        /************************************************/

        hbmBitmap = GpiLoadBitmap(hpsWnd,
                                  NULLHANDLE,
                                  IDB_BITMAP,
                                  32,
                                  32);
        WinReleasePS(hpsWnd);

        /************************************************/
        /* set up MENUITEM data structure               */
        /************************************************/

        miItem.iPosition = 0;
        miItem.afStyle = MIS_BITMAP;
        miItem.afAttribute = 0;
        miItem.id = IDM_BITMAP;
        miItem.hwndSubMenu = NULLHANDLE;
        miItem.hItem = hbmBitmap;

        hwndMenu = WinWindowFromID(WinQueryWindow(hwndClient,
                                                  QW_PARENT),
                                   FID_MENU);

        /************************************************/
        /* Set MENUITEM                                 */
        /************************************************/

        WinSendMsg(hwndMenu,
                   MM_SETITEM,
                   MPFROM2SHORT(0,
                                TRUE),
                   MPFROMP(&miItem));
    }
    break;
case  WM_PAINT :
    {
        HPS             hpsPaint;
        RECTL           rclPaint;

        hpsPaint = WinBeginPaint(hwndClient,
                                 NULLHANDLE,
                                 &rclPaint);

        /************************************************/
        /* clear window                                 */
        /************************************************/

        WinFillRect(hpsPaint,
                    &rclPaint,
                    SYSCLR_WINDOW);
        WinEndPaint(hpsPaint);
    }
    break;
case  WM_COMMAND :
    switch (SHORT1FROMMP(mpParm1))
    {
        case  IDM_ITEM1 :
        case  IDM_ITEM2 :
        case  IDM_ITEM3 :
        case  IDM_BITMAP :
```

```
case  IDM_CUT :
case  IDM_COPY :
   {
      HWND              hwndFrame;
      HWND              hwndMenu;
      USHORT            usAttr;
      MRESULT           mrReply;
      CHAR              achText[64];

      /*****************************************/
      /* get the menu window handle            */
      /*****************************************/

      hwndFrame = WinQueryWindow(hwndClient,
                                 QW_PARENT);
      hwndMenu = WinWindowFromID(hwndFrame,
                                 FID_MENU);

      /*****************************************/
      /* show menuitem information             */
      /*****************************************/

      displayMenuInfo(hwndMenu,
                      SHORT1FROMMP(mpParm1),
                      hwndClient);

      /*****************************************/
      /* if this is the checked menu item, toggle  */
      /* it                                    */
      /*****************************************/

      if (SHORT1FROMMP(mpParm1) == IDM_ITEM1)
      {
         mrReply = WinSendMsg(hwndMenu,
                              MM_QUERYITEMATTR,
                              MPFROM2SHORT(IDM_ITEM1,
                                           TRUE),
                              MPFROMSHORT(MIA_CHECKED
                              ));

         usAttr = SHORT1FROMMR(mrReply);

         /*****************************************/
         /* toggle checked bit                    */
         /*****************************************/

         usAttr ^= MIA_CHECKED;

         if (usAttr != 0)
         {
            strcpy(achText,
                   " ~Checked item\tAlt + C");
         }
         else
         {
            strcpy(achText,
                   " ~Unchecked item\tAlt + C");
         }                 /* endif               */

         /*****************************************/
         /* change to newly toggled state         */
         /*****************************************/

         WinSendMsg(hwndMenu,
                    MM_SETITEMATTR,
                    MPFROM2SHORT(IDM_ITEM1,
                                 TRUE),
```

```
                                MPFROM2SHORT(MIA_CHECKED,
                                             usAttr));

                        /****************************************/
                        /* change text to reflect change of state */
                        /****************************************/

                        WinSendMsg(hwndMenu,
                                   MM_SETITEMTEXT,
                                   MPFROMSHORT(IDM_ITEM1),
                                   MPFROMP(achText));
                     }                          /* endif                */
                  }
                  break;
               default  :
                  return  WinDefWindowProc(hwndClient,
                                           ulMsg,
                                           mpParm1,
                                           mpParm2);

            }                                   /* endswitch            */
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

      }                                         /* endswitch            */
      return  MRFROMSHORT(FALSE);
}

VOID displayMenuInfo(HWND hwndMenu,USHORT usMenuItem,HWND
                     hwndClient)
{
   USHORT           usAllStyles;
   USHORT           usAttr;
   USHORT           usSzText;
   CHAR             achItemText[32];
   CHAR             achText[128];

   /*********************************************************/
   /* look for all different kinds of attributes            */
   /*********************************************************/

   usAllStyles = MIA_NODISMISS|MIA_FRAMED|MIA_CHECKED|
      MIA_DISABLED|MIA_HILITED;

   usAttr = SHORT1FROMMR(WinSendMsg(hwndMenu,
                                    MM_QUERYITEMATTR,
                                    MPFROM2SHORT(usMenuItem,
                                                 TRUE),
                                    MPFROMSHORT(usAllStyles)));

   /*********************************************************/
   /* query menu item text                                  */
   /*********************************************************/

   usSzText = SHORT1FROMMR(WinSendMsg(hwndMenu,
                                      MM_QUERYITEMTEXT,
                                      MPFROM2SHORT(usMenuItem,
                                                   30),
                                      MPFROMP(achItemText)));

   sprintf(achText,
           "Menu Item: \"%s\"\nMenu Item Styles are: 0x%04x",
           usSzText?achItemText:" (null) ",
           usAttr);
```

```
   /*********************************************************/
   /* display information                                   */
   /*********************************************************/

   WinMessageBox(HWND_DESKTOP,
                 hwndClient,
                 achText,
                 "Menu Information",
                 0,
                 MB_OK);

   return ;
}
```

**MENU.RC**

```
#include <os2.h>
#include "menu.h"

ICON RES_FRAME MENU.ICO
BITMAP IDB_BITMAP MENU.BMP

MENU RES_FRAME
{
   SUBMENU "~Menu", IDM_SUB1
   {
      MENUITEM "~Checked\tAlt+C", IDM_ITEM1, MIS_TEXT, MIA_CHECKED
      MENUITEM "~Framed\tAlt+F", IDM_ITEM2, MIS_TEXT, MIA_FRAMED
      MENUITEM "~Text\tAlt+T", IDM_ITEM3, MIS_TEXT
      MENUITEM SEPARATOR
      MENUITEM "", IDM_BITMAP
   }
   SUBMENU "~Edit", IDM_EDIT
   {
      MENUITEM "~Cut", IDM_CUT
      MENUITEM "C~opy", IDM_COPY
      MENUITEM "~Paste", IDM_PASTE, MIS_TEXT, MIA_DISABLED
   }
   MENUITEM "F1=Help", IDM_HELP, MIS_HELP | MIS_BUTTONSEPARATOR
}

ACCELTABLE RES_FRAME
{
   "c", IDM_ITEM1, ALT
   "f", IDM_ITEM2, ALT
   "t", IDM_ITEM3, ALT
}
```

**MENU.H**

```
#define RES_FRAME               256

#define IDM_SUB1                 512
#define IDM_ITEM1                513
#define IDM_ITEM2                514
#define IDM_ITEM3                515
#define IDM_BITMAP               516
#define IDM_EDIT                 528
#define IDM_CUT                  529
#define IDM_COPY                 530
#define IDM_PASTE                531
#define IDM_HELP                 544

#define IDB_BITMAP              1024
```

**MENU.MAK**

```
MENU.EXE:                       MENU.OBJ \
                                MENU.RES
        LINK386 @<<
MENU
MENU
MENU
OS2386
MENU
<<
        RC MENU.RES MENU.EXE

MENU.RES:                       MENU.RC \
                                MENU.H
        RC -r MENU.RC MENU.RES

MENU.OBJ:                       MENU.C \
                                MENU.H
        ICC -C+ -Kb+ -Ss+ MENU.C
```

**MENU.DEF**

```
NAME MENU WINDOWAPI

DESCRIPTION 'Menu example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

# The Resource File

The menu for a frame window can be created two ways: either statically, using the resource file, or dynamically, using *WinCreateWindow* with the class WC_MENU. The easiest way is to create a menu in the resource file, and this example will do just that.

```
MENU RES_CLIENT
```

The MENU keyword in a resource file indicates that a menu is being defined. The next word is the resource ID, RES_CLIENT. All resources including icons, accelerator tables, and menus, that are attached to the frame window share the same resource ID. This resource ID will automatically attach all resources indicated by the FCF_* flags used in *WinCreateStdWindow*. This can cause the function to fail if a resource is defined with the FCF_ flag and not in the .RC file.

```
{
   SUBMENU "~Menu", IDM_SUB1
   {
      MENUITEM "~Checked\tAlt+C", IDM_ITEM1, MIS_TEXT, MIA_CHECKED
      MENUITEM "~Framed\tAlt+F", IDM_ITEM2, MIS_TEXT, MIA_FRAMED
      MENUITEM "~Text\tAlt+T", IDM_ITEM3, MIS_TEXT
      MENUITEM SEPARATOR
      MENUITEM "", IDM_BITMAP
   }
   SUBMENU "~Edit", IDM_EDIT
   {
      MENUITEM "~Cut", IDM_CUT
      MENUITEM "C~opy", IDM_COPY
      MENUITEM "~Paste", IDM_PASTE, MIS_TEXT, MIA_DISABLED
   }
   MENUITEM "F1=Help", IDM_HELP, MIS_HELP | MIS_BUTTONSEPARATOR
}
```

The \t characters on the MENUITEM indicate that a tab is placed between the text and the text that follows. The text following the tab is the information on the accelerator key. Just because we have defined the menu text to indicate an accelerator key does not guarantee its existence.

The options after the resource IDs are the menu item styles. A comma is used to separate the styles from the menu item attributes. Attributes are used to describe the state of a menu item and are designed to be turned on and off on the fly. The previous example program contains examples of five different kinds of menu items: Checked, Text, Framed, Bitmap, and Disabled. A menu item that is checked or unchecked is an example of a menu item attribute. The attributes specified in Table 14.4 are available.

## Menu Item Attributes

Table 14.4 Menu Item Attributes

| Item Attribute | Description |
|---|---|
| MIA_HILITED | The menu item is selected. |
| MIA_CHECKED | A check will appear next to this menu item if TRUE. |
| MIA_DISABLED | The menu item will appear in a grayed, disabled state. |
| MIA_FRAMED | The menu item is enclosed within a frame. |
| MIA_NODISMISS | The pull-down menu containing this menu item will not be dismissed until told to do so. |

## Creating the Menu Bitmap

There are two ways to use a bitmap as a menu item. One is to include it in the resource file; the other is to load it during the message processing. In this example, we'll choose the latter method.

```
hbmBitmap = GpiLoadBitmap ( hpsWnd,
                            NULLHANDLE,
                            IDB_BITMAP,
                            32,
                            32 ) ;
```

For more information on *GpiLoadBitmap*, see Chapter 12.

The bitmap handle, *hbmBitmap*, is returned from *GpiLoadBitmap*.

```
typedef struct _MENUITEM    /* mi */
{
   SHORT   iPosition;
   USHORT  afStyle;
   USHORT  afAttribute;
   USHORT  id;
   HWND    hwndSubMenu;
   ULONG   hItem;
} MENUITEM;
typedef MENUITEM *PMENUITEM;
```

A MENUITEM structure is used to tell the menu how this menu item is to appear. As always when passing structures, all fields must be initialized. For the menu item style, we use MIS_BITMAP. The ID is IDM_BITMAP. *hItem* is the handle to the item—in this case, *hbmBitmap*.

```
miItem.iPosition = 0 ;
miItem.afStyle = MIS_BITMAP ;
miItem.afAttribute = 0 ;
miItem.id = IDM_BITMAP ;
miItem.hwndSubMenu = NULLHANDLE ;
miItem.hItem = hbmBitmap ;
```

In the MENU.RC file, a spot was created for the IDM_BITMAP menu item. The MM_SETITEM message is sent to finish the job.

```
WinSendMsg ( hwndMenu,
             MM_SETITEM,
             MPFROM2SHORT ( 0, TRUE ) ,
             MPFROMP ( &miItem )) ;
```

*mpParm1* is composed of two USHORTS. The first is always 0, and the second is a flag indicating that submenus are to be included in the search. We do want to include submenus. The second message parameter is a pointer to the MENUITEM structure.

## The Client Window Procedure *ClientWndProc*

The client window procedure is where all of the menu handling is done. The WM_COMMAND message is sent to the owner, *hwndClient*, whenever the user has selected some item from the menu, using the mouse, keyboard, or accelerator key. The example finds out which menu item is selected and displays a message box with information about the item. The menu item IDM_ITEM1 will have the check mark toggled on and off whenever it is selected.

```
case  WM_COMMAND :
        switch (SHORT1FROMMP(mpParm1))
        {
           case  IDM_ITEM1 :
           case  IDM_ITEM2 :
           case  IDM_ITEM3 :
           case  IDM_BITMAP :
           case  IDM_CUT :
           case  IDM_COPY :
              {
                 HWND              hwndFrame;
                 HWND              hwndMenu;
                 USHORT            usAttr;
                 MRESULT           mrReply;
                 CHAR              achText[64];

                 hwndFrame = WinQueryWindow(hwndClient,
                                            QW_PARENT);
                 hwndMenu = WinWindowFromID(hwndFrame,
                                            FID_MENU);
```

The menu item ID is contained in *mpParm1* of the WM_COMMAND message. After the ID is obtained, we obtain the menu window handle. The menu handle is used later. *WinWindowFromID* will return the menu window handle when the special ID, FID_MENU, is used. The first parameter is the parent of the menu, the frame window.

```
if (SHORT1FROMMP(mpParm1) == IDM_ITEM1)
{
   mrReply = WinSendMsg(hwndMenu,
                        MM_QUERYITEMATTR,
                        MPFROM2SHORT(IDM_ITEM1,
                                     TRUE),
                        MPFROMSHORT(MIA_CHECKED
                        ));

   usAttr = SHORT1FROMMR(mrReply);
```

If the menu item ID is IDM_ITEM1, we query whether the MIA_CHECKED bit is set, using the message MM_QUERYITEMATTR. *mpParm1* consists of two USHORTS. The lower bytes are the menu item ID to query, IDM_ITEM1. The upper bytes indicate whether to include submenus. This is applicable when you want to query all menu items on a pull-down, or sublevel, menu. *mpParm2* is the attribute mask for the query. We want to know only whether the MIA_CHECKED bit is set, so this will be the mask we use. A mask can be a collection of attributes OR'ed together or only one. The value of the bit is returned in the variable *usAttr*.

```
usAttr ^= MIA_CHECKED;
```

Once we know whether the menu item is checked, we want to reverse the state of the MIA_CHECKED bit in order to toggle the check mark.

```
if (usAttr != 0)
{
   strcpy(achText,
          " ~Checked item\tAlt + C");
}
else
{
   strcpy(achText,
          " ~Unchecked item\tAlt + C");
}                /* endif                */

WinSendMsg(hwndMenu,
           MM_SETITEMATTR,
           MPFROM2SHORT(IDM_ITEM1,
                        TRUE),
           MPFROM2SHORT(MIA_CHECKED,
                        usAttr));

WinSendMsg(hwndMenu,
           MM_SETITEMTEXT,
           MPFROMSHORT(IDM_ITEM1),
           MPFROMP(achText));
```

The next thing to do is to set the menu with the new menu item state, and also update the menu item text to reflect the change. The checked state is determined by AND'ing *usAttr* and MIA_CHECKED. The message MM_SETITEMTEXT is used to set the menu item text to the new string. *mpParm1* is set to the menu item ID, IDM_ITEM1. *mpParm2* is a pointer to the text string. The message MM_SETITEMATTR is used to set the menu item attribute to the new value in *usAttr*. The message parameters are equivalent to the MM_QUERYITEMATTR message parameters, except that MM_SETITEMATTR has an extra SHORT in *mpParm2* that contains attribute data.

## The User Function *displayMenuInfo*

After the user selects a menu item, a message box is popped up to display various bits of information about the menu item. The menu item attributes are found using MM_QUERYITEMATTR. Instead of using just one menu item attribute mask, the values MIA_NODISMISS, MIA_FRAMED, MIA_CHECKED, MIA_DISABLED, and MIA_HILITED are OR'ed together.

```
usAllStyles = MIA_NODISMISS | MIA_FRAMED | MIA_CHECKED |
              MIA_DISABLED | MIA_HILITED ;

usAttr = SHORT1FROMMR ( WinSendMsg ( hwndMenu,
                        MM_QUERYITEMATTR,
                        MPFROM2SHORT ( usMenuItem, TRUE ) ,
                        MPFROMSHORT ( usAllStyles )) ) ;

usSzText = SHORT1FROMMR ( WinSendMsg ( hwndMenu,
                          MM_QUERYITEMTEXT,
                          MPFROM2SHORT ( usMenuItem, 30 ) ,
                          MPFROMP ( achItemText )) ) ;
```

The return from the message will yield the state of all these attributes OR'ed together. MM_QUERYITEMTEXT is used to query the menu item text. *mpParm1* is two USHORTS. The lower bytes contain the menu item ID; the upper bytes contain the length of the text input buffer, *achItemText*. The second message parameter is a pointer to the text input buffer.

The last step is to call *WinMessageBox* to display the menu item information.

## Pop-up Menus

The following example will demonstrate how to create a pop-up menu suitable for the OS/2 Warp environment. An icon is created on the client window. If the user clicks the context menu mouse button (the right one by default) on the icon, a pop-up menu will appear.

**POPUP.C**

```
#define  INCL_WIN
#include <os2.h>
#include "popup.h"
#include "stdlib.h"
MRESULT EXPENTRY ClientWndProc(HWND hwnd,ULONG msg,MPARAM mp1,
                               MPARAM mp2);

#define  CLS_CLIENT    "MyClass"
typedef          struct
{
   HWND              hwndMenu;

   HPOINTER          hptrFileIcon;
} MENUDATA,*PMENUDATA;

INT main(VOID)
{

   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   BOOL              bLoop;
   QMSG              qmMsg;
```

```
LONG            lWidth,lHeight;

habAnchor = WinInitialize(0);
hmqQueue = WinCreateMsgQueue(habAnchor,
                             0);

WinRegisterClass(habAnchor,
                 CLS_CLIENT,
                 ClientWndProc,
                 0,
                 sizeof(PVOID));

/*********************************************************/
/* create frame window                                   */
/*********************************************************/

ulFlags = FCF_TASKLIST|FCF_TITLEBAR|FCF_SYSMENU|FCF_MINMAX|
   FCF_SIZEBORDER|FCF_SHELLPOSITION;

hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                               0,
                               &ulFlags,
                               CLS_CLIENT,
                               "Popup Menu Example",
                               0,
                               NULLHANDLE,
                               0,
                               NULL);

/*********************************************************/
/* get screen height and width                           */
/*********************************************************/

lWidth = WinQuerySysValue(HWND_DESKTOP,
                          SV_CXSCREEN);

lHeight = WinQuerySysValue(HWND_DESKTOP,
                           SV_CYSCREEN);

/*********************************************************/
/* if failed, display error, and set to default value    */
/*********************************************************/

if (!lWidth || !lHeight)
{
   lWidth = 640;
   lHeight = 480;
}

if (hwndFrame != NULLHANDLE)
{

   /*****************************************************/
   /* set window position                               */
   /*****************************************************/

   WinSetWindowPos(hwndFrame,
                   NULLHANDLE,
                   lWidth/8,
                   lHeight/8,
                   lWidth/8*6,
                   lHeight/8*6,
                   SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
```

```
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile                */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                   */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PMENUDATA        pmdMenuData;

   switch (ulMsg)
   {

      case  WM_CREATE :
         {
            pmdMenuData = malloc(sizeof(MENUDATA));
            WinSetWindowPtr(hwndClient,
                            0,
                            pmdMenuData);

            pmdMenuData->hwndMenu = WinLoadMenu(hwndClient,
                                                NULLHANDLE,
                                                IDM_POPUP);
            pmdMenuData->hptrFileIcon = WinLoadFileIcon
               ("POPUP.EXE",
                FALSE);
         }
         break;

      case  WM_DESTROY :

         pmdMenuData = WinQueryWindowPtr(hwndClient,
                                         0);
         if (pmdMenuData)
         {
            if (pmdMenuData->hptrFileIcon != NULLHANDLE)
            {
               WinFreeFileIcon(pmdMenuData->hptrFileIcon);
            }                            /* endif                   */
            if (pmdMenuData->hwndMenu)
               WinDestroyWindow(pmdMenuData->hwndMenu);
            free(pmdMenuData);
         }                               /* endif                   */
         break;

      case  WM_PAINT :
         {
            HPS              hpsPaint;
            RECTL            rclInvalid;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     &rclInvalid);
            WinFillRect(hpsPaint,
```

```
                    &rclInvalid,
                    SYSCLR_WINDOW);
         pmdMenuData = WinQueryWindowPtr(hwndClient,
                                         0);

         if (pmdMenuData->hptrFileIcon != NULLHANDLE)
         {
            WinDrawPointer(hpsPaint,
                           50,
                           50,
                           pmdMenuData->hptrFileIcon,
                           DP_NORMAL);
         }                          /* endif                */
         WinEndPaint(hpsPaint);
      }
      break;
   case  WM_CONTEXTMENU :
      {
         POINTL            ptlMouse;
         RECTL             rclIcon;
         HAB               habAnchor;
         BOOL              bInside;
         BOOL              bKeyboardUsed;

         pmdMenuData = WinQueryWindowPtr(hwndClient,
                                         0);
         habAnchor = WinQueryAnchorBlock(hwndClient);
         bKeyboardUsed = SHORT2FROMMP(mpParm2);

         /*-------------------------------------------------*/
         /* If the mouse was used, check to see if the      */
         /* pointer is over the icon, else always display   */
         /* the menu.                                       */
         /*-------------------------------------------------*/

         if (!bKeyboardUsed)
         {

            rclIcon.xLeft = 50;
            rclIcon.xRight = rclIcon.xLeft+WinQuerySysValue
               (HWND_DESKTOP,
                SV_CXICON);

            rclIcon.yBottom = 50;
            rclIcon.yTop = rclIcon.yBottom+WinQuerySysValue
               (HWND_DESKTOP,
                SV_CYICON);

            ptlMouse.x = (LONG)SHORT1FROMMP(mpParm1);
            ptlMouse.y = (LONG)SHORT2FROMMP(mpParm1);

            bInside = WinPtInRect(habAnchor,
                                  &rclIcon,
                                  &ptlMouse);
         }
         else
         {

            bInside = TRUE;
            ptlMouse.x = 100;
            ptlMouse.y = 100;

         }                          /* endif                */
```

```
            if ((bInside) && (pmdMenuData->hwndMenu))
            {
               WinPopupMenu(hwndClient,
                            hwndClient,
                            pmdMenuData->hwndMenu,
                            ptlMouse.x,
                            ptlMouse.y,
                            IDM_ICON,
                            PU_POSITIONONITEM|PU_KEYBOARD|
                               PU_MOUSEBUTTON1|PU_MOUSEBUTTON2);
            }                          /* endif                */
         }
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}
```

## POPUP.RC

```
#include <os2.h>
#include "popup.h"

MENU IDM_POPUP
{
   MENUITEM "~Icon View", IDM_ICON
   MENUITEM "~Name View", IDM_NAME
   MENUITEM "~Text View", IDM_TEXT
   MENUITEM "~Detail View", IDM_DETAIL
}
```

## POPUP.H

```
#define IDM_POPUP                256
#define IDM_ICON                 257
#define IDM_NAME                 258
#define IDM_TEXT                 259
#define IDM_DETAIL               260
```

## POPUP.MAK

```
POPUP.EXE:                       POPUP.OBJ \
                                 POPUP.RES
        LINK386 @<<
POPUP
POPUP
POPUP
OS2386
POPUP
<<
        RC POPUP.RES POPUP.EXE

POPUP.RES:                       POPUP.RC \
                                 POPUP.H
        RC -r POPUP.RC POPUP.RES

POPUP.OBJ:                       POPUP.C \
                                 POPUP.H
        ICC -C+ -Kb+ -Ss+ POPUP.C
```

**POPUP.DEF**

```
NAME POPUP WINDOWAPI

DESCRIPTION 'Popup example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

# Creating a Pop-up Menu

```
pmdMenuData = malloc(sizeof(MENUDATA));
WinSetWindowPtr(hwndClient,
                0,
                pmdMenuData);

pmdMenuData->hwndMenu = WinLoadMenu(hwndClient,
                                    NULLHANDLE,
                                    IDM_POPUP);
```

The pop-up menu is created almost exactly as a regular menu is. The pop-up template contains the same keywords and definitions as the regular pull-down template. When the client window is being created (the WM_CREATE processing), the menu template is loaded.

```
HWND WinLoadMenu(HWND hwndFrame,
                 HMODULE hmod,
                 ULONG idMenu);
```

*WinLoadMenu* has three parameters. *hwndFrame* is the owner and parent window handle. *hmod* is the resource identifier if the menu resource is located in a .DLL, and *idMenu* is the menu resource ID. *WinLoadMenu* returns a menu handle that will be used later in the *WinPopupMenu* function. For now, it is stored in the window word of the client area. One performance note here: We could have used *WinLoadMenu* in the WM_CONTEXTMENU processing, because WM_CREATE is called once and WM_CONTEXTMENU is called as many times as the user chooses, considerable time and system resources are saved if we load the menu in the WM_CREATE processing. Whenever possible, programmers should keep message processing as lean as possible and be careful of loading resources multiple times.

# I Think I Can, I Think Icon

```
pmdMenuData->hptrFileIcon = WinLoadFileIcon("POPUP.EXE",
                                            FALSE);
```

One of the functions introduced in OS/2 2.0 is *WinLoadFileIcon*. This is a nifty function to "lift" an icon from some file to use in a program. This example takes the file icon associated with itself and paints it on the client window.

```
HPOINTER WinLoadFileIcon(PSZ pszFileName,
                         BOOL fPrivate);
```

*WinLoadFileIcon* has two parameters. The first is the file name. The second is a flag that indicates whether the icon needs to be "public" or "private." A "public" icon is much easier on system resources,

but it is a read-only version of the icon. That's all that this example needs. A pointer handle, *hptrFileIcon*, to the icon is returned. Once again, the handle is stored in the client's window word for future use.

```
WinDrawPointer(hpsPaint,
               50,
               50,
               pmdMenuData->hptrFileIcon,
               DP_NORMAL);
```

*WinDrawPointer* actually will paint the icon on the client window. For more information on this function, see Chapter 12.

## Popping Up a Menu

```
rclIcon.xLeft = 50;
rclIcon.xRight = rclIcon.xLeft+WinQuerySysValue
   (HWND_DESKTOP,
    SV_CXICON);

rclIcon.yBottom = 50;
rclIcon.yTop = rclIcon.yBottom+WinQuerySysValue
   (HWND_DESKTOP,
    SV_CYICON);

ptlMouse.x = (LONG)SHORT1FROMMP(mpParm1);
ptlMouse.y = (LONG)SHORT2FROMMP(mpParm1);

bInside = WinPtInRect(habAnchor,
                      &rclIcon,
                      &ptlMouse);
```

In this example, when the user clicks the context menu mouse button or uses the context menu keystroke, we'll pop up a menu. The message we'll use to track that event is WM_CONTEXTMENU.

```
BOOL WinPtInRect(HAB hab,
                 PRECTL prcl,
                 PPOINTL pptl);
```

We use *WinPtInRect* to determine if the mouse is over the icon that we have drawn already. *hab* is the anchor block handle. *prcl* is a pointer to the points region of the rectangle coordinates. *pptl* is a pointer to the points region. If the point lies within the rectangle, TRUE is returned. If the mouse is over the icon, we pop up the menu.

## The Workhorse Function *WinPopupMenu*

```
WinPopupMenu ( hwndClient,
               hwndClient,
               pmdMenuData->hwndMenu,
               ptlMouse.x,
               ptlMouse.y,
               IDM_ICON,
               PU_POSITIONONITEM | PU_KEYBOARD |
               PU_MOUSEBUTTON1 | PU_MOUSEBUTTON2 ) ;
```

The pop-up menu actually is made visible by *WinPopupMenu*. This function handles all the user I/O and returns WM_COMMAND messages to the owner window, just as a regular pull-down menu does.

```
BOOL WinPopupMenu(HWND hwndParent,
                  HWND hwndOwner,
                  HWND hwndMenu,
                  LONG x,
                  LONG y,
                  LONG idItem,
                  ULONG fs);
```

The first and second parameters are the parent and owner windows, respectively. The client window, *hwndClient*, is used for both. The next parameter is the menu handle of the popup menu. The next two parameters are the x and y coordinates at which to place the menu. The last two parameters are used to control the initial display state and user interface for the menu. IDM_ICON is the menu item we want to be selected initially.

The last parameter is a collection of flags. Table 14.5 specifies the flags available.

Table 14.5 Popup Menu Flags

| Flag | Description |
|---|---|
| PU_POSITIONONITEM | Will cause the ID specified in the previous parameter to appear directly above where the mouse pointer is. This flag overrides the x, y coordinates as placement of the menu. It also causes the specified menu item ID to appear selected when the pop-up menu appears. |
| PU_KEYBOARD | Lets the user use the keyboard keys to traverse the menu choices and select an item. |
| PU_MOUSEBUTTON2 | Enables the user to use mouse button 2 to select a menu item. |
| PU_MOUSEBUTTON1 | Enables the user to use mouse button 1 to select a menu item. |

**Gotcha!**

For pop-up menus, the WM_INITMENU documentation does not state that the menu identifier for the top-level menu will be FID_MENU.

# Chapter 15

# List Boxes

A list box (see Figure 15.1) is a control that provides the user with a list of choices. Single or multiple items can be selected; the default is single. A list box can scroll horizontally, vertically, or both. List boxes, by default, contain only text entries, although they are not limited to only text.

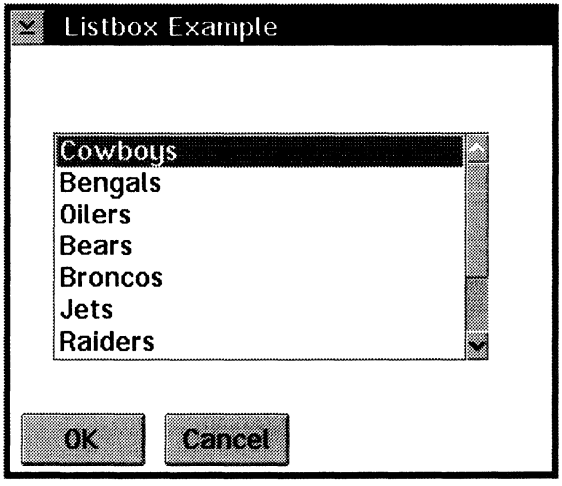



Figure 15.1 A list box control.

The items in a list box should be presented in some order meaningful to the user. A list box should be large enough to have six or eight choices visible at all times and wide enough to display an item of average width without horizontal scrolling. If multiple selection is supported, informative text should be provided to indicate the current number of selected items.

## List Box Styles

The styles presented in Table 15.1 can be used when creating a list box.

Table 15.1 List Box Styles

| Style | Description |
|---|---|
| LS_MULTIPLESEL | Supports selection of multiple items. |
| LS_OWNERDRAW | Generates a WM_DRAWITEM whenever certain parts are to be drawn. |
| LS_NOADJUSTPOS | Will not size the list box. |

| Style | Description |
|---|---|
| LS_HORZSCROLL | Will have a horizontal scroll bar along the bottom and will support horizontal scrolling. |
| LS_EXTENDEDSEL | Lets the user select more than one item using a point-end-point selection technique. |

## Extended Selection

List boxes also support a selection technique known as extended selection. Extended selection supports a "swiping" technique to select the list box items. Table 15.2 shows the keystrokes and mouse actions defined in an extended-selection list box.

Table 15.2 Extended Selection List Box Keystrokes

| Movement | Action |
|---|---|
| Click mouse button on object | Selects object; all others are deselected. |
| Drag mouse from start point of selection to end point of selection | Selects all objects in area; all other objects are deselected. |
| Press SHIFT key while cursor is at start point and use ↑ and ↓ keys to move to end point | Selects all objects in area; all other objects are deselected. |
| Click mouse button on object while pressing Ctrl key | Selects object; all other selected objects are left selected. |
| Press Ctrl+spacebar, or spacebar, while cursor is positioned at object | Selects object; all other selected objects are left selected. |
| Press Ctrl key while dragging mouse from start point of selection to end point of selection | Selects all objects in area; all other objects are deselected. |

The following LIST1 example program shows a very introductory list box program. This list box has the LS_MULTIPLESEL style and communicates with the client area to have the selections displayed in the window.

**LIST1.C**

```
#define  INCL_WIN
#define  INCL_GPI
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list1.h"


#define  CLS_CLIENT     "MyClass"
#define  NUM_ENTRIES    10

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);
MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);
VOID DisplayError(CHAR *pszText);
int main(void);
```

```
typedef          struct
{
   USHORT           ausListBoxSel[NUM_ENTRIES];
} LISTBOXINFO,*PLISTBOXINFO;

int main()
{
   HMQ              hmqQueue;
   HAB              habAnchor;
   ULONG            ulFlags;
   HWND             hwndFrame;
   HWND             hwndClient;
   QMSG             qmMsg;

   /************************************************************/
   /* initialization stuff                                     */
   /************************************************************/

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SYNCPAINT|CS_SIZEREDRAW|CS_CLIPCHILDREN,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_MENU|FCF_SIZEBORDER|
      FCF_MINMAX|FCF_SHELLPOSITION;

   /************************************************************/
   /* create the frame and client windows                      */
   /************************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Plain Listbox Example",
                                  0,
                                  (HMODULE)0,
                                  ID_CLIENT,
                                  &hwndClient);

   if (hwndFrame != NULLHANDLE)
   {
      while (WinGetMsg(habAnchor,
                       &qmMsg,
                       NULLHANDLE,
                       0,
                       0))
         WinDispatchMsg(habAnchor,
                        &qmMsg);
      WinDestroyWindow(hwndFrame);
   }                                    /* endif                */

   /************************************************************/
   /* clean up                                                 */
   /************************************************************/

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

```
MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)

{
   PLISTBOXINFO     pliInfo;
   BOOL             bReturn;

   /*********************************************************/
   /* retrieve listbox info from the client window          */
   /*********************************************************/

   pliInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {
      case  WM_CREATE :

         /*****************************************************/
         /* allocate window word                              */
         /*****************************************************/

         pliInfo = (PLISTBOXINFO)calloc(1,
                                        sizeof(LISTBOXINFO));
         if (!pliInfo)
         {
            DisplayError( "Memory Allocation Failure:1 ") ;
            return (MRFROMSHORT(TRUE));
         }                              /* endif              */

         /*****************************************************/
         /* assign window word                                */
         /*****************************************************/

         bReturn = WinSetWindowPtr(hwndClient,
                         QWL_USER,
                         pliInfo);
         if (!bReturn)
            DisplayError( "WinSetWindowPtr Failure:1") ;

         break;
      case  WM_DESTROY :

         /*****************************************************/
         /* free up window word                               */
         /*****************************************************/

         if (pliInfo)
            free(pliInfo);
         break;
      case  WM_PAINT :
         {
            HPS              hpsPresentationSpace;
            RECTL            rectInvalidRect,rclPaintRegion,
               rclNewPaint;
            USHORT           i;

            hpsPresentationSpace = WinBeginPaint(hwndClient,
                                                 NULLHANDLE,
                                                 &rectInvalidRect
               );

            /**************************************************/
            /* get window size                                */
            /**************************************************/
```

```
        bReturn = WinQueryWindowRect(hwndClient,
                          &rclPaintRegion);
        if (!bReturn){
            DisplayError("WinQueryWindowRect Failure:1");
            break;
        }

        /*************************************************/
        /* start 3/4 of the way across window            */
        /*************************************************/

        rclNewPaint.xLeft = (rclPaintRegion.xRight-
           rclPaintRegion.xLeft)  / 4 * 3;
        rclNewPaint.xRight = rclPaintRegion.xRight;

        /*************************************************/
        /* set the top and bottom coordinates            */
        /*************************************************/

        rclNewPaint.yBottom = rclPaintRegion.yBottom;
        rclNewPaint.yTop = rclPaintRegion.yTop;

        /*************************************************/
        /* fill the invalidated rectangle with white     */
        /*************************************************/

        WinFillRect(hpsPresentationSpace,
                    &rectInvalidRect,
                    CLR_WHITE);

        WinDrawText(hpsPresentationSpace,
                    -1,
                    "You have selected:",
                    &rclNewPaint,
                    0,
                    0,
                    DT_LEFT|DT_TEXTATTRS);

        /*************************************************/
        /* drop down one line                            */
        /*************************************************/

        rclNewPaint.yTop -= 15;

        /*************************************************/
        /* if item is selected, drop down one line, and  */
        /* draw the selected listbox item                */
        /*************************************************/

        for (i = 0; i < NUM_ENTRIES; i++)
           if (pliInfo->ausListBoxSel[i] == TRUE)
           {

              rclNewPaint.yTop -= 15;
              WinDrawText(hpsPresentationSpace,
                          -1,
                          pszListBoxEntry[i],
                          &rclNewPaint,
                          0,
                          0,
                          DT_LEFT|DT_TEXTATTRS);

           }                        /* end if selected      */
        WinEndPaint(hpsPresentationSpace);
        break;

    }
```

```
   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  IDM_DISPLAY :
            {
            ULONG ulReturn ;
            /***********************************************/
            /* load up dialog                              */
            /***********************************************/

            ulReturn = WinDlgBox(HWND_DESKTOP,
                      hwndClient,
                      DlgProc,
                      NULLHANDLE,
                      IDD_LISTBOX,
                      NULL);
            if (ulReturn == DID_ERROR)
                 DisplayError("WinDlgBox Failure:1") ;
            break;
            }
         case  IDM_EXIT :

            /***********************************************/
            /* close up window                             */
            /***********************************************/

            WinPostMsg(hwndClient,
                       WM_CLOSE,
                       MPVOID,
                       MPVOID);
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }
      break;

   case  WM_SIZE :
      WinPostMsg(hwndClient,
                 UM_LISTBOXSEL,
                 MPVOID,
                 MPVOID);
      break;
   case  UM_LISTBOXSEL :
      {

         SHORT            sSelect = 0;
         SHORT            sIndex = LIT_FIRST;
         HWND             hwndDlg;
         USHORT           i;

         /***********************************************/
         /* first set all to unselected                 */
         /***********************************************/

         for (i = 0; i < NUM_ENTRIES; i++)
            pliInfo->ausListBoxSel[i] = FALSE;

         hwndDlg = WinWindowFromID(HWND_DESKTOP,
                                   IDD_LISTBOX);

         /***********************************************/
         /* get selected items from listbox             */
         /***********************************************/

         while (sSelect != LIT_NONE && hwndDlg)
```

```
            {

               sSelect = (SHORT)WinSendDlgItemMsg(hwndDlg,
                                                  IDL_LISTBOX,
                                                LM_QUERYSELECTION
                                                      ,
                                                  MPFROMSHORT
                                                  (sIndex),
                                                  MPVOID);

               pliInfo->ausListBoxSel[sSelect] = TRUE;

               /***********************************************/
               /* set query to start at last selected item    */
               /***********************************************/

               sIndex = sSelect;
            }

            /***************************************************/
            /* invalidate the window                           */
            /***************************************************/

            WinInvalidateRect(hwndClient,
                              NULL,
                              FALSE);
            break;
         }
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);
   }                                   /* endswitch            */

   return (MRFROMSHORT(FALSE));
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)

{
   USHORT           i;

   switch (ulMsg)
   {

      case  WM_INITDLG :
         {
            HWND             hwndListBox;

            hwndListBox = WinWindowFromID(hwndDlg,
                                          IDL_LISTBOX);
            for (i = 0; i < NUM_ENTRIES; i++)
               WinInsertLboxItem(hwndListBox,
                                 LIT_END,
                                 pszListBoxEntry[i]);

            WinSendDlgItemMsg(hwndDlg,
                              IDL_LISTBOX,
                              LM_SELECTITEM,
                              MPFROMSHORT(0),
                              MPFROMSHORT(TRUE));
         }
         break;

      case  WM_COMMAND :
```

```
        {
           SHORT            sCommand;
           HWND             hwndClient;

           sCommand = SHORT1FROMMP(mpParm1);
           switch (sCommand)
           {
              case  DID_OK :
                 hwndClient = WinQueryWindow(hwndDlg,
                                             QW_OWNER);
                 if (!hwndClient){
                   DisplayError("WinQueryWindow Failure:1");
                   break;
                 }
                 WinPostMsg(hwndClient,
                            UM_LISTBOXSEL,
                            MPVOID,
                            MPVOID);

                 /*******************************************/
                 /* if hit OK, don't dismiss dialog         */
                 /*******************************************/

                 return (MRFROMSHORT(TRUE));

              case  DID_CANCEL :
                 hwndClient = WinQueryWindow(hwndDlg,
                                             QW_OWNER);
                 if (!hwndClient){
                   DisplayError("WinQueryWindow Failure:1");
                   break;
                 }
                 WinPostMsg(hwndClient,
                            UM_LISTBOXSEL,
                            MPVOID,
                            MPVOID);
                 WinDismissDlg(hwndDlg,
                               DID_CANCEL );
                 break;

           }                          /* end switch sCommand  */
           break;
        }
     default  :
        return (WinDefDlgProc(hwndDlg,
                              ulMsg,
                              mpParm1,
                              mpParm2));
   }                                  /* endswitch            */
   return (MRFROMSHORT(FALSE));
}


VOID DisplayError(CHAR *pszText)
{

   /*********************************************************/
   /* small function to display error string                */
   /*********************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
```

```
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}
```

## LIST1.RC

```
#include <os2.h>
#include "LIST1.H"

MENU ID_CLIENT
{
   SUBMENU "~File", -1
   {
      MENUITEM "~Display Dialog", IDM_DISPLAY
      MENUITEM SEPARATOR
      MENUITEM "E~xit", IDM_EXIT
   }
}


DLGTEMPLATE IDD_LISTBOX LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG  "Listbox Example", IDD_LISTBOX, 12, 6, 170, 107, WS_VISIBLE,
            FCF_SYSMENU | FCF_TITLEBAR
            PRESPARAMS PP_BACKGROUNDCOLORINDEX, CLR_WHITE
    BEGIN
        LISTBOX         IDL_LISTBOX, 14, 28, 135, 63, LS_MULTIPLESEL
            PRESPARAMS PP_BACKGROUNDCOLORINDEX, CLR_WHITE
        PUSHBUTTON      "OK", DID_OK, 3, 1, 40, 14
        PUSHBUTTON      "Cancel", DID_CANCEL, 48, 1, 40, 14
    END
END
```

## LIST1.H

```
#define UM_LOADDLG                  (WM_USER+1)
#define UM_LISTBOXSEL               (WM_USER+2)
#define IDD_LISTBOX                  200
#define IDL_LISTBOX                  201
#define ID_CLIENT                    202
#define IDM_DISPLAY                  203
#define IDM_EXIT                     204



CHAR *pszListBoxEntry[] = {
         "Cowboys",
         "Bengals",
         "Oilers",
         "Bears",
         "Broncos",
         "Jets",
         "Raiders",
         "Rams",
         "Giants",
         "Redskins" };
```

**LIST1.MAK**

```
LIST1.EXE:                      LIST1.OBJ \
                                  LIST1.RES
        LINK386 @<<
LIST1
LIST1
LIST1
OS2386
LIST1
<<
        RC LIST1.RES LIST1.EXE

LIST1.RES:                      LIST1.RC \
                                  LIST1.H
        RC -r LIST1.RC LIST1.RES

LIST1.OBJ:                      LIST1.C \
                                  LIST1.H LIST1.MAK
        ICC -C+ -Kb+ -Ss+ LIST1.C
```

**LIST1.DEF**

```
NAME LISTBOX WINDOWAPI
DESCRIPTION 'Listbox example.
             Copyright (c) 1992-1995 by Kathleen Panov.
             All rights reserved.'
```

In the LIST1 sample program, the dialog box will post a message, UM_LISTBOXSEL, to the client area when the OK button is pressed. When the client area receives this message, it queries the list box to determine which items have been selected. These items are stored in the user-defined window word area for the client window. Also, a flag, *fSelectedItems*, is set to indicate items have been selected.

When the WM_PAINT message is received, the client area is cleared. If the flag *fSelectedItems* is set, the items in the window word are written to the client area.

## Initializing the Client Window

The structure LISTBOXINFO is used to hold the list box information.

```
typedef struct
{
      USHORT ausListBoxSel[NUM_ENTRIES];
} LISTBOXINFO,*PLISTBOXINFO;
```

The array *ausListBoxSel[]* is used to hold the items that have been selected.

The WM_CREATE message processing is where the memory is allocated for the structure LISTBOXINFO. *WinSetWindowPtr* is used to assign the pointer to the structure *pliInfo* to the window word.

## Initializing the List Box

```
hwndListBox = WinWindowFromID(hwndDlg,
                              IDL_LISTBOX);

for (i = 0; i < NUM_ENTRIES; i++)
             WinInsertLboxItem(hwndListBox,
                               LIT_END,
                               pszListBoxEntry[i]);
```

The WM_INITDLG message processing initializes the list box. The first step is to obtain the window handle of the list box using *WinWindowFromID*. The dialog box is the parent of all the controls in it. The macro *WinInsertLboxItem* is a shortened version of the function *WinSendDlgItemMsg*, designed specially to insert items into the list box. The first parameter is the list box window handle, *hwndListBox*. The second parameter indicates the position in the list box to insert the item. Acceptable entries are either an integer value indicating the placement of the item (0 indicates the topmost item) or the constant LIT_END. Also, the list box control is smart enough to sort the items alphabetically. The constants LIT_SORTASCENDING and LIT_SORTDESCENDING can be used to automate this process. Alphabetization takes some time, though; sorting the list box items before inserting them in the list box may increase performance. The last parameter is the text string to enter into the list box. The header file LISTBOX.H contains the definition for *pszListBoxEntry*.

```
WinSendDlgItemMsg ( hwndDlg,
                    IDL_LISTBOX,
                    LM_SELECTITEM,
                    MPFROMSHORT ( 0 ) ,
                    MPFROMSHORT ( TRUE )) ;
```

One other nit about the list box: The first item must be selected manually. The message LM_SELECTITEM will do this for us. The first parameter is the index of the list box item to be selected. The second parameter indicates whether the item is selected (TRUE) or deselected (FALSE). Notice that this time we use the function *WinSendDlgItemMsg*; this is another way to send messages to items in a dialog box.

## The WM_COMMAND Message Dialog Processing

```
hwndClient = WinQueryWindow(hwndDlg,
                            QW_OWNER);

WinPostMsg(hwndClient,
           UM_LISTBOXSEL,
           MPVOID,
           MPVOID);

return (MRFROMSHORT(TRUE));
```

When the user presses either the OK or the CANCEL button, the system sends a WM_COMMAND message to the dialog box. *mpParm1* contains the ID of the pushbutton, either DID_OK or DID_CANCEL. If the user presses DID_OK, the system sends a user-defined message, UM_LISTBOXSEL, to the client window and returns TRUE. This prevents the system from dismissing the dialog box.

If the user presses the CANCEL button, the dialog box is destroyed, using *WinDismissDlg*. Also, a UM_LISTBOXSEL message is sent to reset the LISTBOXINFO structure and repaint the client window area.

## Processing the UM_LISTBOXSEL Message

```
SHORT           sSelect = 0;
SHORT           sIndex = LIT_FIRST;
HWND            hwndDlg;
USHORT          i;

for (i = 0; i < NUM_ENTRIES; i++)
   pliInfo->ausListBoxSel[i] = FALSE;

hwndDlg = WinWindowFromID(HWND_DESKTOP,
                          IDD_LISTBOX);

while (sSelect != LIT_NONE && hwndDlg)
{

   sSelect = (SHORT)WinSendDlgItemMsg(hwndDlg,
                                      IDL_LISTBOX,
                                      LM_QUERYSELECTION,
                                      MPFROMSHORT( sIndex ),
                                      MPVOID);

   pliInfo->ausListBoxSel[sSelect] = TRUE;

   sIndex = sSelect;
}
pliInfo->fSelectedItems = TRUE;
```

When the client window receives the UM_LISTBOXSEL message, it is the client's job to find the selected list box items. Our list box has the style LS_MULTIPLESEL, so the user can select as many items as he or she wants. Because so many items can be selected, the procedure to find all of them can be a little tricky; not difficult, just tricky. The message LM_QUERYSELECTION starts at the list box item specified in *mpParml* and returns the first selected item it finds. This is a fairly simple procedure to code. A *while* loop continues searching until *sSelect* equals LIT_NONE (in other words, no more items are selected). We next send a LM_QUERYSELECTION message to the list box window, with the variable *sIndex* indicating the index of the item at which to start the search. At the start of the loop, this variable is LIT_FIRST, the first item in the list box. When the first selected item is found, the variable *sSelect* contains the index of the item. As the loop traverses through the items in the list box, the starting search point is updated to *sSelect*. As a selected item is found, the corresponding index in the array *ausListBoxSel[]* is set to TRUE. This information is used in the WM_PAINT processing.

The last part of this message processing is to signal the client window to repaint itself.

## The Client Window Painting Routine

The WM_PAINT processing is where the items selected in the list box actually are written to the client area window. *WinFillRect* fills the drawing region with the color CLR_WHITE.

```
WinQueryWindowRect(hwndClient,
                   &rclPaintRegion);

rclNewPaint.xLeft = (rclPaintRegion.xRight-
   rclPaintRegion.xLeft) / 4 * 3;
rclNewPaint.xRight = rclPaintRegion.xRight;

rclNewPaint.yBottom = rclPaintRegion.yBottom;
rclNewPaint.yTop = rclPaintRegion.yTop;

WinFillRect(hpsPresentationSpace,
            &rectInvalidRect,
            CLR_WHITE);
```

If the user has selected some items, *WinDrawText* is used to write a heading on the client area. The array *ausListBoxSel[]* is cycled through to find each selected item and write the list box item text to the client area as well.

## Owner-Drawing Controls

An owner-draw style can be used for many of the Presentation Manager controls. This style sends a WM_DRAWITEM message when some portion of the control is to be drawn. This feature lets the programmer customize the appearance of the control.

The LISTBOX example program creates an owner-drawn list box that has system bitmaps and their titles as the selectable items.

**LISTBOX.C**

```
#define  INCL_WIN
#define  INCL_GPI
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "listbox.h"

typedef struct _BITMAPDATA
{
   CHAR              achName[20];
   USHORT            usNumber;
} BITMAPDATA,*PBITMAPDATA;

#define  MAX_BITMAPS   9
#define  CLS_CLIENT    "MyClass"

BITMAPDATA        abdBitmaps[MAX_BITMAPS] =
{

   "SBMP_CHILDSYSMENU",
   SBMP_CHILDSYSMENU,
   "SBMP_MAXBUTTON",
   SBMP_MAXBUTTON,
   "SBMP_MENUATTACHED",
   SBMP_MENUATTACHED,
   "SBMP_MINBUTTON",
   SBMP_MINBUTTON,
   "SBMP_PROGRAM",
   SBMP_PROGRAM,
   "SBMP_RESTOREBUTTON",
   SBMP_RESTOREBUTTON,
   "SBMP_SIZEBOX",
   SBMP_SIZEBOX,
   "SBMP_SYSMENU",
   SBMP_SYSMENU,
   "SBMP_TREEMINUS",
   SBMP_TREEMINUS
} ;

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                                  mpParm1,MPARAM mpParm2);

BOOL Draw1Bitmap(HPS hpsDraw,HBITMAP hbmBitmap,PRECTL prclDest);

MRESULT EXPENTRY DlgProc(HWND hwndWnd,ULONG ulMsg,MPARAM mpParm1,
                           MPARAM mpParm2);
```

```
INT main(VOID)
{
   HMQ               hmqQueue;
   HAB               habAnchor;
   ULONG             ulFlags;
   HWND              hwndFrame;
   HWND              hwndClient;
   BOOL              bLoop;
   QMSG              qmMsg;
   LONG              lWidth,lHeight;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW|CS_SYNCPAINT,
                    0);

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_MENU|FCF_SIZEBORDER|
      FCF_MINMAX;

   /************************************************************/
   /* create frame and client window                           */
   /************************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Listbox Example",
                                  0,
                                  NULLHANDLE,
                                  ID_RESOURCE,
                                  &hwndClient);

   /************************************************************/
   /* get screen height and width                              */
   /************************************************************/

   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /************************************************************/
   /* set size and position of frame window                    */
   /************************************************************/

   if (hwndFrame != NULLHANDLE)
   {
      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      10,
                      10,
                      lWidth/10*8,
                      lHeight/10*8,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
```

```
        WinDispatchMsg(habAnchor,
                       &qmMsg);
        bLoop = WinGetMsg(habAnchor,
                          &qmMsg,
                          NULLHANDLE,
                          0,
                          0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)

{
   switch (ulMsg)
   {

      case  WM_COMMAND :
         switch (SHORT1FROMMP(mpParm1))
         {
            case  IDM_DISPLAY :

               /*********************************************/
               /* load up dialog                            */
               /*********************************************/

               WinDlgBox(HWND_DESKTOP,
                         hwndClient,
                         DlgProc,
                         NULLHANDLE,
                         IDD_LISTBOX,
                         NULL);
               break;
            case  IDM_EXIT :

               /*********************************************/
               /* close up window                           */
               /*********************************************/

               WinPostMsg(hwndClient,
                          WM_CLOSE,
                          MPVOID,
                          MPVOID);
               break;
            default  :
               return  WinDefWindowProc(hwndClient,
                                        ulMsg,
                                        mpParm1,
                                        mpParm2);
         }
         break;
      case  WM_PAINT :
         {
            HPS              hpsPaint;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     NULL);

            /*********************************************/
            /* erase the invalidated region              */
            /*********************************************/
```

```
            GpiErase(hpsPaint);
            WinEndPaint(hpsPaint);
         }
         break;

      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,

                                  mpParm2);
   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)

{
   switch (ulMsg)
   {

      case  WM_INITDLG :
         {
            USHORT           usIndex;

            /************************************************/
            /* insert dummy items into listbox as placeholders */
            /************************************************/

            for (usIndex = 0; usIndex < MAX_BITMAPS; usIndex++)
            {
               WinSendDlgItemMsg(hwndDlg,
                                 IDL_LISTBOX,
                                 LM_INSERTITEM,
                                 MPFROMSHORT(usIndex),
                                 MPFROMP(""));
            }                          /* endfor               */

            /************************************************/
            /* select the first listbox item                */
            /************************************************/

            WinSendDlgItemMsg(hwndDlg,
                              IDL_LISTBOX,
                              LM_SELECTITEM,
                              MPFROMSHORT(0),
                              MPFROMSHORT(TRUE));
         }
         break;

      case  WM_COMMAND :
         switch (SHORT1FROMMP(mpParm1))
         {
            case  DID_OK :
            case  DID_CANCEL :
               WinDismissDlg(hwndDlg,
                             FALSE);
               break;

            default  :
               return  WinDefDlgProc(hwndDlg,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
         }                             /* endswitch            */
         break;
```

```
case  WM_MEASUREITEM :
   {
      HPS               hpsChar;
      FONTMETRICS       fmMetrics;
      LONG              lMaxCy;
      USHORT            usIndex;
      HBITMAP           hbmBitmap;
      BITMAPINFOHEADER2 bmihHeader;

      /*************************************************/
      /* get font size                                 */
      /*************************************************/

      hpsChar = WinGetPS(hwndDlg);
      GpiQueryFontMetrics(hpsChar,
                          (LONG)sizeof(fmMetrics),
                          &fmMetrics);
      WinReleasePS(hpsChar);

      lMaxCy = fmMetrics.lMaxBaselineExt;

      /*************************************************/
      /* get size of bitmaps                           */
      /*************************************************/

      for (usIndex = 0; usIndex < MAX_BITMAPS; usIndex++)
      {
         hbmBitmap = WinGetSysBitmap(HWND_DESKTOP,
                                     abdBitmaps[usIndex].
                                        usNumber);

         bmihHeader.cbFix = 16;
         GpiQueryBitmapInfoHeader(hbmBitmap,
                                  &bmihHeader);

         /*************************************************/
         /* which is larger, previous max or bitmap       */
         /*************************************************/

         lMaxCy = max(lMaxCy,
                      bmihHeader.cy);

         /*************************************************/
         /* free the bitmap                               */
         /*************************************************/

         GpiDeleteBitmap(hbmBitmap);
      }                             /* endfor              */
      return  MRFROMLONG(lMaxCy+10);
   }

case  WM_DRAWITEM :
   {
      POWNERITEM        poiItem;
      HBITMAP           hbmBitmap;
      RECTL             rclText;

      poiItem = (POWNERITEM)PVOIDFROMMP(mpParm2);

      rclText = poiItem->rclItem;
      rclText.xLeft = (rclText.xRight-rclText.xLeft)/7;

      /*************************************************/
      /* draw the bitmap name                          */
      /*************************************************/
```

```
            WinDrawText(poiItem->hps,
                        -1,
                        abdBitmaps[poiItem->idItem].achName,
                        &rclText,
                        poiItem->fsState?CLR_YELLOW:CLR_BLUE,
                        poiItem->fsState?CLR_BLUE:CLR_WHITE,
                        DT_LEFT|DT_VCENTER|DT_ERASERECT);

            rclText = poiItem->rclItem;
            rclText.xRight = (rclText.xRight-rclText.xLeft)/7;

            /*****************************************************/
            /* fill the rectangle with white                     */
            /*****************************************************/

            WinFillRect(poiItem->hps,
                        &rclText,
                        CLR_WHITE);

            hbmBitmap = WinGetSysBitmap(HWND_DESKTOP,
                                        abdBitmaps[poiItem->
                                           idItem].usNumber);

            /*****************************************************/
            /* draw the bitmap, then delete                      */
            /*****************************************************/

            Draw1Bitmap(poiItem->hps,
                        hbmBitmap,
                        &rclText);

            GpiDeleteBitmap(hbmBitmap);

            /*****************************************************/
            /* set both states to FALSE and return TRUE          */
            /*****************************************************/

            poiItem->fsState = FALSE;
            poiItem->fsStateOld = FALSE;

            return  MRFROMSHORT(TRUE);
         }

      default  :
         return  WinDefDlgProc(hwndDlg,
                               ulMsg,
                               mpParm1,
                               mpParm2);
   }                                  /* endswitch              */
   return  MRFROMSHORT(FALSE);
}

BOOL Draw1Bitmap(HPS hpsDraw,HBITMAP hbmBitmap,PRECTL prclDest)
{
   BITMAPINFOHEADER2 bmihHeader;
   POINTL            ptlPoint;
   BOOL              bRc;

   bmihHeader.cbFix = 16;
   GpiQueryBitmapInfoHeader(hbmBitmap,
                            &bmihHeader);

   /*********************************************************/
   /* set the x and y coordinates                           */
   /*********************************************************/

   ptlPoint.x = (prclDest->xRight-prclDest->xLeft-bmihHeader.cx)/
      2+prclDest->xLeft;
```

```
   ptlPoint.y = (prclDest->yTop-prclDest->yBottom-bmihHeader.cy)/
      2+prclDest->yBottom;

   /*************************************************************/
   /* draw the bitmap                                           */
   /*************************************************************/

   bRc = WinDrawBitmap(hpsDraw,
                       hbmBitmap,
                       NULL,
                       &ptlPoint,
                       0,
                       0,
                       DBM_NORMAL|DBM_IMAGEATTRS);
   return  bRc;
}
```

## LISTBOX.RC

```
#include <os2.h>
#include "listbox.h"

MENU ID_RESOURCE
{
   SUBMENU "~File", -1
   {
      MENUITEM "~Display Dialog", IDM_DISPLAY
      MENUITEM SEPARATOR
      MENUITEM "E~xit", IDM_EXIT
   }
}


DLGTEMPLATE IDD_LISTBOX LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG  "Listbox Example", IDD_LISTBOX, 12, 13, 197, 146, WS_VISIBLE,
            FCF_SYSMENU | FCF_TITLEBAR
    BEGIN
        LISTBOX         IDL_LISTBOX, 14, 18, 152, 121, LS_OWNERDRAW
        DEFPUSHBUTTON   "OK", DID_OK, 3, 3, 40, 13, WS_GROUP
        PUSHBUTTON      "Cancel", DID_CANCEL, 48, 3, 40, 13
    END
END
```

## LISTBOX.H

```
#define IDD_LISTBOX                 256
#define IDL_LISTBOX                 512
#define ID_RESOURCE                 100
#define IDM_DISPLAY                 101
#define IDM_EXIT                    102
```

**LISTBOX.MAK**

```
LISTBOX.EXE:                    LISTBOX.OBJ \
                                LISTBOX.RES
        LINK386 @<<
LISTBOX
LISTBOX
LISTBOX
OS2386
LISTBOX
<<
        RC LISTBOX.RES LISTBOX.EXE

LISTBOX.RES:                    LISTBOX.RC \
                                LISTBOX.H
        RC -r LISTBOX.RC LISTBOX.RES

LISTBOX.OBJ:                    LISTBOX.C \
                                LISTBOX.H
        ICC -C+ -Kb+ -Ss+ LISTBOX.C
```

**LISTBOX.DEF**

```
NAME LISTBOX WINDOWAPI

DESCRIPTION 'Second listbox example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

The beginning of the program should look familiar. The structure BITMAPDATA is defined:

```
typedef struct _BITMAPDATA {
   CHAR   achName [20] ;
   USHORT usNumber ;
} BITMAPDATA, * PBITMAPDATA ;
```

The first field, *achName*, is the #define'd text string of each system bitmap. The second field, *usNumber*, is the number of the system bitmap. When we draw the bitmaps, we'll use this structure to access the bitmaps we want.

# DlgProc

```
for (usIndex = 0; usIndex < MAX_BITMAPS; usIndex++)
{
   WinSendDlgItemMsg(hwndDlg,
                     IDL_LISTBOX,
                     LM_INSERTITEM,
                     MPFROMSHORT(usIndex),
                     MPFROMP(""));
}                          /* endfor                */
```

The WM_INITDLG message is where the initialization of the dialog box and all its components takes place. In this case, we want to initialize the list box. *WinSendDlgItemMsg* can be used to communicate directly with it. The message LM_INSERTITEM is used to insert items into the list box. If this was not an owner-drawn list box, the actual text strings would be inserted here; however, because this is an owner-drawn list box, it is important to tell the list box there will be eight items. The message LM_SELECTITEM is used to set the first item to the selected state.

## The WM_MEASUREITEM Message

```
for (usIndex = 0; usIndex < MAX_BITMAPS; usIndex++)
{
hbmBitmap = WinGetSysBitmap(HWND_DESKTOP,
                            abdBitmaps[usIndex].
                            usNumber);
   bmihHeader.cbFix = 16;
   GpiQueryBitmapInfoHeader(hbmBitmap,
                            &bmihHeader);
   lMaxCy = max(lMaxCy,
                bmihHeader.cy);
   GpiDeleteBitmap(hbmBitmap);
}                          /* endfor                */
return  MRFROMLONG(lMaxCy+10);
```

The WM_MEASUREITEM message must be processed for an owner-drawn list box and also for horizontal scrolling list boxes. This message tells the list box how tall or, in some cases, how wide each list box item is to be. The tallest, or widest, size should be returned in order for all the list box items to have a consistent look. In our example, all items are the same size. *GpiQueryFontMetrics* is used to get all sorts of information about the selected font. The one piece of the FONTMETRICS structure we are interested in is *fm.Metrics.lMaxBaselineExt.* This indicates the maximum height of the font. This is compared to the maximum height of the system bitmap. This information is contained in the BITMAPINFOHEADER structure that is obtained using *GpiQueryBitmapInfoHeader.* After the comparison, we free the bitmap handle with *GpiDeleteBitmap*

## The WM_DRAWITEM Message

The WM_DRAWITEM is the most complicated message processing in this example. This message is sent to the owner that will be doing the drawing whenever an item needs to be selected, unselected, or drawn. The second parameter in the WM_DRAWITEM message is a pointer to an OWNERITEM structure, which looks like this:

```
  typedef struct _OWNERITEM
{
     HWND    hwnd;
     HPS     hps;
     ULONG   fsState;
     ULONG   fsAttribute;
     ULONG   fsStateOld;
     ULONG   fsAttributeOld;
     RECTL   rclItem;
     LONG    idItem;
     ULONG   hItem;
  } OWNERITEM;
  typedef OWNERITEM *POWNERITEM;
```

This structure has pretty much everything you need to draw a list box item.

## An Introduction to Owner-drawn States

The OWNERITEM structure contains the variables *fsState* and *fsStateOld.* The state variables indicate whether an item needs selection highlighting. When an item's selection highlighting is changing, the item needs to be redrawn, and the *fsState* field will be set differently from the *fsStateOld* field. A state of TRUE indicates the item is selected; FALSE indicates an unselected item. Programmers can draw the highlighting themselves or let the system handle the highlighting and unhighlighting. The flowchart

depicted in Figure 15.2 lists the possible combination of states and returns and the action by both the program and the system.

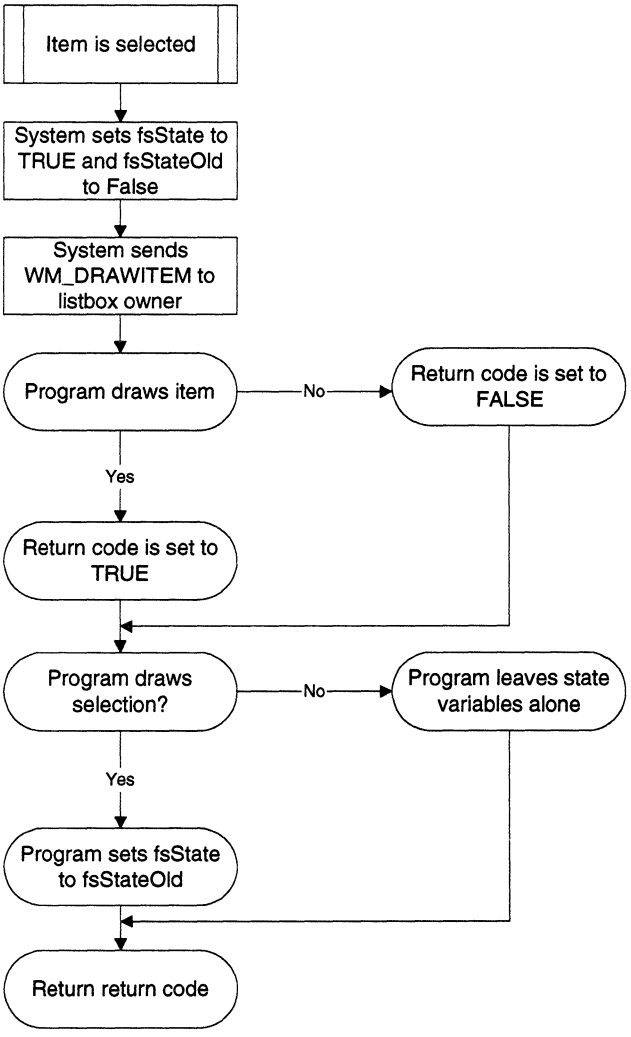



Figure 15.2 Flowchart of owner-drawn selection.

The system sets these variables before the WM_DRAWITEM message is sent; it looks at what is returned in them after the WM_DRAWITEM message has been processed to determine whether to handle the highlighting of the item. If *fsState* is equal to *fsStateOld*, the system will do no highlighting. If the variables are not equal to each other, the system will highlight them or unhighlight them by inverting the item rectangle.

## Drawing the List Box Labels

```
poiItem = (POWNERITEM)PVOIDFROMMP(mpParm2);
rclText = poiItem->rclItem;
rclText.xLeft = (rclText.xRight-rclText.xLeft)/7;
WinDrawText(poiItem->hps,
            -1,
            abdBitmaps[poiItem->idItem].achName,
            &rclText,
            poiItem->fsState?CLR_YELLOW:CLR_BLUE,
            poiItem->fsState?CLR_BLUE:CLR_WHITE,
            DT_LEFT|DT_VCENTER|DT_ERASERECT);
```

A pointer to the OWNERITEM structure is contained in *mpParm2*. The *rclItem* field is the RECTL structure of the specific list box item that needs to be drawn. We indent the text one-seventh of the way across and use the function *WinDrawText* to write the bitmap name. Notice the use of the flag DT_ERASERECT in the last parameter. This flag erases the drawing area before Presentation Manager draws the text.

## Drawing the Bitmaps

```
rclText = poiItem->rclItem;
rclText.xRight = (rclText.xRight-rclText.xLeft)/7;

WinFillRect(poiItem->hps,
            &rclText,
            CLR_WHITE);
hbmBitmap = WinGetSysBitmap(HWND_DESKTOP,
                            abdBitmaps[poiItem->
                            idItem].usNumber);

Draw1Bitmap(poiItem->hps,
            hbmBitmap,
            &rclText);

GpiDeleteBitmap(hbmBitmap);
```

The next thing to do is get a handle to the bitmap we want to draw in our list box item. *WinGetSysBitmap* is used to do this. The first parameter is the desktop window handle, HWND_DESKTOP. The second parameter is the system bitmap number. *poiItem->idItem* is the index of the selected item. We use this index as the index into the *abdBitmaps* structure. *Draw1Bitmap* is a very simple user-defined function we use to actually draw the bitmap. Once the bitmap has been drawn, some cleanup will be necessary. The handle of the bitmap needs to be freed using *GpiDeleteBitmap*.

```
poiItem -> fsState = FALSE ;
poiItem -> fsStateOld = FALSE ;
return MRFROMSHORT ( TRUE ) ;
```

The last step in our message processing is to set all the appropriate variables correctly for the window procedure. We set *fsState* and *fsStateOld* to FALSE to tell the system we already have done the highlighting. A return code of TRUE indicates that the item has been drawn already, so please do not draw it again. If FALSE had been returned here, the text " ", the string that was used in the LM_INSERTITEM message, would be placed over all the wonderful work we've done so far.

For more information on drawing bitmaps, see Chapter 12.

## Summary

A list box is a very simple control to use, yet it provides a powerful level of functionality. This chapter has introduced the concepts of a regular list box and an owner-drawn list box. Developers interested in creating their own, even more advanced list box, should refer to the series of articles by Mark Benge and Matt Smith starting in the January/February 1994 *OS/2 Developer* magazine.

# Chapter 16

# Buttons

The easiest controls to use are buttons. Buttons belong to the class WC_BUTTON. There are five types of buttons—push buttons, radio buttons, three-state buttons, check boxes, and owner-drawn.

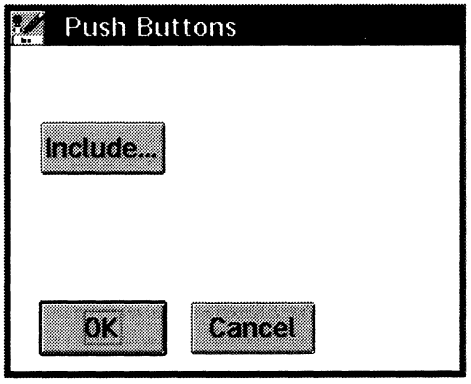



Figure 16.1 Push buttons.

A push button (see Figure 16.1) sends a WM_COMMAND to its owner immediately when it is pressed. This feature distinguishes the push button from the other button types. Push buttons commonly are used to initiate such actions as "OK," "Cancel," and "Help."

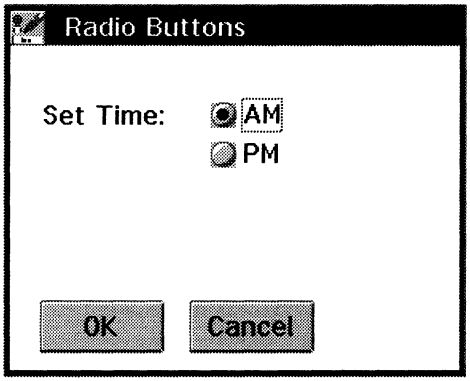



Figure 16.2 Radio buttons.

Radio buttons (see Figure 16.2) are designed to be used when only one item in a group can be selected. For instance, indicating "AM" or "PM" as a period of time is an example of where radio buttons should be used. There are two styles of radio buttons: BS_AUTORADIOBUTTON and BS_RADIOBUTTON. When using the BS_RADIOBUTTON, the application must highlight the selected button and unhighlight the button previously selected. The system handles this automatically when the BS_AUTORADIOBUTTON is used. When radio buttons are used, the application can send a BM_QUERYCHECKINDEX message to determine which button was selected when the user exited the dialog box.

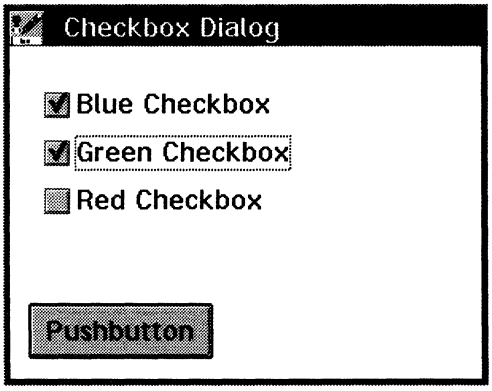



Figure 16.3 Check boxes.

In cases where more than one choice can be selected, check boxes (see Figure 16.3) should be used. Two styles define check boxes: BS_CHECKBOX or BS_AUTOCHECKBOX. The difference between the two styles is similar in manner to their radio button counterparts, BS_RADIOBUTTON and BS_AUTORADIOBUTTON.

## Button Styles

The styles listed in Table 16.1 can be used when creating buttons.

Table 16.1 Button Styles

| Style | Description |
|---|---|
| BS_3STATE | Creates a three-state check box that can be selected, unselected, or disabled. |
| BS_AUTO3STATE | Creates a three-state check box whose state is set by the system automatically. |
| BS_AUTOCHECKBOX | Creates a check box that the system will toggle automatically between selected and unselected. |
| BS_AUTORADIOBUTTON | Creates a radio button that will disable other radio buttons in the group automatically when it is selected. |
| BS_AUTOSIZE | Will size the push button to fit the text label, if -1 is specified as width and height. |
| BS_BITMAP | Creates a push button, labeled with a bitmap instead of text. |
| BS_CHECKBOX | Creates a check box; it is the application's responsibility to select or deselect the check box. |
| BS_DEFAULT | Creates a button with thick border boxes; used with BS_PUSHBUTTON or BS_USERBUTTON. |
| BS_ICON | Creates a push button, labeled with an icon instead of text. |
| BS_HELP | Creates a push button that sends a WM_HELP message to the owner window; this can be used only with push buttons. |
| BS_MINIICON | Creates a icon push button with a 16x16 icon. |
| BS_NOCURSORSELECT | Creates an auto-radio button that is not selected automatically when the button is moved to with the cursor keys. |
| BS_NOBORDER | Creates a push button with no border; can be used only with push buttons. |
| BS_NOPOINTERFOCUS | Creates a radio button or check box that does not receive the keyboard focus when the user selects it. |
| BS_PUSHBUTTON | Creates a push button. |
| BS_RADIOBUTTON | Creates a radio button. |
| BS_SYSCOMMAND | Creates a button that posts a WM_SYSCOMMAND when selected; can be used only with push buttons. |
| BS_USERBUTTON | Creates a user-defined button; generates a BN_PAINT notification message, sent to its owner, when painting is needed. |

# Example Program

The following program will create a simple dialog box that contains various types of buttons. Buttons are created both in the resource file and by using *WinCreateWindow*.

**BUTTON.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include "button.h"
#define  CLS_CLIENT     "MyClass"
MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

MRESULT EXPENTRY DlgProc(HWND hwndWnd,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);

INT main(VOID)
```

```
{
   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   HWND              hwndClient;
   BOOL              bLoop;
   QMSG              qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW,
                    0);

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_MENU|FCF_SIZEBORDER;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Button Control Example",
                                  0,
                                  NULLHANDLE,
                                  IDR_CLIENT,
                                  &hwndClient);

   if (hwndFrame != NULLHANDLE)
   {

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      50,
                      50,
                      550,
                      380,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {
```

```
      case  WM_ERASEBACKGROUND :
         return  MRFROMSHORT(TRUE);

      case  WM_COMMAND :

         switch (SHORT1FROMMP(mpParm1))
         {

            case  IDM_START :

               /*********************************************/
               /* load and run the dialog box               */
               /*********************************************/

               WinDlgBox(HWND_DESKTOP,
                         hwndWnd,
                         DlgProc,
                         NULLHANDLE,
                         IDD_BUTTON,
                         NULL);
               break;

            case  IDM_EXIT :
               WinPostMsg(hwndWnd,
                          WM_CLOSE,
                          0,
                          0);
               break;

            default  :
               return  WinDefWindowProc(hwndWnd,
                                        ulMsg,
                                        mpParm1,

                                        mpParm2);
         }
         break;
      default  :
         return  WinDefWindowProc(hwndWnd,
                                  ulMsg,
                                  mpParm1,

                                  mpParm2);
   }                                   /* end switch ulMsg     */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY DlgProc(HWND hwndWnd,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)
{
   switch (ulMsg)
   {

      case  WM_INITDLG :
         {
            BTNCDATA         bcdData;
            POINTL           ptl;

            bcdData.cb = sizeof(BTNCDATA);
            bcdData.fsCheckState = 0;
            bcdData.fsHiliteState = 0;

            /*********************************************/
            /* load the information pointer              */
            /*********************************************/
```

```
        bcdData.hImage = WinQuerySysPointer(HWND_DESKTOP,
                                            SPTR_ICONINFORMATION
                                                ,
                                              FALSE);

        /************************************************/
        /* these are the coordinates we want in dialog units*/
        /************************************************/

        ptl.x = 175;
        ptl.y = 25;

        /************************************************/
        /* map out to correct window coordinates        */
        /************************************************/

        WinMapDlgPoints(hwndWnd,
                        &ptl,
                        1,
                        TRUE);

        /************************************************/
        /* create an icon button                        */
        /************************************************/

        WinCreateWindow(hwndWnd,
                        WC_BUTTON,
                        "",
                        WS_VISIBLE|WS_TABSTOP|BS_ICON,
                        ptl.x,
                        ptl.y,
                        WinQuerySysValue(HWND_DESKTOP,
                                         SV_CXICON),
                        WinQuerySysValue(HWND_DESKTOP,
                                         SV_CYICON),
                        hwndWnd,
                        HWND_TOP,
                        IDR_ICON,
                        (PVOID)&bcdData,
                        NULL);

        /************************************************/
        /* check some of the buttons                    */
        /************************************************/

        WinSendDlgItemMsg(hwndWnd,
                          IDC_AUTOCHECKBOX,
                          BM_SETCHECK,
                          MPFROMSHORT(TRUE),
                          NULL);

        WinSendDlgItemMsg(hwndWnd,
                          IDR_AUTORADIOBUTTON,
                          BM_SETCHECK,
                          MPFROMSHORT(TRUE),
                          NULL);
```

```
            WinSendDlgItemMsg(hwndWnd,
                              IDC_AUTO3STATE,
                              BM_SETCHECK,
                              MPFROMSHORT(2),
                              NULL);
         }
         break;
      case  WM_COMMAND :
         switch (SHORT1FROMMP(mpParm1))
         {
            case  DID_OK :
               WinDismissDlg(hwndWnd,
                             LONGFROMMP(mpParm1));
               break;

               /*************************************************/
               /* for the next two items, do not dismiss       */
               /* dialog, instead return FALSE to keep dialog  */
               /* displaying                                    */
               /*************************************************/

            case  IDP_NOBORDER :
            case  IDR_ICON :
               break;
            default  :
               return  WinDefDlgProc(hwndWnd,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
         }                             /* end switch           */
         break;
      default  :
         return  WinDefDlgProc(hwndWnd,
                               ulMsg,
                               mpParm1,

                               mpParm2);
   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}
```

## BUTTON.RC

```
#include <os2.h>
#include "button.h"

MENU IDR_CLIENT
{
   SUBMENU "~File", IDM_FILES
   {
      MENUITEM "~Display dialog...", IDM_START
      MENUITEM "E~xit", IDM_EXIT
   }
}


DLGTEMPLATE IDD_BUTTON PRELOAD MOVEABLE DISCARDABLE
{
   DIALOG "Button dialog", IDD_BUTTON, 28, 23, 258, 110,
          FS_NOBYTEALIGN | WS_VISIBLE,
          FCF_SYSMENU | FCF_TITLEBAR
          PRESPARAMS PP_BACKGROUNDCOLORINDEX, CLR_WHITE
   {
      LTEXT "Radio Button Styles", -1, 135, 95, 86, 8
      LTEXT "Check Box Styles", -1, 11, 95, 76, 8
      LTEXT "Push Button Styles", -1, 55, 48, 82, 8
```

```
      AUTOCHECKBOX "BS_AUTOCHECKBOX", IDC_AUTOCHECKBOX,
         11, 81, 106, 10
      AUTOCHECKBOX "BS_AUTO3STATE", IDC_AUTO3STATE,
         11, 66, 89, 10, BS_AUTO3STATE | WS_GROUP
      AUTORADIOBUTTON "BS_AUTORADIOBUTTON", IDR_AUTORADIOBUTTON,
         131, 81, 121, 10
      AUTORADIOBUTTON "BS_NOPOINTERFOCUS", IDR_NOPOINTER,
         131, 66, 116, 10, BS_NOPOINTERFOCUS
      PUSHBUTTON "BS_HELP", IDP_HELP,
         20, 25, 53, 14, BS_HELP
      PUSHBUTTON "BS_NOBORDER", IDP_NOBORDER,
         83, 25, 82, 14, BS_NOBORDER
      DEFPUSHBUTTON "OK", DID_OK, 10, 5, 40, 12 | WS_GROUP
   }
}
```

## BUTTON.H

```
#define IDR_CLIENT                256
#define IDD_BUTTON                257

#define IDM_FILES                 320
#define IDM_START                 321
#define IDM_EXIT                  322

#define IDC_AUTOCHECKBOX          512
#define IDC_3STATE                513
#define IDC_AUTO3STATE            514
#define IDR_AUTORADIOBUTTON       515
#define IDR_NOPOINTER             516
#define IDR_AUTOSIZE              517
#define IDP_HELP                  518
#define IDP_NOBORDER              519

#define IDR_ICON                  1024
```

## BUTTON.MAK

```
BUTTON.EXE:                       BUTTON.OBJ \
                                  BUTTON.RES
        LINK386 @<<
BUTTON
BUTTON
BUTTON
OS2386
BUTTON
<<
        RC BUTTON.RES BUTTON.EXE

BUTTON.RES:                       BUTTON.RC \
                                  BUTTON.H
        RC -r BUTTON.RC BUTTON.RES

BUTTON.OBJ:                       BUTTON.C \
                                  BUTTON.H
        ICC -C+ -Kb+ -Ss+ BUTTON.C
```

## BUTTON.DEF

```
NAME BUTTON WINDOWAPI
DESCRIPTION 'Button example.
             Copyright (c) 1995 by Kathleen Panov.
             All rights reserved.'

STACKSIZE    32768
```

# The BUTTON.RC Resource File

The following is the code used to define the dialog box. The background color is set to white using the PRESPARAMS keyword in the BUTTON.RC file.

```
DIALOG "Button dialog", IDD_BUTTON, 28, 23, 258, 110,
            FS_NOBYTEALIGN | WS_VISIBLE,
            FCF_SYSMENU | FCF_TITLEBAR
            PRESPARAMS PP_BACKGROUNDCOLORINDEX, CLR_WHITE
```

The creation of the buttons is specified by the keywords PUSHBUTTON, AUTOCHECKBOX, and AUTORADIOBUTTON in the BUTTON.RC resource file.

## *DlgProc*

```
BTNCDATA bcdData ;

bcdData.cb = sizeof ( BTNCDATA ) ;
bcdData.fsCheckState = 0 ;
bcdData.fsHiliteState = 0 ;
bcdData.hImage = WinQuerySysPointer (
                        HWND_DESKTOP,
                        SPTR_ICONINFORMATION,
                                FALSE ) ;
```

The WM_INITDLG message processing is used to create the BS_ICON push button. The information icon is loaded from the system using *WinQuerySysPointer*. This returns a resource handle (HPOINTER) that is needed in the BTNCDATA structure. The BTNCDATA structure is defined as follows.

```
struct {
        USHORT cb;
        USHORT fsCheckState;
        USHORT fsHiliteState;
        LHANDLE hImage;
} BTNCDATA;
```

**Gotcha!**

Programmers must not forget to initialize everything in the BTNCDATA structure. If they don't, they will receive an error. *cb* is always the size of the BTNCDATA structure. *fsCheckState* indicates whether the initial state of a button is checked or unchecked. *fsHiliteState* is used to set the highlight or unhighlight state of the button. The last field, *hImage*, is a handle for a pointer or a bitmap.

# Dialog Units—Can We Talk?

```
ptl.x = 175;
ptl.y = 25;

WinMapDlgPoints(hwndWnd,
                &ptl,
                1,
                TRUE);
```

In this example, we mix the create buttons in the resource file and also dynamically in the C code. There is a difference between the coordinates specified in the resource file and those specified in the C file. The

resource file uses a coordinate system known as *dialog units*. These units are based on the size of the system font and are different from the pixel units that a window coordinate system uses. In order to place the new push button in the right position, we must first map the dialog units to a window coordinate system. The dialog coordinates are placed into a POINTL structure, which consists solely of as x and y elements. The function *WinMapDlgPoints* is explained in Chapter 12.

```
WinCreateWindow(hwndWnd,
                WC_BUTTON,
                "",
                WS_VISIBLE|WS_TABSTOP|BS_ICON,
                ptl.x,
                ptl.y,
                WinQuerySysValue(HWND_DESKTOP,
                                 SV_CXICON),
                WinQuerySysValue(HWND_DESKTOP,
                                 SV_CYICON),
                hwndWnd,
                HWND_TOP,
                IDR_ICON,
                (PVOID)&bcdData,
                NULL);
```

*WinCreateWindow* is used to create the icon push button. The client area of the dialog is used as both the parent and the owner. The text area is specified as "". The styles specified for the button are WS_VISIBLE | WS_TABSTOP | BS_ICON. WS_TABSTOP indicates that the user can press the TAB key to move to the button. On some button styles this is the default and does not have to be specified. This style is associated with push buttons and check boxes automatically. Icon buttons and radio buttons do not.

The placement of the push button is specified at *ptl.x* and *ptl.y*, and the width and height are set at the system values for the icon width (SV_CXICON) and icon height (SV_CYICON), respectively. The dialog window *hwndWnd* will be the owner. HWND_TOP is used to indicate window placement, and IDR_ICON is the ID of the button. The next parameter is the address of the button control data, which is *&ButtonData* in this example.

```
WinSendDlgItemMsg ( hwndWnd,
                    IDC_AUTOCHECKBOX,
                    BM_SETCHECK,
                    MPFROMSHORT ( TRUE ) ,
                    NULL ) ;

WinSendDlgItemMsg ( hwndWnd,
                    IDC_AUTO3STATE,
                    BM_SETCHECK,
                    MPFROMSHORT ( 2 ) ,
                    NULL ) ;
```

The last step in the dialog initialization procedure sends a BM_SETCHECK to both the AUTOCHECKBOX check box and the AUTO3STATE check box. Also, the three-state check box is started in the indeterminate (or gray-scaled) state by specifying 2 as *mpParm2*.

## Button Actions

The icon button style operates just like a push button. An icon button is identical to a push button in appearance except for the image on top, and sends a WM_COMMAND message to its owner when it is pressed. Check boxes and radio buttons will send a WM_CONTROL message to their owners only when selected.

## Summary

Buttons are the easiest control to program. The three varieties of buttons are push buttons, radio buttons, and check boxes. A push button should be used to indicate an action choice, such as "Save," or a routing choice, such as "Include" or "Delete." A radio button should be used to display mutually exclusive choices, and should always be paired with at least one other radio button in a field. A radio button should not be used when a valid user choice is no selection; instead, a check box should be used. A check box should be used to display a binary choice—that is, a choice with two distinct states. Programmers should make sure that both the checked and unchecked states are clearly understandable from the check box text.

# Chapter 17

# Entry Fields

The entry field is perhaps one of the most widely used controls, with possible contenders being the button and the list box (See Figure 17.1). It provides the capability to receive a single line of input as well as to display text as if it were a scrollable static text field. It is also useful for just that: reading or displaying a single line of text. Entry fields are simple controls; *multiline edit controls* (MLEs) which are discussed in Chapter 18, should be used in situations where more complex functionality is required. Simplicity in function does have its advantages, as the entry field also is probably one of the easiest controls to write code for.

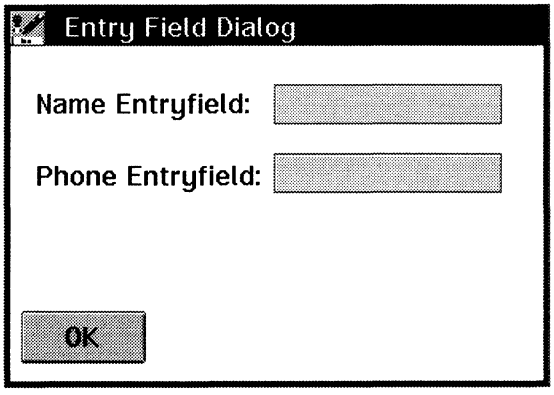



Figure 17.1 Entry field.

The entry field does lack some capabilities that would be very nice to have. For example, being able to accept only certain types of text, having a fully functional *picture string* capability (à la COBOL), and being able to force all text to be upper- or lowercase would be handy. Chapter 27 addresses the issue of adding function to an existing control and illustrates its concepts by an example that allows numeric input only in an entry field.

## Entry Field Basics

Table 17.1 shows the various styles available for the entry field.

Table 17.1 Entry Field Styles

| Style | Description |
|---|---|
| ES_LEFT | Text is left justified. |
| ES_CENTER | Text is center justified. |
| ES_RIGHT | Text is right justified. |
| ES_AUTOSCROLL | Text is scrolled as the cursor moves beyond the visible portion of the entry field. |
| ES_MARGIN | A margin is drawn around the entry field. |
| ES_AUTOTAB | When the maximum number of characters has been entered, the input focus is passed to the next control with the WS_TABSTOP style. |
| ES_READONLY | Text is not modifiable. |
| ES_COMMAND | The entry field is denoted a command entry field. The Help Manager uses this to provide help for the contents of the field, should the user request it. There should be only one entry field per window with this style. |
| ES_UNREADABLE | Text is displayed as a string of asterisks ('*'), one per character of the actual text. |
| ES_AUTOSIZE | The entry field will size itself automatically to insure that the text fits within the visible portion of the control. |
| ES_ANY | The entry field can contain single- and double-byte characters. If the text is converted from an ASCII code page to an EBCDIC code page, there may be an overrun in the converted text. Contrast this with ES_MIXED, where this is not allowed. |
| ES_SBCS | Text is comprised of single-byte characters only. |
| ES_DBCS | Text is comprised of double-byte characters only. |
| ES_MIXED | Text can contain either single- or double-byte characters, which may later be converted to or from an ASCII code page from or to an EBCDIC code page. |

Table 17.1 shows that numerous possibilities exist for creating entry fields. The ES_READONLY style is especially handy for displaying long strings of text for which there is no space; the ES_UNREADABLE is useful for getting information such as passwords from the user in cases where a passerby should not be able to—at a casual glance—perceive the contents.

The entry field is, again, an uncomplicated control; sometimes this leads to inconsistencies with other controls. For example (this applies to buttons also), the *WinSetWindowText* function is used to set the contents of an entry field.

```
BOOL WinSetWindowText(HWND hwndWindow, PSZ pszText);
```

*hwndWindow* is the handle of the entry field to set the text of, and *pszText* is a pointer to the text. The inconsistency is that, as will be seen in other controls, text is usually set—and queried—through messages. However, why overcomplicate things unnecessarily?

As we implied, the text also is queried through a function—the *WinQueryWindowText* function.

```
BOOL WinQueryWindowText(HWND hwndWindow,
                        ULONG ulSzBuffer,
                        PSZ pszBuffer);
```

Again, *hwndWindow* is the handle of the entry field we are querying. *ulSzBuffer* specifies the size of the buffer, and *pszBuffer* points to the receiving buffer. A companion function is helpful here; *WinQueryWindowTextLength* returns the length of the window text.

```
BOOL WinQueryWindowTextLength(HWND hwndWindow);
```

It takes a single parameter—*hwndWindow*—which indicates the window to be queried.

It should be noted that the default maximum text length of an entry field is only 32 bytes. While this may be large enough for most instances, at times a different length might be preferred—to limit the field to 5 characters for a Zip code or increase it to 256 for a file name, for example. This is accomplished by sending the entry field an EM_SETTEXTLIMIT message; passing the maximum number of characters in the first parameter will do the trick.

**Gotcha!**

The limit in the EM_SETTEXTLIMIT message should not include the terminating null character, but the extra byte should be allotted when calling *WinQueryWindowText* and 1 should be added to *WinQueryWindowTextLength*. An interesting point is that, while there is a message for setting the limit, there is no message for querying the limit. This querying can be accomplished using a voodoo incarnation of the WM_QUERYWINDOWPARAMS message, but that seems to be a lot of work for something so simple.

## Selection Basics

Many operations in Presentation Manager programming deal with *selected* items. IBM's Common User Access (CUA) guidelines define a set of different attributes that an object can have, and being *selected* is one of them. A selected object is an object on which an action is to be performed.

Selections have two defining characteristics—an *anchor point* and a *cursor point*. The anchor point is the place where the selection begins; the selection continues until it reaches the *cursor* point, which is where the input cursor is at any given time.

Selections can be performed using either the mouse or the keyboard. Using the keyboard, the arrow keys are used to move the cursor to the desired anchor point; then the arrow keys are used while holding down either *shift* key to expand and contract the text selection. Selecting with the mouse can be done in two ways: *swipe* selection and *shift-click* selection.

*Swiping* is the method by which the mouse is moved to the desired position, the first mouse button is pressed and held, and the mouse is moved over the items to be selected. This is similar in action to direct manipulation, but the intention is different. *Shift-click* selection is closer to using the keyboard; the mouse is clicked at the desired anchor point and then clicked again while the *shift* key is held down to set the cursor point and thus the selected text.

When something is selected, it is given *selection emphasis*, and this is usually conveyed by displaying selected items in *reverse*; this is true for entry fields. Specifically for entry fields (and a few other controls, as we'll see in other chapters), once a section of text is selected, it can be manipulated. For example, any keypress replaces the selected text with the key pressed. If something is pasted from the *clipboard*, which is discussed in the next section, it replaces the selected text.

For the programmer, fortunately, two important messages refer to selections—EM_SETSEL and EM_QUERYSEL; the former sets the current selection and the latter queries what the current selection is, if one exists. See Appendix A for the specifics of each message.

## The Entry Field and the Clipboard

No engineer can do without one; it is indispensible in meetings when a person needs to write and there is no table. A *clipboard* is what we are referring to. For those who do not know what it is, it is a piece of compressed wood—usually slightly larger than a sheet of paper—with a metal clip on top to hold papers in place when it is written on. Most, if not all, windowing systems have a beast of the same name, although (usually) the purpose is a bit different: A clipboard in a GUI environment is used for the temporary placement of data so that it may be copied to other places, whether in the same application that placed the data there or not.

From the viewpoint of an entry field, there are three interfaces to the clipboard, all via messages. The EM_CUT message removes the selected text and places it on the clipboard. The EM_COPY message copies the selected text onto the clipboard, but the text remains in the entry field. The EM_PASTE message copies the data from the clipboard and inserts it either at the current cursor position or, if there is selected text in the entry field, replaces the currently selected text. Again, see Appendix A for the specifics of each message.

## And Other Things

Already we have a control that is quite usable. However, IBM provided some additional functionality. Two of these are *read only* and *unreadable data*, and they are specified by the two window styles ES_READONLY and ES_UNREADABLE.

The effect of ES_READONLY is rather obvious—it prevents the user from changing the contents of the entry field. Text may be selected and copied to the clipboard, but it may not be cut from the entry field, nor may other text be pasted into the entry field. The need for this is evident when text of an indeterminable length must be displayed on a fixed amount of screen "real estate." Using an entry field allows the text to be placed in the required space, because it can be scrolled so that the entire text can be viewed.

The implementation of ES_UNREADABLE is difficult to fathom. While the purpose is evident—to prevent the contents from being viewed—the method by which this is achieved is not. Currently, each character is displayed as an asterisk; this is a poor choice, since the most frequent application of ES_UNREADABLE is for computer passwords, where using an asterisk eliminates the need to guess how many letters are in the value. A programmer who needs to provide secure access should not use the ES_UNREADABLE style.

## ENTRY1—Entry Field Samples

The following application displays some entry fields with different styles. The point is not to demonstrate any particular piece of code, for the entry field is very simple-minded; its intended purpose is to show the effects of the various styles that an entry field can have.

**ENTRY.C**

```
#define INCL_WINDIALOGS
#define INCL_WINENTRYFIELDS
#include <os2.h>
#include "entryrc.h"
```

```
MRESULT EXPENTRY entryDlgProc(HWND hwndWnd,
                              ULONG ulMsg,
                              MPARAM mpParm1,
                              MPARAM mpParm2)
{
   switch (ulMsg) {
   case WM_INITDLG:
      {
         WinSetDlgItemText(hwndWnd,DEF_ENTRY1,"Left");
         WinSetDlgItemText(hwndWnd,DEF_ENTRY2,"Center");
         WinSetDlgItemText(hwndWnd,DEF_ENTRY3,"Right");
         WinSetDlgItemText(hwndWnd,DEF_ENTRY4,"Read only");
         WinSetDlgItemText(hwndWnd,DEF_ENTRY5,"Unreadable");
      }
      break;
   default:
      return WinDefDlgProc(hwndWnd,ulMsg,mpParm1,mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(VOID)
{
   HAB habAnchor;
   HMQ hmqQueue;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinDlgBox(HWND_DESKTOP,
             HWND_DESKTOP,
             entryDlgProc,
             NULLHANDLE,
             DLG_ENTRYFIELDS,
             NULL);

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

## ENTRY.RC

```
#include <os2.h>
#include "entryrc.h"

DLGTEMPLATE DLG_ENTRYFIELDS LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG "Entryfield Sample", DLG_ENTRYFIELDS, 111, 29, 150, 130,
      WS_VISIBLE,
      FCF_SYSMENU | FCF_TITLEBAR
   {
      GROUPBOX "Alignment", -1, 5, 70, 140, 55, NOT WS_GROUP
      LTEXT "Left aligned", -1, 10, 105, 65, 8, NOT WS_GROUP
      LTEXT "Center aligned", -1, 10, 90, 65, 8, NOT WS_GROUP
      LTEXT "Right aligned", -1, 10, 75, 65, 8, NOT WS_GROUP
      GROUPBOX "Miscellaneous", -1, 5, 30, 140, 40, NOT WS_GROUP
      LTEXT "Read-only", -1, 9, 50, 65, 8, NOT WS_GROUP
      LTEXT "Unreadable", -1, 10, 35, 65, 8, NOT WS_GROUP
      ENTRYFIELD "", DEF_ENTRY1, 82, 105, 56, 8, ES_MARGIN |
         WS_GROUP | ES_AUTOSCROLL
      ENTRYFIELD "", DEF_ENTRY2, 82, 90, 56, 8, ES_CENTER |
         ES_MARGIN | ES_AUTOSCROLL
      ENTRYFIELD "", DEF_ENTRY3, 82, 75, 56, 8, ES_RIGHT |
         ES_MARGIN | ES_AUTOSCROLL
```

```
      ENTRYFIELD "", DEF_ENTRY4, 82, 50, 56, 8, ES_MARGIN |
         ES_READONLY | WS_GROUP | ES_AUTOSCROLL
      ENTRYFIELD "", DEF_ENTRY5, 82, 35, 56, 8, ES_MARGIN |
         ES_UNREADABLE | ES_AUTOSCROLL
      DEFPUSHBUTTON "Cancel", DID_CANCEL, 11, 10, 40, 13,
         WS_GROUP
   }
}
```

## ENTRYRC.H

```
#define DLG_ENTRYFIELDS          256
#define DEF_ENTRY1               257
#define DEF_ENTRY2               258
#define DEF_ENTRY3               259
#define DEF_ENTRY4               260
#define DEF_ENTRY5               261
```

## ENTRY.MAK

```
APP=ENTRY

$(APP).EXE:                      $(APP).OBJ \
                                 $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                      $(APP).RC \
                                 $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                      $(APP).C \
                                 $(APP)RC.H
        ICC -C+ -Kb+ -Ss+ $(APP).C
```

## ENTRY.DEF

```
NAME ENTRY WINDOWAPI

DESCRIPTION 'Entryfields Sample
Copyright (c) 1993-1995 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

# Chapter 18

# Multiline Edit Controls

When OS/2 1.1 was released in the middle of the Macintosh™ era, many people wondered why it didn't have a control similar to that used in any of the Mac's popular, easy-to-use word processors. IBM's answer in OS/2 1.2 was the *multiline edit control* (usually abbreviated as MLE); this control provided a similar yet simpler version of what people saw on the Macintosh. It supported the multiline text entry and browsing that they were familiar with and the anchor point/cursor point selection style discussed in Chapter 17.

But let's not stop there: The MLE was also one of the first controls to support a selectable font, and it can handle very large text buffers easily. Being a *stream-based* editing control means that word wrap also came cheaply. Finally, it included a primitive *undo* capability.

Unfortunately, IBM tried (and failed) to emulate the Macintosh; it has no multifont capability, which contributed heavily to the ease-of-use that made the Mac such a big seller. Also, it seems clumsily written. Even with all of these problems, the MLE still is quite usable and is nifty for grabbing a chunk of text from the user when needed. MLEs are used everywhere—in the WPS (settings pages), in containers (editing icon text), and so on.

## Terminology, Etc.

Table 18.1 shows the styles available for the MLE control.

Table 18.1 MLE Styles

| Style | Description |
|---|---|
| MLS_BORDER | Creates an MLE with a surrounding border. |
| MLS_DISABLEUNDO | Specifies that the MLE should ignore undo actions. |
| MLS_HSCROLL | Specifies that the MLE should have a horizontal scrollbar. |
| MLS_IGNORETAB | Specifies that the MLE should ignore the *tab* key and instead pass the WM_CHAR message to its owner. |
| MLS_READONLY | Creates an MLE that is read-only. |
| MLS_WORDWRAP | Specifies that the MLE should wrap words to the next line that do not fit on the current line. |
| MLS_VSCROLL | Specifies that the MLE should have a vertical scrollbar. |

The MLE has a concept of an *import/export buffer* that is used to set and query the text in the control (called *importing* and *exporting* text). Also, since the control is used frequently to read from and write to files, the MLE supports different end-of-line formats.

Table 18.2 MLE End-of-Line Formats

| Format | Description |
|---|---|
| CR-LF | A carriage return (CR) followed by a line feed (LF) denotes the end of a line. |
| LF | LF denotes the end of a line. |
| Windows MLE | On import, CR CR LF is ignored, and CR LF is interpreted as end of line. On export, CR LF is used to specify end of line and CR CR LF is used to denote line breaks caused by word wrapping. |

To set the import/export buffer, the MLE expects to receive an MLM_SETIMPORTEXPORT message before receiving any MLM_IMPORT (import text from buffer) or MLM_EXPORT (export text to buffer). The format of the text to be imported or exported is specified in the MLM_FORMAT message. The MLM_SETIMPORTEXPORT message simply tells the MLE the address of the buffer to be used in later message; thus, if this message is sent followed immediately by an MLM_IMPORT message, whatever was in the buffer will get imported. Similarly, multiple MLM_IMPORT messages can be sent to import the same text multiple times.

Additional messages that correspond to well-known or easily understood capabilities are the MLM_SETSEL (set selection) and MLM_SETWRAP (set word wrap) messages.

Two items need to be noted. The first is the concept of an *insertion point* (datatype is IPT), which is simply an offset in the MLE from the beginning of the text. The second is that of line numbers; it may seem obvious since we are programming using a language whose arrays begin at index 0, but it doesn't hurt to state explicitly that line numbers, when used in the various MLE messages, begin at 0 also.

# MLE1

The following sample shows an MLE and performs some rudimentary operations with it.

**MLE1.C**

```
#define INCL_WINFRAMEMGR
#define INCL_WINMLE
#define INCL_WINMENUS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mle1rc.h"

#define CLS_CLIENT                "MLE1SampleClass"

#define MYM_ACTIVATE              (WM_USER)

typedef struct _INSTDATA {
   ULONG ulSzStruct;              // Size of the structure
   HAB habAnchor;                 // Anchor block handle
   HWND hwndFrame;                // Frame window handle
   HWND hwndMle;                  // MLE window handle
} INSTDATA, *PINSTDATA;

VOID addText(HWND hwndMle)
//------------------------------------------------------------------
// This function adds text to the MLE starting at position 0.
//
// Input:  hwndMle - MLE window handle
//------------------------------------------------------------------
{
   CHAR achImpExp[256];
```

```
   USHORT usIndex;
   IPT iInsert;

   //-------------------------------------------------------------
   // Set the import/export buffer and the format to
   // MLFIE_NOTRANS.  Remember that, internally, a '\n'
   // character is a LF only.  Only when it gets output to a
   // "text-mode" stream does it get converted to CR-LF.
   //-------------------------------------------------------------
   WinSendMsg(hwndMle,
              MLM_SETIMPORTEXPORT,
              MPFROMP(achImpExp),
              MPFROMLONG(sizeof(achImpExp)));
   WinSendMsg(hwndMle,
              MLM_FORMAT,
              MPFROMLONG(MLFIE_NOTRANS),
              0);

   //-------------------------------------------------------------
   // Insert 20 lines of text
   //-------------------------------------------------------------
   iInsert=0;

   for (usIndex=1; usIndex<=20; usIndex++) {
      sprintf(achImpExp,"This is line %d.\n",usIndex);

      WinSendMsg(hwndMle,
                 MLM_IMPORT,
                 MPFROMP(&iInsert),
                 MPFROMLONG(strlen(achImpExp)));
   } /* endfor */
}

VOID selectAllText(HWND hwndMle)
//-------------------------------------------------------------------
// This function selects all of the text.
//
// Input:  hwndMle - MLE window handle
//-------------------------------------------------------------------
{
   ULONG ulSzText;

   //-------------------------------------------------------------
   // Query the amount of text in the MLE.  This will be our
   // cursor point.  Our anchor point is always 0.
   //-------------------------------------------------------------
   ulSzText=LONGFROMMR(WinSendMsg(hwndMle,
                                  MLM_QUERYTEXTLENGTH,
                                  0,
                                  0));

   WinSendMsg(hwndMle,
              MLM_SETSEL,
              MPFROMLONG(0),
              MPFROMLONG(ulSzText));
}

MRESULT EXPENTRY clientWndProc(HWND hwndWnd,
                               ULONG ulMsg,
                               MPARAM mpParm1,
                               MPARAM mpParm2)
//-------------------------------------------------------------------
// Client window procedure.
//
// Input, Output, Returns:  message-specific
//-------------------------------------------------------------------
{
```

```
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);

   switch (ulMsg) {
   case WM_CREATE:
      //-------------------------------------------------------
      // Allocate memory for the instance data.
      //-------------------------------------------------------
      pidData=calloc(1,sizeof(INSTDATA));
      if (pidData==NULL) {
         WinAlarm(HWND_DESKTOP,WA_ERROR);
         return MRFROMSHORT(TRUE);
      } /* endif */

      //-------------------------------------------------------
      // Initialize the instance data.
      //-------------------------------------------------------
      pidData->ulSzStruct=sizeof(INSTDATA);

      WinSetWindowPtr(hwndWnd,0,pidData);

      pidData->habAnchor=WinQueryAnchorBlock(hwndWnd);
      pidData->hwndFrame=WinQueryWindow(hwndWnd,QW_PARENT);

      //-------------------------------------------------------
      // Create the MLE.
      //-------------------------------------------------------
      pidData->hwndMle=WinCreateWindow(hwndWnd,
                                       WC_MLE,
                                       "",
                                       WS_VISIBLE |
                                          MLS_HSCROLL |
                                          MLS_VSCROLL |
                                          MLS_BORDER,
                                       0,
                                       0,
                                       0,
                                       0,
                                       hwndWnd,
                                       HWND_TOP,
                                       WND_MLE,
                                       NULL,
                                       NULL);
      if (pidData->hwndMle==NULLHANDLE) {
         WinAlarm(HWND_DESKTOP,WA_ERROR);
         free(pidData);
         return MRFROMSHORT(TRUE);
      } /* endif */
      break;
   case WM_DESTROY:
      free(pidData);
      break;
   case WM_SIZE:
      //-------------------------------------------------------
      // Resize the MLE according to our new dimensions.
      //-------------------------------------------------------
      WinSetWindowPos(pidData->hwndMle,
                      NULLHANDLE,
                      0,
                      0,
                      SHORT1FROMMP(mpParm2),
                      SHORT2FROMMP(mpParm2),
                      SWP_SIZE);
      break;
```

```
case WM_SETFOCUS:
   //-------------------------------------------------------
   // If we are getting the focus, post ourselves a message
   // to change it to the MLE.  We may not do it now,
   // because PM is still in the focus change processing,
   // meaning that any call to WinSetFocus() will get
   // overwritten when the focus change processing completes.
   //-------------------------------------------------------
   if (SHORT1FROMMP(mpParm2)) {
      WinPostMsg(hwndWnd,MYM_SETFOCUS,0,0);
   } /* endif */
   break;
case MYM_SETFOCUS:
   WinSetFocus(HWND_DESKTOP,pidData->hwndMle);
   break;
case WM_COMMAND:
   switch (SHORT1FROMMP(mpParm1)) {
   case MI_ADDTEXT:
      addText(pidData->hwndMle);
      break;
   case MI_SELECTALL:
      selectAllText(pidData->hwndMle);
      break;
   case MI_READONLY:
      {
         BOOL bReadOnly;
         HWND hwndMenu;

         //-------------------------------------------------
         // Query the current read-only state and toggle.
         //-------------------------------------------------
         bReadOnly=SHORT1FROMMR(
                     WinSendMsg(pidData->hwndMle,
                                MLM_QUERYREADONLY,
                                0,
                                0));
         bReadOnly=!bReadOnly;

         //-------------------------------------------------
         // Set the new read-only state.
         //-------------------------------------------------
         WinSendMsg(pidData->hwndMle,
                    MLM_SETREADONLY,
                    MPFROMSHORT(bReadOnly),
                    0);

         //-------------------------------------------------
         // Check or uncheck the menu item.
         //-------------------------------------------------
         hwndMenu=WinWindowFromID(pidData->hwndFrame,
                                  FID_MENU);
         WinSendMsg(hwndMenu,
                    MM_SETITEMATTR,
                    MPFROM2SHORT(MI_READONLY,TRUE),
                    MPFROM2SHORT(MIA_CHECKED,
                                 bReadOnly?MIA_CHECKED:0));
      }
      break;
   case MI_EXIT:
      WinPostMsg(hwndWnd,WM_CLOSE,0,0);
      break;
   default:
      return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
   } /* endswitch */
   break;
default:
   return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
```

```
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(VOID)
//---------------------------------------------------------------
// Standard main function for PM applications.
//---------------------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
            FCF_SIZEBORDER | FCF_MENU | FCF_SHELLPOSITION |
            FCF_TASKLIST;

   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_CLIENT,
                                "MLE Sample 1",
                                0,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      } /* endwhile */

      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

**MLE1.RC**

```
#include <os2.h>
#include "mle1rc.h"

MENU RES_CLIENT
{
   SUBMENU "~Sample", M_SAMPLE
   {
      MENUITEM "~Add text", MI_ADDTEXT
      MENUITEM "~Select all", MI_SELECTALL
      MENUITEM "~Read only", MI_READONLY
      MENUITEM SEPARATOR
      MENUITEM "E~xit", MI_EXIT
   }
}
```

**MLE1RC.H**

```
#define RES_CLIENT                256
#define WND_MLE                   257

#define M_SAMPLE                  320
#define MI_ADDTEXT                321
#define MI_SELECTALL              322
#define MI_READONLY               323
#define MI_EXIT                   324
```

**MLE1.MAK**

```
APP=MLE1

$(APP).EXE:                       $(APP).OBJ \
                                  $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                       $(APP).RC \
                                  $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                       $(APP).C \
                                  $(APP)RC.H
        ICC -C+ -Kb+ -Ss+ $(APP).C
```

**MLE1.DEF**

```
NAME MLE1 WINDOWAPI NEWFILES

DESCRIPTION 'MLE Sample 1
Copyright (c) 1994-1995 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

The code does most of the important work in *addText* and *selectAllText*.

```
WinSendMsg(hwndMle,
           MLM_SETIMPORTEXPORT,
           MPFROMP(achImpExp),
           MPFROMLONG(sizeof(achImpExp)));
WinSendMsg(hwndMle,
           MLM_FORMAT,
           MPFROMLONG(MLFIE_NOTRANS),
           0);
```

As was stated earlier, the import/export transfer buffer must be set before any text is imported. Also, since the internal representation of a new-line character is simply a line feed, we have to tell the MLE that the format of the imported text is just that (MLFIE_NOTRANS).

```
iInsert=0;

for (usIndex=1; usIndex<=20; usIndex++) {
   sprintf(achImpExp,"This is line %d.\n",usIndex);

   WinSendMsg(hwndMle,
              MLM_IMPORT,
              MPFROMP(&iInsert),
              MPFROMLONG(strlen(achImpExp)));
} /* endfor */
```

Finally, we loop to insert 20 lines of text. As can be seen in the message section at the end of this chapter, MLM_IMPORT updates *mpParm1* to reflect the point just after the place where the last character was inserted; this is to prepare the application for the next import (or export, for MLM_EXPORT).

The processing of the input focus is interesting.

```
case WM_SETFOCUS:
   if (SHORT1FROMMP(mpParm2)) {
      WinPostMsg(hwndWnd,MYM_SETFOCUS,0,0);
   } /* endif */
   break;
case MYM_SETFOCUS:
   WinSetFocus(HWND_DESKTOP,pidData->hwndMle);
   break;
```

While a focus change is in progress, applications are not supposed to call *WinSetFocus* or *WinFocusChange*. Presentation Manager will not prevent this from being done, but since it has not completed the focus change processing, any window to which the focus is assigned will lose it immediately. The only way to accomplish this—as in the code just given—is to *post* a message that will call *WinSetFocus*. Since posting is being done, not sending, the message gets executed whenever it gets dispatched, which is after the focus change has completed.

## How to Upset a User Rather Quickly

Upon running MLE1, it is noticeable how the control repainted itself whenever any changes took place. Whenever an application does a lot of textual manipulations, this can look rather nasty. Fortunately, two messages can be used to disable and enable screen updates—MLM_DISABLEREFRESH and MLM_ENABLEREFRESH. The first messages tells the MLE that the application is making many changes and that it should not update the display until an MLM_ENABLEREFRESH message is sent.

**Gotcha!**

The MLM_DISABLEREFRESH message does not work as advertised; instead of disabling display updates and disabling the mouse pointer, it simply disables the mouse pointer. A better way to perform this action is to use the *WinEnableWindowUpdate* function specifying FALSE as the second parameter (and reenabling using the same function with TRUE as the second parameter). Also, the MLM_DISABLEREFRESH message disables the mouse systemwide, instead of just over itself, which can be quite annoying for operations that take up large

amounts of time. An application that is guilty of this is the *System Editor*; readers can start the editor and read in a file that is greater than 500K to see an example of this.

## No Refreshment

MLE2 is the next sample to be looked at. It calls *WinEnableWindowUpdate* to disable the window refresh before inserting the text and calls it again to enable the window refresh afterward. Its behavior should be compared with that of MLE1.

**MLE2.C**

```
#define INCL_WINFRAMEMGR
#define INCL_WINMLE
#define INCL_WINMENUS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mle2rc.h"

#define CLS_CLIENT              "MLE2SampleClass"

#define MYM_SETFOCUS            (WM_USER)

typedef struct _INSTDATA {
   ULONG ulSzStruct;            // Size of the structure
   HAB habAnchor;               // Anchor block handle
   HWND hwndFrame;              // Frame window handle
   HWND hwndMle;                // MLE window handle
} INSTDATA, *PINSTDATA;

VOID addText(HWND hwndMle)
//--------------------------------------------------------------
// This function adds text to the MLE starting at position 0.
//
// Input:  hwndMle - MLE window handle
//--------------------------------------------------------------
{
   CHAR achImpExp[256];
   USHORT usIndex;
   IPT iInsert;

   //-----------------------------------------------------------
   // Set the import/export buffer and the format to
   // MLFIE_NOTRANS.  Remember that, internally, a '\n'
   // character is a LF only.  Only when it gets output to a
   // "text-mode" stream does it get converted to CR-LF.
   //-----------------------------------------------------------
   WinSendMsg(hwndMle,
              MLM_SETIMPORTEXPORT,
              MPFROMP(achImpExp),
              MPFROMLONG(sizeof(achImpExp)));
   WinSendMsg(hwndMle,
              MLM_FORMAT,
              MPFROMLONG(MLFIE_NOTRANS),
              0);

   //-----------------------------------------------------------
   // Insert 20 lines of text.
   //-----------------------------------------------------------
   WinEnableWindowUpdate(hwndMle,FALSE);
```

```
   iInsert=0;

   for (usIndex=1; usIndex<=20; usIndex++) {
      sprintf(achImpExp,"This is line %d.\n",usIndex);

      WinSendMsg(hwndMle,
                 MLM_IMPORT,
                 MPFROMP(&iInsert),
                 MPFROMLONG(strlen(achImpExp)));
   } /* endfor */

   WinEnableWindowUpdate(hwndMle,TRUE);
}

VOID selectAllText(HWND hwndMle)
//---------------------------------------------------------------
// This function selects all of the text.
//
// Input:  hwndMle - MLE window handle
//---------------------------------------------------------------
{
   ULONG ulSzText;

   //------------------------------------------------------------
   // Query the amount of text in the MLE.  This will be our
   // cursor point.  Our anchor point is always 0.
   //------------------------------------------------------------
   ulSzText=LONGFROMMR(WinSendMsg(hwndMle,
                                  MLM_QUERYTEXTLENGTH,
                                  0,
                                  0));

   WinSendMsg(hwndMle,
              MLM_SETSEL,
              MPFROMLONG(0),
              MPFROMLONG(ulSzText));
}

MRESULT EXPENTRY clientWndProc(HWND hwndWnd,
                               ULONG ulMsg,
                               MPARAM mpParm1,
                               MPARAM mpParm2)
//---------------------------------------------------------------
// Client window procedure.
//
// Input, Output, Returns:  message-specific
//---------------------------------------------------------------
{
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);

   switch (ulMsg) {
   case WM_CREATE:
      //---------------------------------------------------------
      // Allocate memory for the instance data.
      //---------------------------------------------------------
      pidData=calloc(1,sizeof(INSTDATA));
      if (pidData==NULL) {
         WinAlarm(HWND_DESKTOP,WA_ERROR);
         return MRFROMSHORT(TRUE);
      } /* endif */

      //---------------------------------------------------------
      // Initialize the instance data.
      //---------------------------------------------------------
      pidData->ulSzStruct=sizeof(INSTDATA);

      WinSetWindowPtr(hwndWnd,0,pidData);
```

```
    pidData->habAnchor=WinQueryAnchorBlock(hwndWnd);
    pidData->hwndFrame=WinQueryWindow(hwndWnd,QW_PARENT);

    //---------------------------------------------------------
    // Create the MLE.
    //---------------------------------------------------------
    pidData->hwndMle=WinCreateWindow(hwndWnd,
                                     WC_MLE,
                                     "",
                                     WS_VISIBLE |
                                        MLS_HSCROLL |
                                        MLS_VSCROLL |
                                        MLS_BORDER,
                                     0,
                                     0,
                                     0,
                                     0,
                                     hwndWnd,
                                     HWND_TOP,
                                     WND_MLE,
                                     NULL,
                                     NULL);
    if (pidData->hwndMle==NULLHANDLE) {
       WinAlarm(HWND_DESKTOP,WA_ERROR);
       free(pidData);
       return MRFROMSHORT(TRUE);
    } /* endif */
    break;
 case WM_DESTROY:
    free(pidData);
    break;
 case WM_SIZE:
    //---------------------------------------------------------
    // Resize the MLE according to our new dimensions.
    //---------------------------------------------------------
    WinSetWindowPos(pidData->hwndMle,
                    NULLHANDLE,
                    0,
                    0,
                    SHORT1FROMMP(mpParm2),
                    SHORT2FROMMP(mpParm2),
                    SWP_SIZE);
    break;
 case WM_SETFOCUS:
    //---------------------------------------------------------
    // If we are getting the focus, post ourselves a message
    // to change it to the MLE.  We may not do it now,
    // because PM is still in the focus change processing
    // meaning that any call to WinSetFocus() will get
    // overwritten when the focus change processing completes.
    //---------------------------------------------------------
    if (SHORT1FROMMP(mpParm2)) {
       WinPostMsg(hwndWnd,MYM_SETFOCUS,0,0);
    } /* endif */
    break;
 case MYM_SETFOCUS:
    WinSetFocus(HWND_DESKTOP,pidData->hwndMle);
    break;
 case WM_COMMAND:
    switch (SHORT1FROMMP(mpParm1)) {
    case MI_ADDTEXT:
       addText(pidData->hwndMle);
       break;
    case MI_SELECTALL:
       selectAllText(pidData->hwndMle);
       break;
```

```
      case MI_READONLY:
         {
            BOOL bReadOnly;
            HWND hwndMenu;

            //-------------------------------------------------
            // Query the current read-only state and toggle.
            //-------------------------------------------------
            bReadOnly=SHORT1FROMMR(
                        WinSendMsg(pidData->hwndMle,
                                   MLM_QUERYREADONLY,
                                   0,
                                   0));
            bReadOnly=!bReadOnly;

            //-------------------------------------------------
            // Set the new read-only state.
            //-------------------------------------------------
            WinSendMsg(pidData->hwndMle,
                       MLM_SETREADONLY,
                       MPFROMSHORT(bReadOnly),
                       0);

            //-------------------------------------------------
            // Check or uncheck the menu item.
            //-------------------------------------------------
            hwndMenu=WinWindowFromID(pidData->hwndFrame,
                                     FID_MENU);
            WinSendMsg(hwndMenu,
                       MM_SETITEMATTR,
                       MPFROM2SHORT(MI_READONLY,TRUE),
                       MPFROM2SHORT(MIA_CHECKED,
                                    bReadOnly?MIA_CHECKED:0));
         }
         break;
      case MI_EXIT:
         WinPostMsg(hwndWnd,WM_CLOSE,0,0);
         break;
      default:
         return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
      } /* endswitch */
      break;
   default:
      return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(VOID)
//------------------------------------------------------------------
// Standard main function for PM applications.
//------------------------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);
```

```
   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
               FCF_SIZEBORDER | FCF_MENU | FCF_SHELLPOSITION |
               FCF_TASKLIST;

   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_CLIENT,
                                "MLE Sample 2",
                                0,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      } /* endwhile */

      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

**MLE2.RC**

```
#include <os2.h>
#include "mle2rc.h"

MENU RES_CLIENT
{
   SUBMENU "~Sample", M_SAMPLE
   {
      MENUITEM "~Add text", MI_ADDTEXT
      MENUITEM "~Select all", MI_SELECTALL
      MENUITEM "~Read only", MI_READONLY
      MENUITEM SEPARATOR
      MENUITEM "E~xit", MI_EXIT
   }
}
```

**MLE2RC.H**

```
#define RES_CLIENT              256
#define WND_MLE                 257

#define M_SAMPLE                320
#define MI_ADDTEXT              321
#define MI_SELECTALL            322
#define MI_READONLY             323
#define MI_EXIT                 324
```

**MLE2.MAK**

```
APP=MLE2

$(APP).EXE:                         $(APP).OBJ \
                                    $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                         $(APP).RC \
                                    $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                         $(APP).C \
                                    $(APP)RC.H
        ICC -C+ -Kb+ -Ss+ $(APP).C
```

**MLE2.DEF**

```
NAME MLE2 WINDOWAPI NEWFILES

DESCRIPTION 'MLE Sample 2
Copyright (c) 1994 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

# Clipboard Support

In Chapter 17, we discussed what the clipboard is and which entry-field messages can be used to interface with it. The MLE has a similar set of messages—MLM_COPY, MLM_CUT, and MLM_PASTE—that perform analagous functions. As with the entry field, the first two messages require that some text is selected in the MLE, so these two usually are used in conjunction with the MLM_SETSEL message. Because the concepts associated with the clipboard were explained thoroughly in the last chapter, we will move on to the next topic.

# Navigation without a Sextant

Suppose the insertion point corresponding to a known line number within an MLE has to be found. Or, given an insertion point, the line number where the insertion point can be found has to be determined. Because of the *word-wrap* capability of the MLE, these can be difficult—if not impossible—to calculate without some help from the control. Fortunately, the MLE has two such messages that perform these functions for you; they are MLM_CHARFROMLINE and MLM_LINEFROMCHAR.

# Line by Line

The following example uses the MLM_CHARFROMLINE message to read its contents line by line and to write each line to a file.

**MLE3.C**

```
#define INCL_WINFRAMEMGR
#define INCL_WINMLE
#define INCL_WINMENUS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mle3rc.h"

#define CLS_CLIENT              "MLE3SampleClass"
```

```
#define MYM_SETFOCUS              (WM_USER)

typedef struct _INSTDATA {
   ULONG ulSzStruct;              // Size of the structure
   HAB habAnchor;                 // Anchor block handle
   HWND hwndFrame;                // Frame window handle
   HWND hwndMle;                  // MLE window handle
} INSTDATA, *PINSTDATA;

VOID importText(HWND hwndMle)
//---------------------------------------------------------------
// This function imports text into the MLE.
//
// Input:  hwndMle - MLE window handle
//---------------------------------------------------------------
{
   FILE *pfImport;
   CHAR achImpExp[256];
   IPT iInsert;

   //------------------------------------------------------------
   // Set the import/export buffer and the format to
   // MLFIE_NOTRANS.  Remember that, internally, a '\n'
   // character is a LF only.  Only when it gets output to a
   // "text-mode" stream does it get converted to CR-LF.
   //------------------------------------------------------------
   WinSendMsg(hwndMle,
              MLM_SETIMPORTEXPORT,
              MPFROMP(achImpExp),
              MPFROMLONG(sizeof(achImpExp)));
   WinSendMsg(hwndMle,
              MLM_FORMAT,
              MPFROMLONG(MLFIE_NOTRANS),
              0);

   pfImport=fopen("MLE3.IMP","r");
   if (pfImport==NULL) {
      WinAlarm(HWND_DESKTOP,WA_ERROR);
      return;
   } /* endif */

   WinEnableWindowUpdate(hwndMle,FALSE);

   iInsert=0;

   while (fgets(achImpExp,sizeof(achImpExp),pfImport)!=NULL) {
      WinSendMsg(hwndMle,
                 MLM_IMPORT,
                 MPFROMP(&iInsert),
                 MPFROMLONG(strlen(achImpExp)));
   } /* endwhile */

   WinEnableWindowUpdate(hwndMle,TRUE);

   fclose(pfImport);
}

VOID exportText(HWND hwndMle)
//---------------------------------------------------------------
// This function exports text from the MLE.
//
// Input:  hwndMle - MLE window handle
//---------------------------------------------------------------
{
   FILE *pfExport;
   CHAR achImpExp[256];
```

```
   LONG lNumLines;
   LONG lIndex;
   IPT iBegin;
   LONG lSzLine;

   //-------------------------------------------------------------
   // Set the import/export buffer and the format to
   // MLFIE_NOTRANS.  Remember that, internally, a '\n'
   // character is a LF only.  Only when it gets output to a
   // "text-mode" stream does it get converted to CR-LF.
   //-------------------------------------------------------------
   WinSendMsg(hwndMle,
              MLM_SETIMPORTEXPORT,
              MPFROMP(achImpExp),
              MPFROMLONG(sizeof(achImpExp)));
   WinSendMsg(hwndMle,
              MLM_FORMAT,
              MPFROMLONG(MLFIE_NOTRANS),
              0);

   pfExport=fopen("MLE3.EXP","w");
   if (pfExport==NULL) {
      WinAlarm(HWND_DESKTOP,WA_ERROR);
      return;
   } /* endif */

   WinEnableWindowUpdate(hwndMle,FALSE);

   lNumLines=LONGFROMMR(WinSendMsg(hwndMle,
                                   MLM_QUERYLINECOUNT,
                                   0,
                                   0));

   for (lIndex=0; lIndex<lNumLines; lIndex++) {
      iBegin=LONGFROMMR(WinSendMsg(hwndMle,
                                   MLM_CHARFROMLINE,
                                   MPFROMLONG(lIndex),
                                   0));
      lSzLine=LONGFROMMR(WinSendMsg(hwndMle,
                                    MLM_QUERYLINELENGTH,
                                    MPFROMLONG(iBegin),
                                    0));

      memset(achImpExp,0,sizeof(achImpExp));

      WinSendMsg(hwndMle,
                 MLM_EXPORT,
                 MPFROMP(&iBegin),
                 MPFROMP(&lSzLine));

      fputs(achImpExp,pfExport);
   } /* endfor */

   WinEnableWindowUpdate(hwndMle,TRUE);

   fclose(pfExport);
}

MRESULT EXPENTRY clientWndProc(HWND hwndWnd,
                               ULONG ulMsg,
                               MPARAM mpParm1,
                               MPARAM mpParm2)
//-------------------------------------------------------------
// Client window procedure.
//
// Input, Output, Returns:  message-specific
//-------------------------------------------------------------
{
   PINSTDATA pidData;
```

```
pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);

switch (ulMsg) {
case WM_CREATE:
   //-------------------------------------------------------
   // Allocate memory for the instance data.
   //-------------------------------------------------------
   pidData=calloc(1,sizeof(INSTDATA));
   if (pidData==NULL) {
      WinAlarm(HWND_DESKTOP,WA_ERROR);
      return MRFROMSHORT(TRUE);
   } /* endif */

   //-------------------------------------------------------
   // Initialize the instance data.
   //-------------------------------------------------------
   pidData->ulSzStruct=sizeof(INSTDATA);

   WinSetWindowPtr(hwndWnd,0,pidData);

   pidData->habAnchor=WinQueryAnchorBlock(hwndWnd);
   pidData->hwndFrame=WinQueryWindow(hwndWnd,QW_PARENT);

   //-------------------------------------------------------
   // Create the MLE.
   //-------------------------------------------------------
   pidData->hwndMle=WinCreateWindow(hwndWnd,
                                    WC_MLE,
                                    "",
                                    WS_VISIBLE |
                                       MLS_HSCROLL |
                                       MLS_VSCROLL |
                                       MLS_BORDER,
                                    0,
                                    0,
                                    0,
                                    0,
                                    hwndWnd,
                                    HWND_TOP,
                                    WND_MLE,
                                    NULL,
                                    NULL);
   if (pidData->hwndMle==NULLHANDLE) {
      WinAlarm(HWND_DESKTOP,WA_ERROR);
      free(pidData);
      return MRFROMSHORT(TRUE);
   } /* endif */
   break;
case WM_DESTROY:
   free(pidData);
   break;
case WM_SIZE:
   //-------------------------------------------------------
   // Resize the MLE according to our new dimensions.
   //-------------------------------------------------------
   WinSetWindowPos(pidData->hwndMle,
                   NULLHANDLE,
                   0,
                   0,
                   SHORT1FROMMP(mpParm2),
                   SHORT2FROMMP(mpParm2),
                   SWP_SIZE);
   break;
```

```
   case WM_SETFOCUS:
      //-------------------------------------------------------
      // If we are getting the focus, post ourselves a message
      // to change it to the MLE.  We may not do it now,
      // because PM is still in the focus change processing,
      // meaning that any call to WinSetFocus() will get
      // overwritten when the focus change processing completes.
      //-------------------------------------------------------
      if (SHORT1FROMMP(mpParm2)) {
         WinPostMsg(hwndWnd,MYM_SETFOCUS,0,0);
      } /* endif */
      break;
   case MYM_SETFOCUS:
      WinSetFocus(HWND_DESKTOP,pidData->hwndMle);
      break;
   case WM_COMMAND:
      switch (SHORT1FROMMP(mpParm1)) {
      case MI_IMPORTTEXT:
         importText(pidData->hwndMle);
         break;
      case MI_EXPORTTEXT:
         exportText(pidData->hwndMle);
         break;
      case MI_EXIT:
         WinPostMsg(hwndWnd,WM_CLOSE,0,0);
         break;
      default:
         return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
      } /* endswitch */
      break;
   default:
      return WinDefWindowProc(hwndWnd,ulMsg,mpParm1,mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(VOID)
//------------------------------------------------------------------
// Standard main function for PM applications.
//------------------------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
               FCF_SIZEBORDER | FCF_MENU | FCF_SHELLPOSITION |
               FCF_TASKLIST;
```

```
   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_CLIENT,
                                "MLE Sample 3",
                                0,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,&qmMsg,NULLHANDLE,0,0);
      } /* endwhile */

      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

## MLE3.RC

```
#include <os2.h>
#include "mle3rc.h"

MENU RES_CLIENT
{
   SUBMENU "~Sample", M_SAMPLE
   {
      MENUITEM "~Import text", MI_IMPORTTEXT
      MENUITEM "~Export text", MI_EXPORTTEXT
      MENUITEM SEPARATOR
      MENUITEM "E~xit", MI_EXIT
   }
}
```

## MLE3RC.H

```
#define RES_CLIENT              256
#define WND_MLE                 257

#define M_SAMPLE                320
#define MI_IMPORTTEXT           321
#define MI_EXPORTTEXT           322
#define MI_EXIT                 323
```

## MLE3.MAK

```
APP=MLE3

$(APP).EXE:                     $(APP).OBJ \
                                $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                     $(APP).RC \
                                $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                     $(APP).C \
                                $(APP)RC.H
        ICC -C+ -Kb+ -Ss+ $(APP).C
```

**MLE3.DEF**

```
NAME MLE3 WINDOWAPI NEWFILES

DESCRIPTION 'MLE Sample 3
Copyright (c) 1994 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

The main difference between this sample and the previous two is the addition of the function *exportText*. Its purpose is to read, line by line, the contents of the MLE and to write each line to a file. To do this, we make use of the MLM_QUERYLINECOUNT, MLM_CHARFROMLINE, and MLM_QUERYLINELENGTH messages. First, we need to determine how many lines are in the MLE; the first message does this.

```
lNumLines=LONGFROMMR(WinSendMsg(hwndMle,
                                MLM_QUERYLINECOUNT,
                                0,
                                0));
```

Obviously, we use this as the terminating condition of a *for* loop. Each iteration of the loop performs the following: Determine the offset of the first character on the line using MLM_CHARFROMLINE; query the length of the line using MLM_QUERYLINELENGTH; finally, query the data on the line using MLM_EXPORT.

```
for (lIndex=0; lIndex<lNumLines; lIndex++) {
   iBegin=LONGFROMMR(WinSendMsg(hwndMle,
                                MLM_CHARFROMLINE,
                                MPFROMLONG(lIndex),
                                0));
   lSzLine=LONGFROMMR(WinSendMsg(hwndMle,
                                 MLM_QUERYLINELENGTH,
                                 MPFROMLONG(iBegin),
                                 0));

   memset(achImpExp,0,sizeof(achImpExp));

   WinSendMsg(hwndMle,
              MLM_EXPORT,
              MPFROMP(&iBegin),
              MPFROMP(&lSzLine));

   fputs(achImpExp,pfExport);
} /* endfor */
```

**Gotcha!**

The MLM_QUERYLINECOUNT takes as its parameter an insertion point instead of a line number, as would be imagined.

## Searching for What Was That Again?

An action that is commonly performed on large quantities of text is searching for a particular string. Before digging out Knuth volumes, readers should take note of the MLM_SEARCH message. This message will do both search and search-and-replace actions on the text contained within the MLE. The

method of communication is via the MLE_SEARCHDATA structure, which specifies the string to search for and (optionally) a replacement string.

```
typedef struct _SEARCH
{
   USHORT cb;
   PCHAR  pchFind;
   PCHAR  pchReplace;
   SHORT  cchFind;
   SHORT  cchReplace;
   IPT    iptStart;
   IPT    iptStop;
   USHORT cchFound;
} MLE_SEARCHDATA;
```

*cb* specifies the size of the structure. *pchFind* points to the search text. *pchReplace* points to the text to replace with. *cchFind* specifies the length of the search text. *cchReplace* specifies the length of the replacement text. *iptStart* on entry specifies the search starting point. If this is -1, the cursor position is used. On exit, *iptStart* specifies the insertion point of the first character of the occurrence found, if one is found. *iptStop* specifies the search ending point. If this is -1, the end of text is used. If this is less than *iptStart*, the search wraps to the beginning of the text after it reaches the end. *cchFound* specifies the length of the found text.

*mpParm1* specifies one or more flags that are used to determine the action of the search.

Table 18.3 *mpParm1* in MLM_SEARCH Message

| Flag | Description |
|---|---|
| MLFSEARCH_CASESENSITIVE | Find a string that also matches in case. |
| MLFSEARCH_SELECTMATCH | Select the text if found and scroll the text if necessary to bring it into view. |
| MLFSEARCH_CHANGEALL | Replaces all occurrences of *pchFind* with the text pointed to by *pchReplace* between *iptStart* and *iptStop*. *cchFound* has no meaning because the text has been changed. *iptStart* points to the position where the last change occurred, or is equal to *iptStop* if no occurrences of *pchFind* were found. The current position is not changed; the current selection, however, is deselected. |

Since the MLE can hold a large quantity of text, searches conceivably can take a long time to complete. Because of this, the MLE periodically sends the application a WM_CONTROL message with an MLN_SEARCHPAUSE notification code; this allows the application to halt the search (usually per the user's request); it also can be used to implement a progress indicator.

## As If That Weren't Enough

Finally, there are a number of messages that perform miscellaneous functions. To select a font, there is the MLM_SETFONT message, which is a bit tricky to use since it expects a *font attributes* structure (FATTRS). Fortunately, the Font Dialog (see Chapter 26) returns the FATTRS structure for the font selected, so if we consent to using this (a good idea), we can avoid a lot of work. The current font is returned in an FATTRS structure by the MLM_QUERYFONT message.

**Gotcha!**

The MLM_SETFONT message is the only way to change the font of an MLE control. *WinSetPresParam* will not work as it does with the other window classes.

*Undo* support is provided through three messages: MLM_UNDO, which actually performs the undo operation; MLM_QUERYUNDO, which returns whether an MLM_UNDO will perform an *undo* or *redo* (the opposite of undo) and the class of change that will be undone or redone (text change, font change, etc.); and MLM_RESETUNDO, which indicates to the MLE that it should not allow an undo to occur.

*Text-modified* queries are provided by the MLM_QUERYCHANGED message, and the text-modified flag can be set using the MLM_SETCHANGED message.

# Chapter 19

# Other Window Classes

Quick-minded readers will have observed that there are more window classes available to the programmer than what are listed on the contents page. The remaining window classes, however, either are rarely used directly by an application or are too trivial to warrant a separate chapter. This chapter serves as a catchall to discuss these unmentionables.

Table 19.1 below lists these window classes and provides a brief description of them.

Table 19.1 Window Classes Covered in this Chapter

| Constant | Description |
|---|---|
| WC_COMBOBOX | Combo box. This is a combination of an entry field and a list box. It responds to all messages for both controls; additionally, there are a few messages specifically for this class. |
| WC_FRAME | Frame. This window is used as the primary window for most applications, and is also the basis for dialog windows. While the typical interaction with this class is through *subclassing*, the frame window has some useful messages for the developer. |
| WC_SCROLLBAR | Scrollbar. This can be found in many applications, where the environment is larger than the amount of screen space allocated for the application. Scrollbars allow the user to change the visible portion by *scrolling* the window. |
| WC_STATIC | Static window. This is a window whose contents are *static*, that is, unchangable by the user. Typically, windows of this class are textual in nature, but they can also be icons, bitmaps, and so on. |
| WC_TITLEBAR | Titlebar. On a *standard window*, this window is placed between the *system menu* and the *min/max buttons*. It provides a placement for the frame window text and also allows quick access to window resizing, maximizing, and restoring. |

## Combo Boxes

A *combo box* is displayed as an entry field with either a down arrow displayed to its right or a list box displayed below it. Its primary purpose is to display a list of items that can be selected from but added to at the user's discretion. A *drop-down combo box* is especially useful when screen "real-estate" is limited but a list is still needed; in such cases the down arrow is displayed to the right of an entry field.

Because a combo box is simply a handy way of putting together two existing window classes, the designers decided that it should be able to accept messages for both of its ancestors. Thus, any entry field message (EM_) and list box message (LM_) can be sent to the control with the expected results. The reader is referred to the chapters dealing with those control classes for more information.

The table below lists the combo-box styles.

Table 19.2 Combo-Box Styles

| Style | Description |
|---|---|
| CBS_SIMPLE | Both the entry field and the list box are displayed. Whenever an item in the list box is selected, the text is displayed in the entry field. If the item required is not in the list, the user can type the desired value in the entry field. |
| CBS_DROPDOWN | This is the same as CBS_SIMPLE except the list box is hidden until the user requests that it be shown; this is accomplished either by clicking with the mouse on the down arrow or pressing the Ctrl-Down arrow keys. |
| CBS_DROPDOWNLIST | This is the same as CBS_DROPDOWN except the entry field is *read-only*, meaning items cannot be entered manually by the user. |

The following simple application illustrates the different types of combo boxes.

**COMBO.C**

```
#define INCL_WINDIALOGS
#define INCL_WINENTRYFIELDS
#define INCL_WINLISTBOXES
#include <os2.h>
#include <stdio.h>
#include "comborc.h"

MRESULT EXPENTRY comboDlgProc(HWND hwndWnd,
                              ULONG ulMsg,
                              MPARAM mpParm1,
                              MPARAM mpParm2)
{
   switch (ulMsg) {
   case WM_INITDLG:
      {
         USHORT usIndex;
         CHAR achText[64];

         for (usIndex=1; usIndex<=10; usIndex++) {
            sprintf(achText,"Item %d", usIndex);

            WinInsertLboxItem(WinWindowFromID(hwndWnd,
                                              DCB_CB_COMBO1),
                              LIT_END,
                              achText);
            WinInsertLboxItem(WinWindowFromID(hwndWnd,
                                              DCB_CB_COMBO2),
                              LIT_END,
                              achText);
            WinInsertLboxItem(WinWindowFromID(hwndWnd,
                                              DCB_CB_COMBO3),
                              LIT_END,
                              achText);
         } /* endfor */
      }
```

```
      break;
   default:
      return WinDefDlgProc(hwndWnd,ulMsg,mpParm1,mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(VOID)
{
   HAB habAnchor;
   HMQ hmqQueue;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinDlgBox(HWND_DESKTOP,
             HWND_DESKTOP,
             comboDlgProc,
             NULLHANDLE,
             DLG_COMBOBOXES,
             NULL);

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

## COMBO.RC

```
#include <os2.h>
#include "comborc.h"

DLGTEMPLATE DLG_COMBOBOXES LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG "Combo Boxes", DLG_COMBOBOXES, 67, 31, 200, 160,
      WS_VISIBLE,
      FCF_SYSMENU | FCF_TITLEBAR
   {
        LTEXT "Simple", -1, 10, 130, 75, 8
        LTEXT "Drop down", -1, 10, 80, 75, 8
        LTEXT "Drop down list", -1, 10, 50, 75, 8
        COMBOBOX "", DCB_CB_COMBO1, 90, 95, 100, 45,
           LS_HORZSCROLL | WS_GROUP
        COMBOBOX "", DCB_CB_COMBO2, 90, 45, 100, 45,
           LS_HORZSCROLL | CBS_DROPDOWN
        COMBOBOX "", DCB_CB_COMBO3, 90, 15, 100, 45,
           LS_HORZSCROLL | CBS_DROPDOWNLIST
        DEFPUSHBUTTON "Cancel", DID_CANCEL, 10, 10, 40, 13
   }
}
```

## COMBORC.H

```
#define DLG_COMBOBOXES          256
#define DCB_CB_COMBO1           257
#define DCB_CB_COMBO2           258
#define DCB_CB_COMBO3           259
```

**COMBO.MAK**

```
APP=COMBO

$(APP).EXE:                     $(APP).OBJ \
                                $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                     $(APP).RC \
                                $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                     $(APP).C \
                                $(APP)RC.H
        ICC -C+ -Kb+ -Ss+ $(APP).C
```

**COMBO.DEF**

```
NAME COMBO WINDOWAPI

DESCRIPTION 'Comboboxes Sample
Copyright (c) 1993 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

The combo box, while fairly straightforward in its usage, does have some limitations in its design about which programmers should know. First, there is no easy way to have an ownerdrawn combo box (i.e. ownerdrawn list box within the combo box). This means that, for those with the need to display bitmaps, colors, etc., you're "outta luck." Second, a CBN_SHOWLIST notification indicates when the list is about to be shown, but no corresponding notification indicates when the list is about to be hidden; this one goes in the "honestly, we didn't inhale" group of design idiosyncracies.

**Gotcha!**
When a CBN_SHOWLIST notification is received, the list is *not* shown already, so a CBM_ISLISTSHOWING message will return FALSE. This is documented but often overlooked.

A final note is that combo boxes process the messages and notifications for the entry field and list box by acting as a dispatcher. Thus, code may need slight modifications if it is being copied from another source that was used solely for an entry field/list box and not a combo box. For example, instead of a LN_ENTER notification, there is the CBN_ENTER notification.

# Frames

A *frame* window is, as mentioned in Chapter 9, one of the components of a *standard window*. It's primary purpose is to keep things organized—it receives messages from the various components (e.g., menu, sizing border, etc.) and dispatches them to the appropriate windows with a "need to know;" it is the parent of all of the standard window components, which keeps them contained within its boundaries; it provides a standard look to a standard window (thus the name), giving the feeling of consistency to the system. Because the frame is the parent of all of the components, oftentimes its parent is the desktop itself; when this is the case, it is referred to as the *top-level window* for the application.

Direct interfacing with the frame does not yield many useful functions—the real "meat" of the frame is accessed through subclassing. (See Chapter 27 for more information on subclassing.)

A note on the WM_UPDATEFRAME message: After looking at the description of the message, readers will undoubtedly question the reasoning for such a message, if the client is the one to add or delete the controls. The answer can be said in one word (with a bit of explanation afterward): "housekeeping." Just because a control or two has been added or deleted by the programmer doesn't mean the frame is going to know about it. The programmer *must* indicate the changes to the frame so that it can resize the controls properly when it is resized.

# Scrollbars

Scrollbars are used to allow the user to specify a value, within a specified range. Originally, they were intended as navigational tools within windows (thus, their name), when the viewable area was larger than the visible area. Since then, however, many other purposes have been designed and other, more specialized controls have been created as a result. (See "Combo Boxes" earlier in this chapter, Chapter 24, and Chapter 25.) These controls, however, are still quite useful, even if only for scrolling the contents of a window.

A scrollbar consists of three parts—the *buttons*, the *slidertrack*, and the *thumb*. The *buttons* are found on the ends of the scrollbar, and they are used to adjust the position up or down by a "unit." The *thumb* is a rectangular area in the middle of the scrollbar and is used to adjust the position by an arbitrary amount; usually it also indicates the amount of data visible compared with the total amount of data available. The thumb is sometimes referred to as the *slider*, but we will refrain from doing so in order to avoid confusion with the control of the same name. The *slidertrack* is everything else, and the thumb is contained by the slidertrack; the slidertrack is used to adjust the position up/left or down/right one "page" by clicking above/left or below/right of the thumb.

A few properties are associated with a scrollbar. The first is the *range*; it is an inclusive set of numbers greater than or equal to zero in which the value of the scrollbar can fall. The fact that neither boundary can be less than zero is significant, since application code may have to be adjusted to account for this. When the scrollbar is created, it has the default range 0 to 100. The second property is the *thumbsize*; it indicates to the user the amount of data that is visible relative to the total amount of data available for viewing.

Table 19.3 lists the scrollbar styles.

Table 19.3 Scrollbar Styles

| Style | Description |
|---|---|
| SBS_HORZ | Creates a horizontal scrollbar. |
| SBS_VERT | Creates a vertical scrollbar. |
| SBS_THUMBSIZE | Specifies that the SBCDATA structure in the call to *WinCreateWindow* contains valid values for the *cVisible* and *cTotal* fields. |
| SBS_AUTOTRACK | The scrollbar scrolls as more information is displayed on the screen. |
| SBS_AUTOSIZE | The scrollbar thumb changes size to reflect the amount of data in the window. |

Because the scrollbar is such a simple control, programming it is simple. What is difficult is how the scrollbar is used in an application; it is easy to specify what the valid range of values is and even query the current value, but it isn't as easy to scroll a window appropriately or to change the green component in a color window; these things will not be covered in this chapter because the possibilities are endless.

# Statics

*Static* controls have the dubious role of providing information to the user that cannot be modified by him or her. This information can take many forms, the more common of which is text and icons/bitmaps. However, many people fail to realize that there are many forms of the static control and that this flexibility compensates for its lack of functionality.

Speaking of "lack of functionality," let's describe it in a single sentence. For textual static controls, *WinSetWindowText* and *WinQueryWindowText* set and query the current text being displayed; for bitmapped (including icons) static controls, two messages are used to specify the bitmap or icon handle and query the current handle.

Table 19.4 lists the static control styles.

Table 19.4 Static Control Styles

| Style | Description |
|---|---|
| SS_AUTOSIZE | Specifies that the control is to size itself so that its contents fit. |
| SS_BITMAP | Specifies that the control is to contain a bitmap, and the text of the control specifies the resource id of the bitmap. If the first byte of the text is hexadecimal x'FF', then the second and third bytes are used as the low and high word of the resource id of the bitmap to load, respectively. If the first byte of the text is '#', then the remainder of the text is considered to be an ASCII representation of the resource ID of the bitmap to load. If the text is empty or does not follow the above format, no bitmap is loaded. |
| SS_BKGNDFRAME | Creates a box whose color is that of the background. This is similar to, but not the same as, SS_GROUPBOX. |
| SS_BKGNDRECT | Creates a solid rectangle whose color is that of the background. |
| SS_FGNDFRAME | Creates a box whose color is that of the foreground. This is similar to, but not the same as, SS_GROUPBOX. |
| SS_FGNDRECT | Creates a solid rectangle whose color is that of the foreground. This is often used for background shadowing and very thick underlining. |
| SS_GROUPBOX | Creates a box as in SS_FGNDFRAME, except that the text of the static control is displayed in the top left of the box. This is used to group like controls together with an associated heading. |
| SS_HALFTONEFRAME | Creates a box that has a halftone outline. This is similar to, but not the same as, SS_GROUPBOX. |
| SS_HALFTONERECT | Creates a box filled with halftone shading. This is similar to, but not the same as, SS_GROUPBOX. |
| SS_ICON | The same as SS_BITMAP, except that the resource loaded is expected to be an icon or pointer instead of a bitmap. |
| SS_SYSICON | The same as SS_BITMAP, except that the resource ID that is specified in the text is interpreted as a SPTR_ constant and is used to obtain a system icon as in the *WinQuerySysPointer* function. |
| SS_TEXT | Specifies that the static control is to display the text in the manner specified. See the following text for more information. |

**Gotcha!**

For dialogs containing static controls with the style SS_BITMAP or SS_ICON, the bitmap, icon, or pointer must reside in the resource area of the *executable*. This is true even if the dialog template is defined in the resource area of a DLL. If this behavior is unacceptable, the programmer must use an empty string for the text, load the bitmap, icon, or pointer in the dialog procedure, and specify this as the (already loaded) resource to use by sending the control an SM_SETHANDLE message.

For static controls with the style SS_TEXT, a number of additional styles can be applied that control alignment and word-wrapping. Horizontally, DT_LEFT, DT_CENTER, and DT_RIGHT specify left, center, and right-aligned text. Vertically, DT_TOP, DT_VCENTER, and DT_BOTTOM specify top, center, and bottom-aligned text. Additionally, DT_WORDBREAK can be specified if and only if DT_LEFT and DT_TOP are specified; this indicates that words are to be wrapped to the next line if they do not fit completely within the control's area at the current vertical position. If none of these flags is specified, the default is DT_LEFT and DT_TOP.

Static controls have one other use: Since they do nothing other than display themselves, they are very handy for adding the programmer's behavior within a dialog via subclassing. (See Chapter 27 for more information on subclassing windows.)

## Titlebars

The *titlebar* is a control whose role in the *standard window* is perfunctory, yet it is still quite important. It automatically provides for mouse-oriented changing of the window's position and maximizing and restoring of the window's size. Also, its interaction with the frame insures that, whenever the frame's window text is changed, it is updated to reflect the new text.

Even with this, there isn't much the programmer can do with the titlebar control directly. Its functions are strictly defined and were not built with other uses in mind. There are no titlebar-specific styles, and it accepts only two messages. These are described in Appendix A.

# Chapter 20

# Drag and Drop

While the capability to drag and drop an icon from one window to another has been present since OS/2 1.1, a standardized, robust method for providing this essential function was not introduced until OS/2 1.3 with the *Drg* functions and their associated DM_ messages. But what is drag and drop, really?

Drag and drop is the capability of using the mouse to manipulate directly the transfer and placement of data within single or multiple applications. Objects can either be "moved" or "copied" from a source window to a target window. ("Moved" and "copied" are application-defined concepts.)

Drag and drop can be seen from two viewpoints: from the viewpoint of the *source*, who initiates the drag; and from the aspect of the *target*, which can accept or reject a dragging operation. We will examine both of these as well as what to do once the target is established.

## Tennis, Anyone?

In a nutshell, the source window is responsible for determining that the user is attempting to drag an object, initializing the appropriate data structures, and finally calling either *DrgDrag* or *DrgDragFiles* (a version of *DrgDrag* specifically for file objects). Determining that the user is attempting to drag an object is the easiest part, since the system will send a WM_BEGINDRAG message with the pointer position in *mpParm1*. (This is not entirely true. If a child control receives a WM_BEGINDRAG message, it might alert the programmer to this through a WM_CONTROL message, but it is not required that it do so.)

After it has been decided that a drag operation is necessary, the application needs to allocate and initialize three structure types: DRAGINFO, DRAGITEM, and DRAGIMAGE. (There are actually four; the DRAGTRANSFER structure is used once a target has been established.) The DRAGINFO structure contains information about the drag as an entity. The DRAGITEM structures describe each object being dragged. Finally, the DRAGIMAGE structures each describe the appearance of the object under the pointer while it is being dragged.

```
typedef struct _DRAGINFO {
   ULONG cbDraginfo;
   USHORT cbDragitem;
   USHORT usOperation;
   HWND hwndSource;
   SHORT xDrop;
   SHORT yDrop;
   USHORT cditem;
   USHORT usReserved;
} DRAGINFO, *PDRAGINFO;
```

In the DRAGINFO structure, *cbDraginfo* is the size of the DRAGINFO structure in bytes. *cbDragitem* is the size of the DRAGITEM structure contained therein. *usOperation* is the default operation that can be, but is not required to be, set by the source and inspected by the target; it is a DO_ constant. *hwndSource* is the only field not initialized by *DrgAllocDragInfo*, and it is the handle of the window initiating the drag-and-drop operation. *xDrop* and *yDrop* are the coordinates of the object as dropped. *cdItem* specifies the number of DRAGITEM structures stored along with the DRAGINFO structure. *usReserved* is reserved and must be set to 0.

```
typedef struct _DRAGITEM {
      HWND hwndItem;
      ULONG ulItemID;
      HSTR hstrType;
      HSTR hstrRMF;
      HSTR hstrContainerName;
      HSTR hstrSourceName;
      HSTR hstrTargetName;
      SHORT cxOffset;
      SHORT cyOffset;
      USHORT fsControl;
      USHORT fsSupportedOps;
   } DRAGITEM, *PDRAGITEM;
```

In the DRAGITEM structure, *hwndItem* is the handle of the window with which the target should communicate to transfer the information necessary to complete the operation. The only time this would be different from the *hwndSource* field of the DRAGINFO structure is when an application contains many "standard" windows as the children of the main window. *hstrType* is the *type* of the item represented by the DRAGITEM structure. *hstrRMF* is the *rendering mechanism* used to transfer the information and *format* of the data being transfered. *hstrContainerName* is the name of the container that holds the object being dragged. With a file object, for example, this would be the directory where the file resides. *hstrSourceName* and *hstrTargetName* are the names of the object at its original location and the *suggested* name of the object at the target location. The target does not have to use the suggested name; it is up to the application programmer. *cxOffset* and *cyOffset* specify the offset from the hotspot of the pointer to the lower left corner of the image representing the object and is copied here by the system from the corresponding fields in the DRAGIMAGE structure. *fsControl* specifies one or more DC_ constants describing any special attributes of the object being dragged. Finally, *fsSupportedOps* specifies the operations that can be performed as part of the drag—the object may be copied, moved, linked ("shadowed"), and so on.

```
typedef struct _DRAGIMAGE {
      USHORT cb;
      ULONG fl;
      USHORT cptl;
      LHANDLE hImage;
      SIZEL sizlStretch;
      SHORT cxOffset;
      SHORT cyOffset;
   } DRAGIMAGE, *PDRAGIMAGE;
```

In the DRAGIMAGE structure, *cb* specifies the size of the structure in bytes. *fl* specifies a number of DRG_ constants describing the type of data that is given in this structure.

Table 20.1 DRG_ Constants

| Constant | Description |
|---|---|
| DRG_BITMAP | *hImage* specifies a bitmap handle. |
| DRG_CLOSED | The polygon specified is to be closed. If specified, DRG_POLYGON also must be specified. |
| DRG_ICON | *hImage* specifies an icon handle. |
| DRG_POLYGON | *hImage* specifies an array of POINTL structures. |
| DRG_STRETCH | The bitmap or icon is to be stretched to fit the specified size. If specified, DRG_BITMAP or DRG_ICON also must be specified. |
| DRG_TRANSPARENT | An outline of the icon is to be shown only. If specified, DRG_ICON also must be specified. |

*cPtl* specifies the number of points if *fl* contains DRG_POLYGON. *hImage* can specify one of many things, depending on what flags are set in *fl*, as seen in Table 20.1. *sizlStretch* specifies the size that the bitmap or icon should be stretched to. *cxOffset* and *cyOffset* specify the offset of the lower left corner of the image, relative to the hotspot of the cursor as the object is dragged. These two fields are copied into the DRAGITEM structure.

At this point, probably few of the fields in these structures make any sense. It is important to realize that, because the target will more likely than not exist as part of another process, simple allocation of these structures will not suffice, due to OS/2's memory protection features. They must be allocated in shared memory through the use of the *DrgAllocDraginfo* and *DrgAddStrHandle* functions.

```
PDRAGINFO DrgAllocDraginfo(ULONG cditem);
HSTR DrgAddStrHandle(PSZ psz);
```

The former accepts the number of items being dragged and returns a pointer to the shared DRAGINFO structure, whose individual DRAGITEM structures must be initialized using the *DrgSetDragitem* function. The latter takes a pointer to a string and returns a "string handle"—a pointer to a shared memory block containing (among other things) the string passed to the function.

# Initialization Code for Drag and Drop Source

The following is the typical initialization code used in a Presentation Manager application to initiate a drag-and-drop operation.

```
HWND hwndWindow;
PDRAGINFO pdiDrag;
DRAGITEM diItem;

pdiDrag=DrgAllocDraginfo(1);

//-------------------------------------------------------------
// Note that DrgAllocDraginfo() initializes all of the DRAGINFO
// fields *except* hwndSource.
//-------------------------------------------------------------
pdiDrag->hwndSource=hwndWindow;

diItem.hwndItem=hwndWindow;
diItem.ulItemID=1L;                    // Unique identifier
diItem.hstrType=DrgAddStrHandle(DRT_TEXT);
diItem.hstrRMF=DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
diItem.hstrContainerName=DrgAddStrHandle("C:\");
diItem.hstrSourceName=DrgAddStrHandle("CONFIG.SYS");
diItem.hstrTargetName=DrgAddStrHandle("CONFIG.BAK");
```

```
        diItem.cxOffset=0;
        diItem.cyOffset=0;
        diItem.fsControl=0;
        diItem.fsSupportedOps=DO_COPYABLE;

        DrgSetDragItem(pdiDrag,&diItem,sizeof(diItem),0);
```

The following sections will explain this listing in more detail.

## Things Never Told to the Programmer That Should Have Been

Before actually taking our forceps to the code, a few concepts need to be introduced. The first is that of the *type* and the *true type* of an object being dragged. The *type* is just that—a string that describes what the object consists of. The *true type* is a type that more accurately describes the object, if such a true type exists. For example, a file that contains C source code might have the type "Plain Text" but have a true type of "C Code." An object can have more than one type, with each separated by commas and the true type appearing as the first type listed. Thus, the *hstrType* field for the C source code would be initialized as *DrgAddStrHandle("C Code, Plain Text")*. OS/2 defines a set of standard types in the form of DRT_ constants.

The second concept that needs to be discussed is the *rendering mechanism and format* (RMF). The *rendering mechanism* is the method by which the data will be communicated from the source to the target. The *format* is the format of the data if the corresponding rendering mechanism as used to transfer the data. These RMF pairs take the form "<rendering_mechanism, format>" , with multiple RMF pairs separated by commas. OS/2 also defines a set of rendering mechanisms, although no constants are defined for them.

Note that if programmers have a fully populated set of RMF pairs ("fully populated" meaning that for every rendering mechanism, every format is available), a shorthand cross-product notation can be used. For example, if there are the rendering mechanisms RA, RB, and RC and the formats FA, FB, and FC, and the following RMF pairs are available:

"<RA,FA>,<RA,FB>,<RA,FC>,<RB,FA>,<RB,FB>,<RB,FC>,<RC,FA>,
<RC,FB>,<RC,FC>"

then this can be represented as "(RA,RB,RC)X(FA,FB,FC)". Obviously, this is a much more concise way of describing the mess. If the thought of having to parse such a monster with so many different combinations just to discover if <RD,FD> is supported drives programmers crazy, they should have no fear—there are functions that will determine this.

Analogous to the relationship between type and true type, there also exists a *native RMF*, which describes the preferred RMF for this object. It is always the first RMF pair listed or the first RMF pair generated in a cross-product. The native RMF might employ faster data transfer algorithms or other such performance boosters, so it should be used by the target whenever possible.

Just because OS/2 defines sets of types, rendering mechanisms, and formats doesn't mean programmers are limited to those sets. If an application needs to use a new format, it can register the appropriate strings describing this with the *DrgAddStrHandle* function. However, the transfer protocol for the rendering mechanisms and the corresponding data formats also should be published so that other applications can understand the new type of RMF.

The next concepts are that of *source name*, *source container Drag and drop:*, and *target name Drag and drop:*. The *source name* is the name of the object being dragged. It is useful because the target application may be able to perform the requested operation without having to interact with the source application. Typically, this is used when dealing with files. The *source container* describes where the object resides. This, again, is useful to the target when deciding how to complete the action. When dealing with files, for example, the source container would be the directory name containing the file. Finally, the *target name* is actually a suggested name, since the target could determine that an object with that name already exists and that the object will receive a new, unique name.

Now that these concepts have been explained, the structures and sample code shown earlier in this chapter should be easier to understand. We are dragging one item, as evidenced in the *DrgAllocDraginfo* call. The one item is of type "text" and will be transferred via the file system using the format "unknown." The file system object resides in the container/directory "C:\" and has the name "CONFIG.SYS". The suggested target name is "CONFIG.BAK", although the target application is free to select a different name.

# Direct Manipulation Is a Real Drag

Assuming that the last section has been understood and that programmers have successfully (and correctly) initialized the DRAGINFO structure and each DRAGITEM structure for each object, we are now ready to call the function that makes all of this hard work worthwhile: *DrgDrag.*

```
(HWND)DrgDrag(HWND hwndSource,
              PDRAGINFO pdiDragInfo,
              PDRAGIMAGE pdiDragImage,
              ULONG ulNumImages,
              ULONG ulTerminateKey,
              PVOID pvReserved);
```

*hwndSource* is the handle of the window initiating the drag operation. *pdiDragInfo* points to the DRAGINFO structure returned from *DrgAllocDraginfo*. *pdiDragImage* points to an array of one or more DRAGIMAGE structures, and *ulNumImages* specifies how many images the array contains. *ulTerminateKey* describes the manner by which the drag is ended and is a VK_ constant.

Table 20.2 Description of VK_ Constants in a Drag Operation

| Constant | Description |
|---|---|
| VK_BUTTON1 | Drag is ended using mouse button 1. |
| VK_BUTTON2 | Drag is ended using mouse button 2. |
| VK_BUTTON3 | Drag is ended using mouse button 3. |
| VK_ENDDRAG | Drag is ended by the mouse button defined in the "System Setup" folder to end a drag. This should be used when dragging is performed in response to a WM_BEGINDRAG message. |

The DRAGIMAGE structure describes the image to be displayed as the object is being dragged. Since only the *DrgDrag* function needs to access this, and since the *DrgDrag* function executes in the context of the process calling it, this structure is not part of the DRAGITEM structure (although having it there would have made things slightly less complicated).

*DrgDrag* returns the window handle of the target window, if one is established. If the user pressed either the ESC key (to end the drag) or the F1 key (to get help for dropping on the current target), NULLHANDLE is returned, and the source is responsible for returning any shared resources consumed by

calling *DrgDeleteDraginfoStrHandles* to delete all of the string handles in the DRAGINFO structure, *DrgDeleteStrHandle* for each HSTR allocated that is not present in the DRAGINFO structure, and *DrgFreeDraginfo* to free the DRAGINFO structure. If this occurred frequently, nothing more would have to be discussed; instead we will assume that the user selected a target window and released the appropriate mouse button to initiate the transfer.

## And Now a Word from Our Sponsor

Since the data transfer actively involves both the source and target windows, now is a good time to view the target's perspective from the beginning. Remember that it is the target's responsibility to provide visual feedback to the user during the drag operation and to initiate the data transfer once the drop has occurred. Visual feedback is accomplished by responding to the appropriate DM_ messages that are sent to the target during the drag.

DM_DRAGOVER This message is sent whenever the pointer enters the target window space to allow it the opportunity to add *target emphasis* to the destination of the drag. This is also sent whenever a key is pressed or released. The message contains a pointer to the DRAGINFO structure, which can be accessed by calling *DrgAccessDraginfo*.

DM_DRAGLEAVE This message is sent to any window previously sent a DM_DRAGOVER message whenever the pointer leaves the target window space to allow it the opportunity to remove any "target emphasis" previously drawn. Note that since this occurs only for a window, the target is responsible for monitoring the mouse position of the DM_DRAGOVER messages when it is a container for other items. This message is not sent if the object(s) are dropped on the window.

DM_DROP This message is sent to the target window when the user drops the object(s) on it. As with DM_DRAGLEAVE, any target emphasis should be removed once this message is received. Normally this message is responded to before any data transfer takes place so that the source can learn the window handle of the target.

DM_DROPHELP This message is sent whenever the user presses F1 during a drag operation. The target should respond by displaying help on the actions that would occur if the object(s) were dropped at the point where F1 was pressed.

Whenever a DM_DRAGOVER message is received, the potential target must determine if the drag operation is valid. For example, a C source file could be dropped on a C compiler object, but not a Pascal source file; by holding down the CTRL key, a file could be copied to the printer, but it is (probably) unlikely that a file could be moved to the printer. At a minimum, the following two conditions must be met for a drop to be possible:

1. Both the source and target must understand at least one common type of each object being dragged.
2. Both the source and target must understand at least one common RMF for each object being dragged.

When determining the state of these conditions, the functions *DrgVerifyType*, *DrgVerifyRMF*, *DrgVerifyTrueType*, and *DrgVerifyNativeRMF* help considerably.

```
BOOL DrgVerifyType(PDRAGITEM pdiItem,PCHAR pchType);
BOOL DrgVerifyRMF(PDRAGITEM pdiItem,PCHAR pchMech,PCHAR pchFmt);
BOOL DrgVerifyTrueType(PDRAGITEM pdiItem,PCHAR pchType);
BOOL DrgVerifyNativeRMF(PDRAGITEM pdiItem,PCHAR pchRMF);
```

In all of these functions, *pdiItem* points to the DRAGITEM structure describing the item being tested. *pchType* specifies the type to compare with. *pchMech* specifies the rendering mechanism. *pchFmt* specifies the data format. *pchRMF* specifies a rendering mechanism and format. All of these functions return TRUE if the condition is met and FALSE if not.

The target responds to the DM_DRAGOVER message with a DOR_ constant.

Table 20.3 DOR_ Constants

| Constant | Description |
|---|---|
| DOR_DROP | Returned whenever the drag is acceptable. This is the only response that can be equated with a "Yes, you can drop here." |
| DOR_NODROP | Returned whenever the location of the object(s) in the target window is unacceptable. |
| DOR_NODROPOP | Returned whenever the operation (copy or move) is unacceptable; this implies that the drag might be valid if the operation is changed. |
| DOR_NEVERDROP | Returned whenever a drag is never acceptable; no further DM_DRAGOVER messages will be sent to the application until the mouse leaves the window and returns. |

**Gotcha!**
Although the DRAGINFO structure is allocated in shared memory and the pointer is passed to the target, the target cannot access the structure until *DrgAccessDraginfo* is called.

# Data Transfer

Okay, let's assume that the user selected one or more objects, depressed the appropriate mouse button, dragged the object(s) over a window, received the feedback that the target is willing to accept the object(s), and let go of the mouse button. What happens next? The answer to this depends on the RMF chosen to transfer the data with. For example, if DRM_OS2FILE is chosen, the target could choose to render the data itself, or maybe it doesn't know the name of the source data (e.g., for security reasons, the source window didn't fill this in), so it must ask the source window to render the data before it can complete the drop operation.

Let us consider each of the three system-defined rendering mechanisms to see the possible chain of events within each.

DRM_OS2FILE This mechanism would be used to transfer the data via the file system. The data does not have to exist already in this form, but could be placed there by the source after receiving a DM_RENDER message from the target.

If the target understands the native RMF and if the true type of the object is a file object, then the target can render the operation without the intervention of the source. However, this might not be feasible; in that

case, a DM_RENDER message would need to be sent to the source so that it can perform the operation. (This could occur if the source doesnot know the name of the file containing the data to be transferred.) If so, the target needs to allocate a DRAGTRANSFER structure (via *DrgAllocDragtransfer*) and fill in the *hstrRenderToName* field; the source sends back a DM_RENDERCOMPLETE message to indicate that the operation is done.

DRM_PRINT This mechanism would be used when the data is dropped onto a printer, and should be used only if the source understands and can process the DM_PRINT message that will be sent to it by the target. This message contains the name of the print queue to which the operation is to be performed.

**Gotcha!**
We have experienced trouble using the *pdriv* field of the *pdosData* field of the PRINTDEST structure passed in as a pointer in *mpParm2* for the DM_PRINTOBJECT message; the printer consistently rejects the data as being invalid when we call *DevOpenDC*. Unfortunately, one cannot simply call *DevPostDeviceModes* (see Chapter 25 for more information) to get a good set of driver data, because the device name is not specified anywhere. The workaround is to call *SplQueryQueue* first using the queue name in *pszLogAddress* field of the *pdosData* field of the PRINTDEST structure to get the PRQINFO3 structure containing the device name.

DRM_DDE This mechanism could be used when the other two do not provide the capability to complete the desired operation. While this is the most flexible of the three mechanisms, it is also the most cumbersome.

The source must understand and be able to process the appropriate WM_DDE_ messages sent to it by the target. Note that a WM_DDE_INITIATE is not required since the target already has the window handle with which it wishes to converse.

Since the topic of DDE could fill an entire chapter by itself, we will not present any more information on this type of data transfer in this chapter.

# A Concrete Example

A lot of material has been explained so far, and an example is sorely needed to cross the boundary from the abstract to the applied. The following application can act as both source and target for direct manipulation. While it is a simple program, it demonstrates the concepts previously described.

**DRAG1.C**

```
#define  INCL_DOSFILEMGR
#define  INCL_WININPUT
#define  INCL_WINPOINTERS
#define  INCL_WINSTDDRAG
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define  CLS_CLIENT     "SampleClass"
#define  DRAG_METHOD    "DRM_OS2FILE"
#define  DRAG_FORMAT    "DRF_UNKNOWN"
#define  DRAG_RMF       "<"DRAG_METHOD","DRAG_FORMAT">"
#define  DRAG_TEMPNAME  "DRAGDROP.TMP"
#define  MYM_DEWDROP    ( WM_USER )
```

```
typedef struct _CLIENTINFO
{
   PDRAGINFO        pdiDrag;

   BOOL             bDragging;
   BOOL             bEmphasis;
   CHAR             achLine[256];
} CLIENTINFO,*PCLIENTINFO;

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile);

VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo);

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Drag - n - Drop Sample"
                                     " Application",
                                  0,
                                  NULLHANDLE,
                                  0,
                                  NULL);

   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
                     0,
                     0);
   while (bLoop)
   {
```

```
      WinDispatchMsg(habAnchor,
                     &qmMsg);
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
   }                                    /* endwhile              */
   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return (0);
}

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile)
{
   CHAR              achPath[CCHMAXPATH];
   CHAR              achFile[CCHMAXPATH];

   DrgQueryStrName(pdiItem->hstrContainerName,
                   sizeof(achPath),
                   achPath);

   DosQueryPathInfo(achPath,
                    FIL_QUERYFULLNAME,
                    achPath,
                    sizeof(achPath));

   if (achPath[strlen(achPath)-1] != '\\')
   {
      strcat(achPath,
             "\\");
   }                                    /* endif                 */
   DrgQueryStrName(pdiItem->hstrSourceName,
                   sizeof(achFile),
                   achFile);

   sprintf(pchFile,
           "%s%s",
           achPath,
           achFile);
   return ;
}

VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo)
//--------------------------------------------------------------------
// This function paints the client window according its current
// status.
//
// Input:  hpsClient - handle to the presentation space.
//         hwndClient - handle to the client window.
//         pciInfo - pointer to the CLIENTINFO structure
//          describing the client window.
//--------------------------------------------------------------------

{
   RECTL             rclWindow;
   USHORT            usIndex;
   POINTL            ptlPaint;

   WinQueryWindowRect(hwndClient,
                      &rclWindow);
   WinFillRect(hpsClient,
               &rclWindow,
               SYSCLR_WINDOW);
```

```
//-------------------------------------------------------------
// Draw the dividing line
//-------------------------------------------------------------

ptlPaint.x = rclWindow.xRight/2;
ptlPaint.y = rclWindow.yBottom;
GpiMove(hpsClient,
        &ptlPaint);

ptlPaint.y = rclWindow.yTop;
GpiLine(hpsClient,
        &ptlPaint);
//-------------------------------------------------------------
// Set the color to indicate emphasized or
// non-emphasized state
//-------------------------------------------------------------

if (pciInfo->bEmphasis)
{
   GpiSetColor(hpsClient,
               CLR_BLACK);
}
else
{
   GpiSetColor(hpsClient,
               SYSCLR_WINDOW);
}                                   /* endif                */
//-------------------------------------------------------------
// Draw/erase the emphasis
//-------------------------------------------------------------

for (usIndex = 1; usIndex < 5; usIndex++)
{
   ptlPaint.x = rclWindow.xRight/2+usIndex;
   ptlPaint.y = rclWindow.yBottom+usIndex;
   GpiMove(hpsClient,
           &ptlPaint);

   ptlPaint.x = rclWindow.xRight-usIndex;
   ptlPaint.y = rclWindow.yTop-usIndex;
   GpiBox(hpsClient,
          DRO_OUTLINE,
          &ptlPaint,
          0,
          0);
}                                   /* endfor               */
//-------------------------------------------------------------
// Draw the instructing text
//-------------------------------------------------------------

WinQueryWindowRect(hwndClient,
                   &rclWindow);
rclWindow.xRight /= 2;

WinDrawText(hpsClient,
            -1,
            "Drag me",
            &rclWindow,
            CLR_BLACK,
            0,
            DT_CENTER|DT_VCENTER);

if (pciInfo->achLine[0] != 0)
{
```

```
      //---------------------------------------------------------
      // Draw the text received if we've been dropped on
      //---------------------------------------------------------
      WinQueryWindowRect(hwndClient,
                         &rclWindow);

      rclWindow.xLeft = rclWindow.xRight/2;

      WinDrawText(hpsClient,
                  -1,
                  pciInfo->achLine,
                  &rclWindow,
                  CLR_BLACK,
                  0,
                  DT_CENTER|DT_VCENTER);
   }                                    /* endif                */
   return ;
}

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//---------------------------------------------------------------
// This function handles the direct-manipulation messages
//
// sent to the source part of the window.
//
// Input, output, returns:  as a window procedure
//---------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {

      case  WM_BEGINDRAG :
         {
            RECTL            rclWindow;
            FILE             *pfFile;
            DRAGITEM         diItem;
            DRAGIMAGE        diImage;
            HWND             hwndTarget;

            WinQueryWindowRect(hwndClient,
                               &rclWindow);
         //---------------------------------------------------
         // If we're in the right half, return
         //---------------------------------------------------

            if (SHORT1FROMMP(mpParm1) > rclWindow.xRight/2)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */
         //---------------------------------------------------
         // Write the text to be dragged to a file,
         // since the type is
         // DRT_OS2FILE.
         //---------------------------------------------------

            pfFile = fopen(DRAG_TEMPNAME,
                           "w");
            if (pfFile == NULL)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */
```

```
      fprintf(pfFile,
              "Dropped text");
      fclose(pfFile);
   //-----------------------------------------------------
   // Allocate the DRAGITEM/DRAGINFO structures and
   // initialize them
   //-----------------------------------------------------

      pciInfo->pdiDrag = DrgAllocDraginfo(1);
      pciInfo->pdiDrag->hwndSource = hwndClient;

      diItem.hwndItem = hwndClient;
      diItem.ulItemID = 1;
      diItem.hstrType = DrgAddStrHandle(DRT_TEXT);
      diItem.hstrRMF = DrgAddStrHandle(DRAG_RMF);
      diItem.hstrContainerName = DrgAddStrHandle(".");
      diItem.hstrSourceName = DrgAddStrHandle(DRAG_TEMPNAME
         );

      diItem.hstrTargetName = diItem.hstrSourceName;
      diItem.cxOffset = 0;
      diItem.cyOffset = 0;
      diItem.fsControl = 0;
      diItem.fsSupportedOps = DO_MOVEABLE;

      DrgSetDragitem(pciInfo->pdiDrag,
                     &diItem,
                     sizeof(diItem),
                     0);
   //-----------------------------------------------------
   // Initialize the DRAGIMAGE structure
   //-----------------------------------------------------

      diImage.cb = sizeof(diImage);
      diImage.cptl = 0;
      diImage.hImage = WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FILE,
                                          FALSE);
      diImage.sizlStretch.cx = 0;
      diImage.sizlStretch.cy = 0;
      diImage.fl = DRG_ICON;
      diImage.cxOffset = 0;
      diImage.cyOffset = 0;
   //-----------------------------------------------------
   // Set the bDragging flag and call DrgDrag().
   //-----------------------------------------------------

      pciInfo->bDragging = TRUE;

      hwndTarget = DrgDrag(hwndClient,
                           pciInfo->pdiDrag,
                           &diImage,
                           1L,
                           VK_ENDDRAG,
                           NULL);

      if (hwndTarget == NULLHANDLE)
      {

         DrgDeleteDraginfoStrHandles(pciInfo->pdiDrag);
         DrgFreeDraginfo(pciInfo->pdiDrag);
         pciInfo->pdiDrag = NULL;
         pciInfo->bDragging = FALSE;
         remove(DRAG_TEMPNAME);
```

```
            WinInvalidateRect(hwndClient,
                              NULL,
                              FALSE);
            WinUpdateWindow(hwndClient);
         }                              /* endif                */
      }
      break;

   case  DM_ENDCONVERSATION :
      {
         PDRAGITEM        pdiItem;
         CHAR             achFullFile[CCHMAXPATH];
      //-------------------------------------------------------
      // Query the item dragged for cleanup purposes
      //-------------------------------------------------------

         pdiItem = DrgQueryDragitemPtr(pciInfo->pdiDrag,
                                       0);
      //-------------------------------------------------------
      // Delete the file used to transfer the data
      //-------------------------------------------------------

         fileFromDragItem(pdiItem,
                          achFullFile);
         remove(achFullFile);
      //-------------------------------------------------------
      // Cleanup
      //-------------------------------------------------------

         DrgDeleteDraginfoStrHandles(pciInfo->pdiDrag);
         DrgFreeDraginfo(pciInfo->pdiDrag);
         pciInfo->pdiDrag = NULL;
         pciInfo->bDragging = FALSE;
      }
      break;

   default  :
      break;
   }                                  /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//------------------------------------------------------------
// This function handles the direct-manipulation messages
// sent to the target part of the window.
//
// Input, output, returns:  as a window procedure
//------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {
      case  DM_DRAGOVER :
         {
            PDRAGINFO        pdiDrag;
            POINTL           ptlDrop;
            RECTL            rclWindow;
            HPS              hpsWindow;
            PDRAGITEM        pdiItem;
```

```
//-----------------------------------------------------
// Get the pointer to the DRAGINFO structure
//-----------------------------------------------------

   pdiDrag = (PDRAGINFO)PVOIDFROMMP(mpParm1);
   DrgAccessDraginfo(pdiDrag);
//-----------------------------------------------------
// Since the drop position is in screen coordinates,
// map them to our window and check if it is
// in the right half of the window.  If not, return.
//-----------------------------------------------------

   ptlDrop.x = SHORT1FROMMP(mpParm2);
   ptlDrop.y = SHORT2FROMMP(mpParm2);

   WinMapWindowPoints(HWND_DESKTOP,
                      hwndClient,
                      &ptlDrop,
                      1);

   WinQueryWindowRect(hwndClient,
                      &rclWindow);

   if (ptlDrop.x < rclWindow.xRight/2)
   {
      if (pciInfo->bEmphasis)
      {
         pciInfo->bEmphasis = FALSE;

         hpsWindow = DrgGetPS(hwndClient);
         paintClient(hpsWindow,
                     hwndClient,
                     pciInfo);
         DrgReleasePS(hpsWindow);
      }                          /* endif                */
      return  MRFROMSHORT(DOR_NODROP);
   }                             /* endif                */
//-----------------------------------------------------
// Check to see if we've already turned emphasis on.
//-----------------------------------------------------

   if (!pciInfo->bEmphasis)
   {
      pciInfo->bEmphasis = TRUE;

      hpsWindow = DrgGetPS(hwndClient);
      paintClient(hpsWindow,
                  hwndClient,
                  pciInfo);
      DrgReleasePS(hpsWindow);
   }                             /* endif                */
//-----------------------------------------------------
// We should only be dragging one item.
//-----------------------------------------------------

   if (DrgQueryDragitemCount(pdiDrag) != 1)
   {
      return  MRFROMSHORT(DOR_NODROP);
   }                             /* endif                */
//-----------------------------------------------------
// Check the true type and native RMF
//-----------------------------------------------------

   pdiItem = DrgQueryDragitemPtr(pdiDrag,
                                 0);
```

```
         if (!DrgVerifyTrueType(pdiItem,
                                DRT_TEXT))
         {
            return  MRFROMSHORT(DOR_NODROP);
         }
         else

            if (!DrgVerifyNativeRMF(pdiItem,
                                    DRAG_RMF))
            {
               return  MRFROMSHORT(DOR_NODROP);
            }                               /* endif                */
         return  MRFROM2SHORT(DOR_DROP,
                              DO_MOVE);
      }
   case  DM_DRAGLEAVE :

      if (pciInfo->bEmphasis)
      {
         HPS              hpsWindow;
      //-------------------------------------------------------
      // Turn off the emphasis
      //-------------------------------------------------------

         pciInfo->bEmphasis = FALSE;

         hpsWindow = DrgGetPS(hwndClient);
         paintClient(hpsWindow,
                     hwndClient,
                     pciInfo);
         DrgReleasePS(hpsWindow);
      }                                  /* endif                */
      break;

   case  DM_DROP :
      WinPostMsg(hwndClient,
                 MYM_DEWDROP,
                 mpParm1,
                 mpParm2);
      break;

   case  DM_DROPHELP :
      WinMessageBox(HWND_DESKTOP,
                    hwndClient,
                    "This is the drag - n - drop help."
                       "  Great, isn't it?",
                    "Help",
                    0,
                    MB_INFORMATION|MB_OK);
      break;

   case  MYM_DEWDROP :
      {
         PDRAGINFO        pdiDrag;
         PDRAGITEM        pdiItem;
         CHAR             achFullFile[CCHMAXPATH];
         FILE             *pfFile;
         HPS              hpsDrop;
      //-------------------------------------------------------
      // Get the pointer to the DRAGINFO structure
      //-------------------------------------------------------

         pdiDrag = (PDRAGINFO)PVOIDFROMMP(mpParm1);
         DrgAccessDraginfo(pdiDrag);
```

```
         //-------------------------------------------------------
         // Since we can render the object ourselves,
         // get the filename and read in the line
         //-------------------------------------------------------

            pdiItem = DrgQueryDragitemPtr(pdiDrag,
                                          0);
            fileFromDragItem(pdiItem,
                             achFullFile);

            pfFile = fopen(achFullFile,
                           "r");
            if (pfFile != NULL)
            {
               fgets(pciInfo->achLine,
                     sizeof(pciInfo->achLine),
                     pfFile);
               fclose(pfFile);
            }                              /* endif                */
         //-------------------------------------------------------
         // Turn the emphasis off and repaint ourselves to
         // show the dropped text
         //-------------------------------------------------------

            pciInfo->bEmphasis = FALSE;

            hpsDrop = WinGetPS(hwndClient);
            paintClient(hpsDrop,
                        hwndClient,
                        pciInfo);
            WinReleasePS(hpsDrop);
         //-------------------------------------------------------
         // Tell the source that we're done with the object
         //-------------------------------------------------------

            WinSendMsg(pdiDrag->hwndSource,
                       DM_ENDCONVERSATION,
                       MPFROMLONG(pdiItem->ulItemID),
                       MPFROMLONG(DMFL_TARGETSUCCESSFUL));
         //-------------------------------------------------------
         // Cleanup
         //-------------------------------------------------------

            DrgFreeDraginfo(pdiDrag);
         }
         break;

      default  :
         break;

   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
//-------------------------------------------------------------
// This function handles the messages sent to the client window.
//
// Input, output, returns:  as a window procedure
//-------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);
```

```
switch (ulMsg)
{
   case  WM_CREATE :
   //-------------------------------------------------------
   // Allocate memory for the instance data structure
   // and initialize it
   //-------------------------------------------------------

      pciInfo = malloc(sizeof(CLIENTINFO));
      if (pciInfo == NULL)
      {
         WinMessageBox(HWND_DESKTOP,
                       hwndClient,
                       "There is not enough memory.",
                       "Error",
                       0,
                       MB_ICONHAND|MB_OK);
         return  MRFROMSHORT(TRUE);
      }                              /* endif                */

      WinSetWindowPtr(hwndClient,
                      0,
                      pciInfo);

      pciInfo->bDragging = FALSE;
      pciInfo->bEmphasis = FALSE;
      pciInfo->achLine[0] = 0;
      break;

   case  WM_DESTROY :
   //-------------------------------------------------------
   // Free the memory used by the instance data
   //-------------------------------------------------------

      free(pciInfo);
      WinSetWindowPtr(hwndClient,
                      0,
                      NULL);
      break;

   case  WM_PAINT :
      {
         HPS              hpsPaint;

         hpsPaint = WinBeginPaint(hwndClient,
                                  NULLHANDLE,
                                  NULL);

         paintClient(hpsPaint,
                     hwndClient,
                     pciInfo);
         WinEndPaint(hpsPaint);
      }
      break;

   case  WM_BEGINDRAG :
   case  DM_ENDCONVERSATION :
      return  doSource(hwndClient,
                       ulMsg,
                       mpParm1,
                       mpParm2);

   case  DM_DRAGOVER :
   case  DM_DRAGLEAVE :
   case  DM_DROP :
   case  DM_DROPHELP :
   case  MYM_DEWDROP :
```

```
        return  doTarget(hwndClient,
                         ulMsg,
                         mpParm1,
                         mpParm2);

      default   :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                    /* endswitch            */
   return  MRFROMSHORT(FALSE);
}
```

**DRAG1.MAK**

```
DRAG1.EXE:                      DRAG1.OBJ
        LINK386 @<<
DRAG1
DRAG1
DRAG1
OS2386
DRAG1
<<

DRAG1.OBJ:                      DRAG1.C
        ICC -C+ -Kb+ -Ss+ DRAG1.C
```

**DRAG1.DEF**

```
NAME DRAG1 WINDOWAPI

DESCRIPTION 'Drag example 1
Copyright (c) 1992 by Larry Salomon
All rights reserved.'

STACKSIZE 16384
```

Since *main* is fairly standard, we'll ignore it except for the fact that we're reserving space for a pointer in the call to *WinRegisterClass*. This will be used to store a pointer to the client's instance data, so that we can avoid global variables. This instance data is allocated and initialized in the WM_CREATE message and is freed in the WM_DESTROY message.

```
typedef struct _CLIENTINFO {
   PDRAGINFO pdiDrag;
   BOOL bDragging;
   BOOL bEmphasis;
   CHAR achLine[256];
} CLIENTINFO, *PCLIENTINFO;
```

The *pdiDrag* field is used only by the source window and points to the DRAGINFO structure allocated via *DrgAllocDraginfo*. *bDragging* and *bEmphasis* specify whether a dragging operation is in progress and whether the client is displaying emphasis, respectively. *achLine* is used only by the target window and contains the line of text that was dropped on the window. For clarity, the processing of the direct-manipulation messages has been separated into those usually associated with the source and target windows. (See *doSource* and *doTarget*.)

What the program does is allow the dragging of text from the left half of the window into either the right half of this window or another instance of this window. (Try starting two copies of DRAG1.EXE to do this.) Whenever the source receives a WM_BEGINDRAG message, the appropriate data structures are initialized and *DrgDrag* is called. The target adds emphasis whenever it receives a DM_DRAGOVER message and returns the appropriate DOR_ value. After the object has been dropped, the target completely renders the data provided by the source and sends the source a DM_ENDCONVERSATION message to terminate the dragging operation.

Readers probably are wondering why we return DOR_NODROP from the DM_DRAGOVER message when we find that we cannot accept the drop because the objects are in an unrecognized type or use an unrecognized RMF. It is true that normally DOR_NEVERDROP would be returned, but it must be remembered that we allow dropping only on the right half of the window; once the pointer moves into the left half, we must remove the target emphasis. However, if we return DOR_NEVERDROP, we never receive another DM_DRAGOVER message until the mouse moves out of the window and then back into the window. This technique is required for *container* windows (where *container* is a concept and does not specify the WC_CONTAINER window class) when the potential targets are not child windows.

**Gotcha!**
It needs to be stated somewhere, and what a better place than here, that there appears to be a bug in OS/2 Warp when using DRG_BITMAP for the DRAGIMAGE to be displayed. The first time the drag and drop is performed, everything works fine; but if the application is exited and restarted, dragging the object using DRG_BITMAP leaves "mouse droppings" behind, making the display quite ugly. We have no information regarding the availability of a fix.

**Gotcha!**
Another important item is that the *cxOffset* and *cyOffset* fields of the DRAGITEM structure cannot be used for the programmer's own purposes, since *DrgDrag* copies the corresponding fields from the DRAGIMAGE structure here. Likewise, *hwndItem* should specify a valid window handle, or unexpected results will occur. Any associated structures that need to be "attached" to a DRAGITEM structure may do so safely by casting the structure to a ULONG and assigning the pointer to the *ulItemID* field.

## More Cement, Please

Let's complicate things by modifying our program to have the source window render the data.

**DRAG2.C**

```
#define  INCL_DOSFILEMGR
#define  INCL_WININPUT
#define  INCL_WINPOINTERS
#define  INCL_WINSTDDRAG
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define  CLS_CLIENT     "SampleClass"
#define  DRAG_METHOD    "DRM_OS2FILE"
#define  DRAG_FORMAT    "DRF_UNKNOWN"
```

```
#define  DRAG_RMF       "<"DRAG_METHOD","DRAG_FORMAT">"
#define  DRAG_TEMPNAME "DRAGDROP.TMP"
#define  MYM_DEWDROP    ( WM_USER )
typedef struct _CLIENTINFO
{
   PDRAGINFO        pdiSource;         // Used by source

   BOOL             bDragging;         // Used by source
   BOOL             bEmphasis;         // Used by source
   PDRAGINFO        pdiTarget;         // Used by source
   CHAR             achLine[256];      // Used by target
} CLIENTINFO,*PCLIENTINFO;

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile);
VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo);

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Drag - n - Drop"
                                     " Sample Application 2",
                                  0,
                                  NULLHANDLE,
                                  0,
                                  NULL);

   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
                     0,
                     0);
   while (bLoop)
   {
```

```
      WinDispatchMsg(habAnchor,
                     &qmMsg);
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
   }                                  /* endwhile              */
   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return (0);
}

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile)
//------------------------------------------------------------
// This function composes a filename from the DRAGITEM structure
//  and returns it in pchFile.  It is assumed that pchFile
//  points to a buffer of size CCHMAXPATH.
//
//
// Input:  pdiItem - points to the DRAGITEM structure containing
//                   the necessary information.
// Output:  pchFile - points to the variable containing the
//                    filename.
//------------------------------------------------------------

{
   CHAR             achPath[CCHMAXPATH];
   CHAR             achFile[CCHMAXPATH];

   DrgQueryStrName(pdiItem->hstrContainerName,
                   sizeof(achPath),
                   achPath);

   DosQueryPathInfo(achPath,
                    FIL_QUERYFULLNAME,
                    achPath,
                    sizeof(achPath));

   if (achPath[strlen(achPath)-1] != '\\')
   {
      strcat(achPath,
             "\\");
   }                                  /* endif                 */
   DrgQueryStrName(pdiItem->hstrSourceName,
                   sizeof(achFile),
                   achFile);

   sprintf(pchFile,
           "%s%s",
           achPath,
           achFile);
   return ;
}

VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo)
//------------------------------------------------------------
// This function paints the client window according
//    its current status.
//
// Input:  hpsClient - handle to the presentation space.
//         hwndClient - handle to the client window.
//         pciInfo - pointer to the CLIENTINFO structure
//                   describing the client window.
//------------------------------------------------------------

{
```

```
RECTL           rclWindow;
USHORT          usIndex;
POINTL          ptlPaint;

WinQueryWindowRect(hwndClient,
                   &rclWindow);
WinFillRect(hpsClient,
            &rclWindow,
            SYSCLR_WINDOW);
//-------------------------------------------------------------
// Draw the dividing line
//-------------------------------------------------------------

ptlPaint.x = rclWindow.xRight/2;
ptlPaint.y = rclWindow.yBottom;
GpiMove(hpsClient,
        &ptlPaint);

ptlPaint.y = rclWindow.yTop;
GpiLine(hpsClient,
        &ptlPaint);
//-------------------------------------------------------------
// Set the color to indicate emphasized or
// non-emphasized state
//-------------------------------------------------------------

if (pciInfo->bEmphasis)
{
   GpiSetColor(hpsClient,
               CLR_BLACK);
}
else
{
   GpiSetColor(hpsClient,
               SYSCLR_WINDOW);
}                                   /* endif                */
//-------------------------------------------------------------
// Draw/erase the emphasis
//-------------------------------------------------------------

for (usIndex = 1; usIndex < 5; usIndex++)
{
   ptlPaint.x = rclWindow.xRight/2+usIndex;
   ptlPaint.y = rclWindow.yBottom+usIndex;
   GpiMove(hpsClient,
           &ptlPaint);

   ptlPaint.x = rclWindow.xRight-usIndex;
   ptlPaint.y = rclWindow.yTop-usIndex;
   GpiBox(hpsClient,
          DRO_OUTLINE,
          &ptlPaint,
          0,
          0);
}                                   /* endfor               */
//-------------------------------------------------------------
// Draw the instructing text
//-------------------------------------------------------------

WinQueryWindowRect(hwndClient,
                   &rclWindow);
rclWindow.xRight /= 2;
```

```
   WinDrawText(hpsClient,
               -1,
               "Drag me",
               &rclWindow,
               CLR_BLACK,
               0L,
               DT_CENTER|DT_VCENTER);

   if (pciInfo->achLine[0] != 0)
   {

      //-------------------------------------------------------
      // Draw the text received if we've been dropped on
      //-------------------------------------------------------
      WinQueryWindowRect(hwndClient,
                         &rclWindow);

      rclWindow.xLeft = rclWindow.xRight/2;

      WinDrawText(hpsClient,
                  -1,
                  pciInfo->achLine,
                  &rclWindow,
                  CLR_BLACK,
                  0L,
                  DT_CENTER|DT_VCENTER);
   }                                  /* endif                */
   return ;
}

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//------------------------------------------------------------
// This function handles the direct-manipulation messages
// sent to the source part of the window.
//
// Input, output, returns:  as a window procedure
//------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {

      case  WM_BEGINDRAG :
         {
            RECTL             rclWindow;
            DRAGITEM          diItem;
            DRAGIMAGE         diImage;
            HWND              hwndTarget;

            WinQueryWindowRect(hwndClient,
                               &rclWindow);
         //-------------------------------------------------------
         // If we're in the right half, return
         //-------------------------------------------------------

            if (SHORT1FROMMP(mpParm1) > rclWindow.xRight/2)
            {

               return  MRFROMSHORT(FALSE);
            }                             /* endif                */
```

```
        //------------------------------------------------------
        // Allocate the DRAGITEM/DRAGINFO structures and
        // initialize them
        //------------------------------------------------------

           pciInfo->pdiSource = DrgAllocDraginfo(1);
           pciInfo->pdiSource->hwndSource = hwndClient;

           diItem.hwndItem = hwndClient;
           diItem.ulItemID = 1;
           diItem.hstrType = DrgAddStrHandle(DRT_TEXT);
           diItem.hstrRMF = DrgAddStrHandle(DRAG_RMF);
           diItem.hstrContainerName = DrgAddStrHandle(".");
           diItem.hstrSourceName = NULLHANDLE;
           diItem.hstrTargetName = NULLHANDLE;
           diItem.cxOffset = 0;
           diItem.cyOffset = 0;
           diItem.fsControl = 0;
           diItem.fsSupportedOps = DO_MOVEABLE;

           DrgSetDragitem(pciInfo->pdiSource,
                          &diItem,
                          sizeof(diItem),
                          0);
        //------------------------------------------------------
        // Initialize the DRAGIMAGE structure
        //------------------------------------------------------

           diImage.cb = sizeof(diImage);
           diImage.cptl = 0;
           diImage.hImage = WinQuerySysPointer(HWND_DESKTOP,
                                               SPTR_FILE,
                                               FALSE);
           diImage.sizlStretch.cx = 0;
           diImage.sizlStretch.cy = 0;
           diImage.fl = DRG_ICON|DRG_TRANSPARENT;
           diImage.cxOffset = 0;
           diImage.cyOffset = 0;
        //------------------------------------------------------
        // Set the bDragging flag and call DrgDrag().
        //------------------------------------------------------

           pciInfo->bDragging = TRUE;

           hwndTarget = DrgDrag(hwndClient,
                                pciInfo->pdiSource,
                                &diImage,
                                1L,
                                VK_ENDDRAG,
                                NULL);
           if (hwndTarget == NULLHANDLE)
           {
              DrgDeleteDraginfoStrHandles(pciInfo->pdiSource);
              DrgFreeDraginfo(pciInfo->pdiSource);
              pciInfo->pdiSource = NULL;
              pciInfo->bDragging = FALSE;
              remove(DRAG_TEMPNAME);

              WinInvalidateRect(hwndClient,
                                NULL,
                                FALSE);
              WinUpdateWindow(hwndClient);
           }                            /* endif                */
        }
        break;
```

```
      case  DM_RENDERPREPARE :
         return  MRFROMSHORT(TRUE);

      case  DM_RENDER :
         {
            PDRAGTRANSFER    pdtXfer;
            CHAR             achFile[CCHMAXPATH];
            FILE             *pfFile;

            pdtXfer = (PDRAGTRANSFER)PVOIDFROMMP(mpParm1);

            DrgQueryStrName(pdtXfer->hstrRenderToName,
                            sizeof(achFile),
                            achFile);
            DrgFreeDragtransfer((PDRAGTRANSFER)PVOIDFROMMP
               (mpParm1));
         //-------------------------------------------------------
         // Write the text to be dragged to a file, since the
         // type is DRT_OS2FILE.
         //-------------------------------------------------------

            pfFile = fopen(achFile,
                           "w");
            if (pfFile != NULL)
            {
               fprintf(pfFile,
                       "Dropped text");
               fclose(pfFile);

               return  MRFROMSHORT(TRUE);
            }

            else
            {
               return  MRFROMSHORT(FALSE);
            }                              /* endif                */
         }

      case  DM_ENDCONVERSATION :
      //----------------------------------------------------------
      // Cleanup
      //----------------------------------------------------------

         DrgDeleteDraginfoStrHandles(pciInfo->pdiSource);
         DrgFreeDraginfo(pciInfo->pdiSource);
         pciInfo->pdiSource = NULL;
         pciInfo->bDragging = FALSE;
         break;

      default  :
         break;
   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//------------------------------------------------------------------
// This function handles the direct-manipulation messages
// sent to the target part of the window.
//
// Input, output, returns:  as a window procedure
//------------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);
```

```
switch (ulMsg)
{
   case  DM_DRAGOVER :
      {
         PDRAGINFO        pdiDrag;
         POINTL           ptlDrop;
         RECTL            rclWindow;
         HPS              hpsWindow;
         PDRAGITEM        pdiItem;
      //-------------------------------------------------------
      // Get the pointer to the DRAGINFO structure
      //-------------------------------------------------------

         pdiDrag = (PDRAGINFO)PVOIDFROMMP(mpParm1);
         DrgAccessDraginfo(pdiDrag);
      //-------------------------------------------------------
      // Since the drop position is in screen coordinates,
      // map them to our window and check if it is in the
      // right half of the window.  If not, return.
      //-------------------------------------------------------

         ptlDrop.x = SHORT1FROMMP(mpParm2);
         ptlDrop.y = SHORT2FROMMP(mpParm2);
         WinMapWindowPoints(HWND_DESKTOP,
                            hwndClient,
                            &ptlDrop,
                            1);

         WinQueryWindowRect(hwndClient,
                            &rclWindow);

         if (ptlDrop.x < rclWindow.xRight/2)
         {
            if (pciInfo->bEmphasis)
            {
               pciInfo->bEmphasis = FALSE;

               hpsWindow = DrgGetPS(hwndClient);
               paintClient(hpsWindow,
                           hwndClient,
                           pciInfo);
               DrgReleasePS(hpsWindow);
            }                          /* endif                */
            return  MRFROMSHORT(DOR_NODROP);
         }                             /* endif                */
      //-------------------------------------------------------
      // Check to see if we've already turned emphasis on.
      //-------------------------------------------------------

         if (!pciInfo->bEmphasis)
         {

            pciInfo->bEmphasis = TRUE;

            hpsWindow = DrgGetPS(hwndClient);
            paintClient(hpsWindow,
                        hwndClient,
                        pciInfo);
            DrgReleasePS(hpsWindow);

         }                             /* endif                */
      //-------------------------------------------------------
      // We should only be dragging one item.
      //-------------------------------------------------------

         if (DrgQueryDragitemCount(pdiDrag) != 1)
```

```
         {
            return  MRFROMSHORT(DOR_NODROP);
         }                              /* endif                */
      //-------------------------------------------------------
      // Check the true type and native RMF
      //-------------------------------------------------------

         pdiItem = DrgQueryDragitemPtr(pdiDrag,
                                       0);

         if (!DrgVerifyTrueType(pdiItem,
                                DRT_TEXT))
         {
            return  MRFROMSHORT(DOR_NODROP);
         }
         else

            if (!DrgVerifyNativeRMF(pdiItem,
                                    DRAG_RMF))
            {
               return  MRFROMSHORT(DOR_NODROP);
            }                           /* endif                */
         return  MRFROM2SHORT(DOR_DROP,
                              DO_MOVE);
      }

   case  DM_DRAGLEAVE :

      if (pciInfo->bEmphasis)
      {
         HPS              hpsWindow;
      //-------------------------------------------------------
      // Turn off the emphasis
      //-------------------------------------------------------

         pciInfo->bEmphasis = FALSE;

         hpsWindow = DrgGetPS(hwndClient);
         paintClient(hpsWindow,
                     hwndClient,
                     pciInfo);
         DrgReleasePS(hpsWindow);
      }                                 /* endif                */
      break;

   case  DM_DROP :
      WinPostMsg(hwndClient,
                 MYM_DEWDROP,
                 mpParm1,
                 mpParm2);
      break;

   case  DM_RENDERCOMPLETE :
      break;

   case  DM_DROPHELP :
      WinMessageBox(HWND_DESKTOP,
                    hwndClient,
                    "This is the drag - n - drop help."
                       "  Great, isn't it?",
                    "Help",
                    0,
                    MB_INFORMATION|MB_OK);
      break;
```

```
case  MYM_DEWDROP :
   {
      HPS               hpsDrop;
      PDRAGITEM         pdiItem;
      CHAR              achRMF[256];
      PDRAGTRANSFER     pdtXfer;
      CHAR              achFile[CCHMAXPATH];
      FILE              *pfFile;
   //------------------------------------------------------
   // Get the pointer to the DRAGINFO structure
   //------------------------------------------------------

      pciInfo->pdiTarget = (PDRAGINFO)PVOIDFROMMP(mpParm1);

      DrgAccessDraginfo(pciInfo->pdiTarget);
   //------------------------------------------------------
   // Turn the emphasis off
   //------------------------------------------------------

      pciInfo->bEmphasis = FALSE;

      hpsDrop = WinGetPS(hwndClient);
      paintClient(hpsDrop,
                  hwndClient,
                  pciInfo);
      WinReleasePS(hpsDrop);
   //------------------------------------------------------
   // If the source did not render the data previously,
   // we need to allocate a DRAGTRANSFER structure and
   // send the source a DM_RENDER message.
   //------------------------------------------------------

      pdiItem = DrgQueryDragitemPtr(pciInfo->pdiTarget,
                                    0);

      if (pdiItem->hstrSourceName == NULLHANDLE)
      {
         DrgQueryNativeRMF(pdiItem,
                           sizeof(achRMF),
                           achRMF);

         pdtXfer = DrgAllocDragtransfer(1);
         pdtXfer->cb = sizeof(DRAGTRANSFER);
         pdtXfer->hwndClient = hwndClient;
         pdtXfer->pditem = pdiItem;
         pdtXfer->hstrSelectedRMF = DrgAddStrHandle(achRMF)
            ;
         pdtXfer->hstrRenderToName = DrgAddStrHandle
            (DRAG_TEMPNAME);
         pdtXfer->ulTargetInfo = 0;
         pdtXfer->usOperation =
            pciInfo->pdiTarget->usOperation;

         pdtXfer->fsReply = 0;
      //------------------------------------------------
      // Does the source need to prepare the item?
      // If so, send a DM_RENDERPREPARE message.
      //------------------------------------------------

         if ((pdiItem->fsControl&DC_PREPARE) != 0)
         {

            if (!SHORT1FROMMR(DrgSendTransferMsg
               (pciInfo->pdiTarget->hwndSource,
                DM_RENDERPREPARE,
                MPFROMP(pdtXfer),
                0L)))
```

```
        {

           DrgDeleteStrHandle(pdtXfer->hstrSelectedRMF)
              ;

           DrgDeleteStrHandle(pdtXfer->hstrRenderToName
              );

           DrgFreeDragtransfer(pdtXfer);
           return  MRFROMSHORT(FALSE);

        }                       /* endif                */
     }                          /* endif                */
  //-------------------------------------------------
  // Render the object
  //-------------------------------------------------

     if (!SHORT1FROMMR(DrgSendTransferMsg
        (pciInfo->pdiTarget->hwndSource,
         DM_RENDER,
         MPFROMP(pdtXfer),
         0L)))
     {

        DrgDeleteStrHandle(pdtXfer->hstrSelectedRMF);
        DrgDeleteStrHandle(pdtXfer->hstrRenderToName);
        DrgFreeDragtransfer(pdtXfer);
        return  MRFROMSHORT(FALSE);

     }                          /* endif                */

     strcpy(achFile,
            DRAG_TEMPNAME);

  }
  else
  {

  //-------------------------------------------------
  // The source already rendered the object, so we
  // can simply read from the file.
  //-------------------------------------------------
     pdtXfer = NULL;

     fileFromDragItem(pdiItem,
                      achFile);
  }                             /* endif                */
  pfFile = fopen(achFile,
                 "r");
  if (pfFile != NULL)
  {
     fgets(pciInfo->achLine,
           sizeof(pciInfo->achLine),
           pfFile);
     fclose(pfFile);

     if (pdtXfer != NULL)
     {
        remove(achFile);
     }                          /* endif                */
  //-------------------------------------------------
  // Repaint ourselves again to show the dropped text
  //-------------------------------------------------

     hpsDrop = WinGetPS(hwndClient);
     paintClient(hpsDrop,
                 hwndClient,
                 pciInfo);
     WinReleasePS(hpsDrop);
```

```
            }                              /* endif                */
         //-------------------------------------------------------
         // Tell the source that we're done rendering
         //-------------------------------------------------------

            WinSendMsg(pciInfo->pdiTarget->hwndSource,
                       DM_ENDCONVERSATION,
                       MPFROMLONG(pdiItem->ulItemID),
                       MPFROMLONG(DMFL_TARGETSUCCESSFUL));
         //-------------------------------------------------------
         // Cleanup
         //-------------------------------------------------------

            if (pdtXfer != NULL)
            {
               DrgDeleteStrHandle(pdtXfer->hstrSelectedRMF);
               DrgDeleteStrHandle(pdtXfer->hstrRenderToName);
               DrgFreeDragtransfer(pdtXfer);
            }                              /* endif                */
            DrgFreeDraginfo(pciInfo->pdiTarget);
         }
         break;
      default  :
         break;
   }                                       /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {

      case  WM_CREATE :
      //-------------------------------------------------------
      // Allocate memory for the instance data structure
      // and initialize it
      //-------------------------------------------------------

         pciInfo = malloc(sizeof(CLIENTINFO));
         if (pciInfo == NULL)
         {
            WinMessageBox(HWND_DESKTOP,
                          hwndClient,
                          "There is not enough memory.",
                          "Error",
                          0,
                          MB_ICONHAND|MB_OK);
            return  MRFROMSHORT(TRUE);
         }                                 /* endif                */

         WinSetWindowPtr(hwndClient,
                         0,
                         pciInfo);

         pciInfo->pdiSource = NULL;
         pciInfo->bDragging = FALSE;
         pciInfo->bEmphasis = FALSE;
         pciInfo->pdiTarget = NULL;
         pciInfo->achLine[0] = 0;
```

```
        break;

      case  WM_DESTROY :
      //-------------------------------------------------------
      // Free the memory used by the instance data
      //-------------------------------------------------------

         free(pciInfo);
         WinSetWindowPtr(hwndClient,
                         0,
                         NULL);
         break;

      case  WM_PAINT :
         {
            HPS              hpsPaint;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     NULL);

            paintClient(hpsPaint,
                        hwndClient,
                        pciInfo);
            WinEndPaint(hpsPaint);
         }
         break;

      case  WM_BEGINDRAG :
      case  DM_RENDERPREPARE :
      case  DM_RENDER :
      case  DM_ENDCONVERSATION :
         return  doSource(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);

      case  DM_DRAGOVER :
      case  DM_DRAGLEAVE :
      case  DM_DROP :
      case  DM_RENDERCOMPLETE :
      case  DM_DROPHELP :
      case  MYM_DEWDROP :
         return  doTarget(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);

      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                  /* endswitch            */
   return  MRFROMSHORT(FALSE);
}
```

**DRAG2.MAK**

```
DRAG2.EXE:                      DRAG2.OBJ
        LINK386 @<<
DRAG2
DRAG2
DRAG2
OS2386
DRAG2
<<

DRAG2.OBJ:                      DRAG2.C
        ICC -C+ -Kb+ -Ss+ DRAG2.C
```

**DRAG2.DEF**

```
NAME DRAG2 WINDOWAPI

DESCRIPTION 'Drag example 2
Copyright (c) 1992 by Larry Salomon
All rights reserved.'

STACKSIZE 16384
```

As can be seen, the case when the source does not render the data prior to calling *DrgDrag* is a bit more involved. This is communicated to the target by not specifying the source name in *hstrSourceName*. After determining that this did not happen, the program allocates another shared structure—DRAGTRANSFER—using a call to *DrgAllocDragtransfer* and sends the source a DM_RENDER message with the target name in the DRAGTRANSFER structure.

```
PDRAGTRANSFER DrgAllocDragtransfer(ULONG cdxfer);
```

*cdxfer* specifies the number of structures to allocate and must be greater than 0. It returns a pointer to the array of structures allocated.

```
typedef struct _DRAGTRANSFER {
   ULONG cb;
   HWND hwndClient;
   PDRAGITEM pditem;
   HSTR hstrSelectedRMF;
   HSTR hstrRenderToName;
   ULONG ulTargetInfo;
   USHORT usOperation;
   USHORT fsReply;
} DRAGTRANSFER, *PDRAGTRANSFER;
```

*cb* is the size of the structure in bytes. *hwndClient* specifies the handle of the window on which the item was dropped. *pditem* points to the DRAGITEM structure within the DRAGINFO structure that was passed via the DM_DROP message representing the item of interest. *hstrSelectedRMF* specifies a string handle that describes the RMF to use when transferring the item. *hstrRenderToName* specifies a string handle that describes the name to be used when rendering the data. *ulTargetInfo* specifies any application-specific data that the target window wishes to communicate to the source. *usOperation* specifies the operation to use—for example, copy, move, or link. *fsReply* is filled in by the source window and specifies a DMFL_ constant. Table 20.4 lists the available constants.

Table 20.4 DMFL_ Constants

| Constant | Description |
|---|---|
| DMFL_NATIVERENDER | The source does not support rendering of the object. This should not be specified unless the source gives enough information for the target to perform the rendering. |
| DMFL_RENDERRETRY | The source does support rendering of the object, but not using the RMF specified. |

*hstrSelectedRMF* and *hstrRenderToName* must have been allocated using the *DrgAddStrHandle* function.

The obvious question here is why to use *DrgSendTransferMsg* instead of the old reliable *WinSendMsg*. The answer is that the DRAGTRANSFER structure, like the DRAGINFO structure, is allocated in shared memory but is not automatically accessible by the other process. The *DrgSendTransferMsg* ensures that the recipient of the message can access the DRAGTRANSFER message in addition to calling *WinSendMsg* on behalf of the source.

Resources must be freed via the appropriate *Drg* functions by both the source and target windows, except for of the two HSTR handles in the DRAGTRANSFER structure. The target window is responsible for freeing these handles.

## *DrgDragFiles*

For drag operations involving only files, a much simplified version of *DrgDrag* can be used: *DrgDragFiles*.

```
BOOL DrgDragFiles(HWND hwndSource,
                  PSZ *apszFiles,
                  PSZ *apszTypes,
                  PSZ *apszTargets,
                  ULONG ulNumFiles,
                  HPOINTER hptrDrag,
                  ULONG ulTerminateKey,
                  BOOL bRender,
                  ULONG ulReserved);
```

*hwndSource* is the handle of the window calling the function. *apszFiles*, *apszTypes*, and *apszTargets* are arrays of pointers to the filenames, file types, and target filenames, respectively. *ulNumFiles* specifies the number of pointers in the *apszFiles*, *apszTypes*, and *apszTargets* arrays. *hptrDrag* is the handle to the pointer to display while dragging. *ulTerminateKey* has the same meaning as in *DrgDrag*, discussed on page 315. *bRender* specifies whether the caller needs to render the files before the transfer can take place. If so, a DM_RENDERFILE message is sent for each file.

That's it! The system takes care of the rest, since files are the only allowed object type.

# From the Top Now

Table 20.5 details the chain of events from the beginning of the drag notification to the end of the data transfer.

Table 20.5 Steps in a Drag/Drop Operation

| Step | Source | Target |
|---|---|---|
| 1 | Receives a WM_BEGINDRAG message | |
| 2 | Allocates the DRAGINFO/DRAGITEM structures using *DrgAllocDraginfo* | |
| 3 | Creates the strings for the type and RMF using *DrgAddStrHandle*. | |
| 4 | Initializes the appropriate number of DRAGIMAGE structures. | |
| 5 | Calls *DrgDrag*. | |
| 6 | | Receives DM_DRAGOVER. |
| 7 | | Calls *DrgAccessDraginfo*. |
| 8 | | Decides if objects are acceptable (both type and RMF). |
| 9 | | Returns the appropriate DOR_ value; if not DOR_DROP, go to step 20. |
| 10 | | If the user presses F1, target receives a DM_DROPHELP; after providing help, go to step 20. |
| 11 | | If user presses ESC, go to step 20. |
| 12 | | User drops objects on target. |
| 13 | | If target can render the objects on its own, do so. Go to step 18. |
| 14 | | Allocates DRAGTRANSFER structures for each object (*DrgAllocDragtransfer*) and sends a DM_RENDER for each object (*DrgSendTransferMsg*). |
| 15 | Renders the object. | |
| 16 | | Copies the objects and deletes them from the source. |
| 17 | | Frees HSTRs for DRAGTRANSFER and DRAGTRANSFER structures (*DrgDeleteStrHandle* and *DrgFreeDragtransfer*). |
| 18 | | Frees HSTRs for DRAGINFO and DRAGINFO structure (*DrgDeleteDraginfoStrHandles* and *DrgFreeDragtransfer*). |
| 19 | | Sends source a DM_ENDCONVERSATION message. |
| 20 | Free HSTRs for DRAGINFO and DRAGINFO structure (*DrgDeleteDraginfoStrHandles* and *DrgFreeDragtransfer*). | |

# Pickup and Drop

OS/2 Warp introduced a new twist on the direct manipulation concept. Because drag and drop is a modal operation—meaning that nothing else can occur while a direct manipulation operation is in progress—it can

be limiting at times. What happens if you start to drag an object and then realize that the target window isn't open yet? You have to press Escape, find the target window and open it, then repeat the operation.

*Pickup and drop* alleviates the headaches caused in these situations by allowing the user to continue using the mouse in the normal fashion while the operation is in progress. Because of this characteristic, pickup and drop is often referred to as *lazy drag and drop.*

Obviously, there are some profound differences from the user's perspective between the modal and modeless versions of direct manipulation. And this means that there are differences in the coding of the two types; fortunately, IBM decided in its wisdom to minimize the impact of choosing one or the other (or both) in your application by changing as little as possible in the manner in which the modeless version is coded. The interface differences are listed here:

- The operation is initiated by holding down the Alt key in addition to using the direct manipulation mouse button.
- Instead of receiving a WM_BEGINDRAG message, the potential source window receives a WM_PICKUP message.
- Whereas in modal operation all objects to be dragged must be selected before beginning the operation, in pickup and drop, objects can be added to the *pickup set* dynamically. In OS/2 Warp, however, all objects must originate from the same source window.
- Because the mouse is still usable after the pickup is initiated, the operation can not be ended by releasing the mouse button like the modal operation is ended in this fashion. The only way to end a direct manipulation operation is to call the *DrgCancelLazyDrag* function. And since the user must communicate to the program that the operation is to be cancelled, the most common method of indicating this is through a menu item.
- Another change that is related to using the mouse is the use of the DRAGIMAGE structures. Since the operation is modeless, the pointer displayed is still subject to the *WinSetPointer* function (via the WM_MOUSEMOVE and WM_CONTROLPOINTER messages). Thus, instead of displaying the DRAGIMAGEs provided by the application initiating the operation, the mouse pointer is only slightly augmented to indicate that the operation is in progress. The DRAGIMAGEs structures are still passed to the *DrgLazyDrag* function for "compatibility" with the parameter list given to *DrgDrag* but they are not used.
- Because the user could request help for any subject during a lazy drag, the DM_DROPHELP message will not be sent during a lazy drag. Help can only be provided via a menu item, for example, and it is the programmer's responsibility to code this support explicitly.
- Because the operation can potentially take a long time to complete, *DrgLazyDrag* returns immediately and the source window is sent a DM_DROPNOTIFY message whenever the user "drops" the objects on a target window via some interface (e.g. menu item). This is probably the most significant change of which the programmer needs to be aware.

# Functions Used for Lazy Drag

In order to make the programmer's job easier, IBM provided many new functions specifically for use with lazy drag.

```
PDRAGINFO DrgReallocDraginfo(PDRAGINFO pdiOld,ULONG ulNumItems);
```

This function reallocates memory to hold a new number of DRAGITEM structures when additional items are to be added to the pickup item set. *pdiOld* points to the old DRAGINFO structure. *ulNumItems* specifies the new number of DRAGITEM structures to be contained by the new DRAGINFO structure.

This function returns a pointer to the new DRAGINFO structure and frees the memory pointed to by the old structure. Once this function is called, *DrgLazyDrag* must be called again to reinitiate the lazy drag operation.

```
PDRAGINFO DrgQueryDraginfoPtr(PDRAGINFO pdiReserved);
```

*pdiReserved* is reserved and must be NULL. This function returns a pointer to the DRAGINFO structure currently in use by a direct manipulation operation. *DrgQueryDragStatus* must be called to determine what type of operation is in progress, however. If NULL is returned, no operation is in progress.

```
PDRAGINFO DrgQueryDraginfoPtrFromDragitem(PDRAGITEM pdiItem);
```

*pdiItem* points to a DRAGITEM structure returned from *DrgQueryDragitemPtr*. This function returns a pointer to the DRAGINFO structure with which the DRAGITEM is associated.

```
PDRAGINFO DrgQueryDraginfoPtrFromHwnd(HWND hwndSource);
```

*hwndSource* is the handle to the source window in a direct manipulation operation. This function returns a poitner to the DRAGINFO structure allocated by the source window.

```
ULONG DrgQueryDragStatus(VOID);
```

This function returns a DGS_ constant specifying what type of drag operation is in progress. Table 20.6 lists the available constants.

Table 20.6 Values of DGS_* Constants

| Constant | Description |
|---|---|
| 0 | No direct manipulation operation is in progress |
| DGS_DRAGINPROGRESS | Modal operation is in progress |
| DGS_LAZYDRAGINPROGRESS | Modeless operation is in progress |

Note that this function could conceivably be handy for determining whether the "standard" function or the version which replaces it when direct manipulation is in progress should be called, for example, *WinGetPS* or *DrgGetPS*.

```
BOOL DrgLazyDrag(HWND hwndSource,
                 PDRAGINFO pdiInfo,
                 PDRAGIMAGE pdiImages,
                 ULONG ulNumImages,
                 PVOID pvReserved);
```

This function initiates a lazy drag operation. *hwndSource* specifies the source window handle. *pdiInfo* points to the DRAGINFO structure. *pdiImages* points to one or more DRAGIMAGE structures. *ulNumImages* specifies the number of DRAGIMAGE structures pointed to by pdiImages. *pvReserved* is reserved and must be NULL.

```
BOOL DrgLazyDrop(HWND hwndTarget,
                 ULONG ulOperation,
                 PPOINTL pptlDrop);
```

This function is called by a target to complete the lazy drag operation. *hwndTarget* is the target window handle. *ulOperation* specifies the operation to be performed and is a DO_ constant. *pptlDrop* points to a POINTL structure containing the mouse position in desktop-relative coordinates. This function returns TRUE if the operation was successfully initiated or FALSE otherwise.

```
BOOL DrgCancelLazyDrag(VOID);
```

This function is used to cancel a lazy drag operation. It returns TRUE if successful, or FALSE otherwise.

**Gotcha!**
With the *DrgQueryDraginfoPtr*, *DrgQueryDraginfoPtrFromHwnd* and *DrgQueryDraginfoPtrFromDragitem* functions, the application must still call *DrgAccessDraginfo* to get access to the structure returned.

**Gotcha!**
Be sure that if you initiate a lazy drag operation it is either completed or cancelled before your application terminates. The authors noticed that when the sample application (see below) was terminated without doing this that the direct manipulation subsystem seemed to get confused and no longer worked correctly.

**Gotcha!**
The Workplace Shell seems to be able to correctly determine if a lazy drag operation is in progress because it offers a "Cancel drag" menu item on its context-sensitive menus. However, selecting the menu item has no apparent effect. We cannot determine why this happens.

## Lazy Drag Sample

Below is a sample application which demonstrates the use of lazy drag and drop.

**DRAG3.C**

```
#define  INCL_DOSFILEMGR
#define  INCL_WINFRAMEMGR
#define  INCL_WININPUT
#define  INCL_WINMENUS
#define  INCL_WINPOINTERS
#define  INCL_WINSTDDRAG
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "drag3rc.h"
#define  CLS_CLIENT     "SampleClass"
#define  DRAG_METHOD    "DRM_OS2FILE"
#define  DRAG_FORMAT    "DRF_UNKNOWN"
#define  DRAG_RMF       "<"DRAG_METHOD","DRAG_FORMAT">"
```

```
#define  DRAG_TEMPNAME "DRAGDROP.TMP"
#define  MYM_DEWDROP   ( WM_USER )
typedef struct _CLIENTINFO
{
   PDRAGINFO        pdiDrag;
   HWND             hwndMenu;
   BOOL             bDragging;
   BOOL             bEmphasis;
   CHAR             achLine[256];
} CLIENTINFO,*PCLIENTINFO;

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile);

VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo);

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2);

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Drag - n - Drop Sample"
                                     " Application 3",
                                  0,
                                  NULLHANDLE,
                                  0,
                                  NULL);

   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
                     0,
                     0);
   while (bLoop)
   {
      WinDispatchMsg(habAnchor,
                     &qmMsg);
```

```
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
   }                                    /* endwhile              */
   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return (0);
}

VOID fileFromDragItem(PDRAGITEM pdiItem,PCHAR pchFile)
{
   CHAR             achPath[CCHMAXPATH];
   CHAR             achFile[CCHMAXPATH];

   DrgQueryStrName(pdiItem->hstrContainerName,
                   sizeof(achPath),
                   achPath);

   DosQueryPathInfo(achPath,
                    FIL_QUERYFULLNAME,
                    achPath,
                    sizeof(achPath));

   if (achPath[strlen(achPath)-1] != '\\')
   {
      strcat(achPath,
             "\\");
   }                                    /* endif                 */
   DrgQueryStrName(pdiItem->hstrSourceName,
                   sizeof(achFile),
                   achFile);

   sprintf(pchFile,
           "%s%s",
           achPath,
           achFile);
   return ;
}

VOID paintClient(HPS hpsClient,HWND hwndClient,PCLIENTINFO
                 pciInfo)
//--------------------------------------------------------------
// This function paints the client window according its current
// status.
//
// Input:  hpsClient - handle to the presentation space.
//         hwndClient - handle to the client window.
//         pciInfo - pointer to the CLIENTINFO structure
//          describing the client window.
//--------------------------------------------------------------

{
   RECTL            rclWindow;
   USHORT           usIndex;
   POINTL           ptlPaint;

   WinQueryWindowRect(hwndClient,
                      &rclWindow);
   WinFillRect(hpsClient,
               &rclWindow,
               SYSCLR_WINDOW);
   //-----------------------------------------------------------
   // Draw the dividing line
   //-----------------------------------------------------------

   ptlPaint.x = rclWindow.xRight/2;
```

```
   ptlPaint.y = rclWindow.yBottom;
   GpiMove(hpsClient,
           &ptlPaint);

   ptlPaint.y = rclWindow.yTop;
   GpiLine(hpsClient,
           &ptlPaint);
   //-------------------------------------------------------------
   // Set the color to indicate emphasized or
   // non-emphasized state
   //-------------------------------------------------------------

   if (pciInfo->bEmphasis)
   {
      GpiSetColor(hpsClient,
                  CLR_BLACK);
   }
   else
   {
      GpiSetColor(hpsClient,
                  SYSCLR_WINDOW);
   }                                    /* endif                */
   //-------------------------------------------------------------
   // Draw/erase the emphasis
   //-------------------------------------------------------------

   for (usIndex = 1; usIndex < 5; usIndex++)
   {
      ptlPaint.x = rclWindow.xRight/2+usIndex;
      ptlPaint.y = rclWindow.yBottom+usIndex;
      GpiMove(hpsClient,
              &ptlPaint);

      ptlPaint.x = rclWindow.xRight-usIndex;
      ptlPaint.y = rclWindow.yTop-usIndex;
      GpiBox(hpsClient,
             DRO_OUTLINE,
             &ptlPaint,
             0,
             0);
   }                                    /* endfor               */
   //-------------------------------------------------------------
   // Draw the instructing text
   //-------------------------------------------------------------

   WinQueryWindowRect(hwndClient,
                      &rclWindow);
   rclWindow.xRight /= 2;

   WinDrawText(hpsClient,
               -1,
               "Drag me",
               &rclWindow,
               CLR_BLACK,
               0,
               DT_CENTER|DT_VCENTER);

   if (pciInfo->achLine[0] != 0)
   {

      //----------------------------------------------------------
      // Draw the text received if we've been dropped on
      //----------------------------------------------------------
      WinQueryWindowRect(hwndClient,
                         &rclWindow);

      rclWindow.xLeft = rclWindow.xRight/2;
```

```
      WinDrawText(hpsClient,
                  -1,
                  pciInfo->achLine,
                  &rclWindow,
                  CLR_BLACK,
                  0,
                  DT_CENTER|DT_VCENTER);
   }                                  /* endif                */
   return ;
}

MRESULT doSource(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//--------------------------------------------------------------
// This function handles the direct-manipulation messages
//
// sent to the source part of the window.
//
// Input, output, returns:  as a window procedure
//--------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {

      case  WM_PICKUP :
         {
            RECTL            rclWindow;
            FILE             *pfFile;
            DRAGITEM         diItem;
            DRAGIMAGE        diImage;
            BOOL             bSuccess;

            if (DrgQueryDragStatus() == DGS_LAZYDRAGINPROGRESS)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */

            WinQueryWindowRect(hwndClient,
                               &rclWindow);
         //-------------------------------------------------------
         // If we're in the right half, return
         //-------------------------------------------------------

            if (SHORT1FROMMP(mpParm1) > rclWindow.xRight/2)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */
         //-------------------------------------------------------
         // Write the text to be dragged to a file,
         // since the type is
         // DRT_OS2FILE.
         //-------------------------------------------------------

            pfFile = fopen(DRAG_TEMPNAME,
                           "w");
            if (pfFile == NULL)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */

            fprintf(pfFile,
                    "Dropped text");
```

```
      fclose(pfFile);
   //-------------------------------------------------------
   // Allocate the DRAGITEM/DRAGINFO structures and
   // initialize them
   //-------------------------------------------------------

      pciInfo->pdiDrag = DrgAllocDraginfo(1);
      pciInfo->pdiDrag->hwndSource = hwndClient;

      diItem.hwndItem = hwndClient;
      diItem.ulItemID = 1;
      diItem.hstrType = DrgAddStrHandle(DRT_TEXT);
      diItem.hstrRMF = DrgAddStrHandle(DRAG_RMF);
      diItem.hstrContainerName = DrgAddStrHandle(".");
      diItem.hstrSourceName = DrgAddStrHandle(DRAG_TEMPNAME
         );

      diItem.hstrTargetName = diItem.hstrSourceName;
      diItem.cxOffset = 0;
      diItem.cyOffset = 0;
      diItem.fsControl = 0;
      diItem.fsSupportedOps = DO_MOVEABLE;

      DrgSetDragitem(pciInfo->pdiDrag,
                     &diItem,
                     sizeof(diItem),
                     0);
   //-------------------------------------------------------
   // Initialize the DRAGIMAGE structure
   //-------------------------------------------------------

      diImage.cb = sizeof(diImage);
      diImage.cptl = 0;
      diImage.hImage = WinQuerySysPointer(HWND_DESKTOP,
                                          SPTR_FILE,
                                          FALSE);
      diImage.sizlStretch.cx = 0;
      diImage.sizlStretch.cy = 0;
      diImage.fl = DRG_ICON;
      diImage.cxOffset = 0;
      diImage.cyOffset = 0;
   //-------------------------------------------------------
   // Set the bDragging flag and call DrgDrag().
   //-------------------------------------------------------

      pciInfo->bDragging = TRUE;

      bSuccess = DrgLazyDrag(hwndClient,
                             pciInfo->pdiDrag,
                             &diImage,
                             1L,
                             NULL);

      if (!bSuccess)
      {

         DrgDeleteDraginfoStrHandles(pciInfo->pdiDrag);
         DrgFreeDraginfo(pciInfo->pdiDrag);
         pciInfo->pdiDrag = NULL;
         pciInfo->bDragging = FALSE;
         remove(DRAG_TEMPNAME);

         WinInvalidateRect(hwndClient,
                           NULL,
                           FALSE);
         WinUpdateWindow(hwndClient);
      }                             /* endif                */
```

```
         }
         break;

      case  DM_DROPNOTIFY :
         break;

      case  DM_ENDCONVERSATION :
         {
            PDRAGITEM        pdiItem;
            CHAR             achFullFile[CCHMAXPATH];
         //-------------------------------------------------------
         // Query the item dragged for cleanup purposes
         //-------------------------------------------------------

            pdiItem = DrgQueryDragitemPtr(pciInfo->pdiDrag,
                                          0);
         //-------------------------------------------------------
         // Delete the file used to transfer the data
         //-------------------------------------------------------

            fileFromDragItem(pdiItem,
                             achFullFile);
            remove(achFullFile);
         //-------------------------------------------------------
         // Cleanup
         //-------------------------------------------------------

            DrgDeleteDraginfoStrHandles(pciInfo->pdiDrag);
            DrgFreeDraginfo(pciInfo->pdiDrag);
            pciInfo->pdiDrag = NULL;
            pciInfo->bDragging = FALSE;
         }
         break;

      default  :
         break;
   }                                  /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

MRESULT doTarget(HWND hwndClient,ULONG ulMsg,MPARAM mpParm1,
                 MPARAM mpParm2)
//------------------------------------------------------------------
// This function handles the direct-manipulation messages
// sent to the target part of the window.
//
// Input, output, returns:  as a window procedure
//------------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {
      case  DM_DRAGOVER :
         {
            PDRAGINFO        pdiDrag;
            POINTL           ptlDrop;
            RECTL            rclWindow;
            HPS              hpsWindow;
            PDRAGITEM        pdiItem;
```

```
//-----------------------------------------------------
// Get the pointer to the DRAGINFO structure
//-----------------------------------------------------

   pdiDrag = (PDRAGINFO)PVOIDFROMMP(mpParm1);
   DrgAccessDraginfo(pdiDrag);
//-----------------------------------------------------
// Since the drop position is in screen coordinates,
// map them to our window and check if it is
// in the right half of the window.  If not, return.
//-----------------------------------------------------

   ptlDrop.x = SHORT1FROMMP(mpParm2);
   ptlDrop.y = SHORT2FROMMP(mpParm2);

   WinMapWindowPoints(HWND_DESKTOP,
                      hwndClient,
                      &ptlDrop,
                      1);

   WinQueryWindowRect(hwndClient,
                      &rclWindow);

   if (ptlDrop.x < rclWindow.xRight/2)
   {
      if (pciInfo->bEmphasis)
      {
         pciInfo->bEmphasis = FALSE;

         hpsWindow = DrgGetPS(hwndClient);
         paintClient(hpsWindow,
                     hwndClient,
                     pciInfo);
         DrgReleasePS(hpsWindow);
      }                          /* endif                */
      return  MRFROMSHORT(DOR_NODROP);
   }                             /* endif                */
//-----------------------------------------------------
// Check to see if we've already turned emphasis on.
//-----------------------------------------------------

   if (!pciInfo->bEmphasis)
   {
      pciInfo->bEmphasis = TRUE;

      hpsWindow = DrgGetPS(hwndClient);
      paintClient(hpsWindow,
                  hwndClient,
                  pciInfo);
      DrgReleasePS(hpsWindow);
   }                             /* endif                */
//-----------------------------------------------------
// We should only be dragging one item.
//-----------------------------------------------------

   if (DrgQueryDragitemCount(pdiDrag) != 1)
   {
      return  MRFROMSHORT(DOR_NODROP);
   }                             /* endif                */
//-----------------------------------------------------
// Check the true type and native RMF
//-----------------------------------------------------

   pdiItem = DrgQueryDragitemPtr(pdiDrag,
                                 0);
```

```
         if (!DrgVerifyTrueType(pdiItem,
                                DRT_TEXT))
         {
            return  MRFROMSHORT(DOR_NODROP);
         }
         else

            if (!DrgVerifyNativeRMF(pdiItem,
                                    DRAG_RMF))
            {
               return  MRFROMSHORT(DOR_NODROP);
            }                           /* endif                */
         return  MRFROM2SHORT(DOR_DROP,
                              DO_MOVE);
      }
   case  DM_DRAGLEAVE :

      if (pciInfo->bEmphasis)
      {
         HPS              hpsWindow;
      //-------------------------------------------------------
      // Turn off the emphasis
      //-------------------------------------------------------

         pciInfo->bEmphasis = FALSE;

         hpsWindow = DrgGetPS(hwndClient);
         paintClient(hpsWindow,
                     hwndClient,
                     pciInfo);
         DrgReleasePS(hpsWindow);
      }                              /* endif                */
      break;

   case  DM_DROP :
      WinPostMsg(hwndClient,
                 MYM_DEWDROP,
                 mpParm1,
                 mpParm2);
      break;

   case  DM_DROPHELP :
      WinMessageBox(HWND_DESKTOP,
                    hwndClient,
                    "This is the drag - n - drop help."
                       "  Great, isn't it?",
                    "Help",
                    0,
                    MB_INFORMATION|MB_OK);
      break;

   case  MYM_DEWDROP :
      {
         PDRAGINFO        pdiDrag;
         PDRAGITEM        pdiItem;
         CHAR             achFullFile[CCHMAXPATH];
         FILE             *pfFile;
         HPS              hpsDrop;
      //-------------------------------------------------------
      // Get the pointer to the DRAGINFO structure
      //-------------------------------------------------------

         pdiDrag = (PDRAGINFO)PVOIDFROMMP(mpParm1);
         DrgAccessDraginfo(pdiDrag);
```

```
      //-----------------------------------------------------
      // Since we can render the object ourselves,
      // get the filename and read in the line
      //-----------------------------------------------------

         pdiItem = DrgQueryDragitemPtr(pdiDrag,
                                       0);
         fileFromDragItem(pdiItem,
                          achFullFile);

         pfFile = fopen(achFullFile,
                        "r");
         if (pfFile != NULL)
         {
            fgets(pciInfo->achLine,
                  sizeof(pciInfo->achLine),
                  pfFile);
            fclose(pfFile);
         }                              /* endif                */
      //-----------------------------------------------------
      // Turn the emphasis off and repaint ourselves to
      // show the dropped text
      //-----------------------------------------------------

         pciInfo->bEmphasis = FALSE;

         hpsDrop = WinGetPS(hwndClient);
         paintClient(hpsDrop,
                     hwndClient,
                     pciInfo);
         WinReleasePS(hpsDrop);
      //-----------------------------------------------------
      // Tell the source that we're done with the object
      //-----------------------------------------------------

         WinSendMsg(pdiDrag->hwndSource,
                    DM_ENDCONVERSATION,
                    MPFROMLONG(pdiItem->ulItemID),
                    MPFROMLONG(DMFL_TARGETSUCCESSFUL));
      //-----------------------------------------------------
      // Cleanup
      //-----------------------------------------------------

         DrgFreeDraginfo(pdiDrag);
      }
      break;
   case  WM_CONTEXTMENU :
      {
         POINTL           ptlPoint;
         RECTL            rclWindow;
         HWND             hwndMenu;

         if (DrgQueryDragStatus() == DGS_LAZYDRAGINPROGRESS)
         {
            WinQueryPointerPos(HWND_DESKTOP,
                               &ptlPoint);

            WinQueryWindowRect(hwndClient,
                               &rclWindow);

            if (ptlPoint.x < rclWindow.xRight/2)
            {
               return  MRFROMSHORT(FALSE);
            }                           /* endif                */
```

```
            hwndMenu = WinLoadMenu(HWND_OBJECT,
                                   NULLHANDLE,
                                   M_LAZYDRAG);

            WinPopupMenu(HWND_DESKTOP,
                         hwndClient,
                         hwndMenu,
                         ptlPoint.x,
                         ptlPoint.y,
                         0,
                         PU_MOUSEBUTTON1|PU_KEYBOARD);
         }                              /* endif                */
      }
      break;
   case  WM_MENUEND :
      if (SHORT1FROMMP(mpParm1) == FID_MENU)
      {
         WinDestroyWindow(HWNDFROMMP(mpParm2));
      }                                 /* endif                */
      break;
   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  MI_DROP :
            {
               POINTL           ptlPoint;

               WinQueryPointerPos(HWND_DESKTOP,
                                  &ptlPoint);
               DrgLazyDrop(hwndClient,
                           DO_DEFAULT,
                           &ptlPoint);
            }
            break;
         case  MI_CANCELDRAG :
            DrgCancelLazyDrag();
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }                                 /* endswitch            */
      break;

   default  :
      break;

}                                       /* endswitch            */
return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
//--------------------------------------------------------------
// This function handles the messages sent to the client window.
//
// Input, output, returns:  as a window procedure
//--------------------------------------------------------------

{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {
      case  WM_CREATE :
```

```
      //--------------------------------------------------------
      // Allocate memory for the instance data structure
      // and initialize it
      //--------------------------------------------------------

         pciInfo = malloc(sizeof(CLIENTINFO));
         if (pciInfo == NULL)
         {
            WinMessageBox(HWND_DESKTOP,
                          hwndClient,
                          "There is not enough memory.",
                          "Error",
                          0,
                          MB_ICONHAND|MB_OK);
            return  MRFROMSHORT(TRUE);
         }                              /* endif                */

         WinSetWindowPtr(hwndClient,
                         0,
                         pciInfo);

         pciInfo->pdiDrag = NULL;
         pciInfo->hwndMenu = WinLoadMenu(HWND_OBJECT,
                                         NULLHANDLE,
                                         M_LAZYDRAG);

         if (pciInfo->hwndMenu == NULLHANDLE)
         {
            WinMessageBox(HWND_DESKTOP,
                          hwndClient,
                          "Could not initialize the application."
                             ,
                          "Error",
                          0,
                          MB_ICONHAND|MB_OK);
            return  MRFROMSHORT(TRUE);
         }                              /* endif                */

         pciInfo->bDragging = FALSE;
         pciInfo->bEmphasis = FALSE;
         pciInfo->achLine[0] = 0;
         break;

      case  WM_DESTROY :
      //--------------------------------------------------------
      // Free the memory used by the instance data
      //--------------------------------------------------------
         if (DrgQueryDragStatus() == DGS_LAZYDRAGINPROGRESS)
         {
            DrgCancelLazyDrag();
         }                              /* endif                */
         WinDestroyWindow(pciInfo->hwndMenu);
         free(pciInfo);
         WinSetWindowPtr(hwndClient,
                         0,
                         NULL);
         break;

      case  WM_PAINT :
         {
            HPS              hpsPaint;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     NULL);
```

```
            paintClient(hpsPaint,
                        hwndClient,
                        pciInfo);
            WinEndPaint(hpsPaint);
         }
         break;

      case  WM_PICKUP :
      case  DM_DROPNOTIFY :
      case  DM_ENDCONVERSATION :
         return  doSource(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);

      case  DM_DRAGOVER :
      case  DM_DRAGLEAVE :
      case  DM_DROP :
      case  DM_DROPHELP :
      case  MYM_DEWDROP :
      case  WM_CONTEXTMENU :
      case  WM_MENUEND :
      case  WM_COMMAND :
         return  doTarget(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);

      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);
   }                                   /* endswitch                */
   return  MRFROMSHORT(FALSE);
}
```

## DRAG3.RC

```
#include <os2.h>
#include "drag3rc.h"

MENU M_LAZYDRAG
{
   MENUITEM "Drop item", MI_DROP
   MENUITEM "Cancel drag", MI_CANCELDRAG
}
```

## DRAG3RC.H

```
#define M_LAZYDRAG              256
#define MI_DROP                 257
#define MI_CANCELDRAG           258
```

## DRAG3.MAK

```
DRAG3.EXE:                      DRAG3.OBJ \
                                DRAG3.RES
        LINK386 @<<
DRAG3
DRAG3
DRAG3
OS2386
DRAG3
<<
        RC DRAG3.RES DRAG3.EXE
```

```
DRAG3.RES:                      DRAG3.RC \
                                DRAG3RC.H
        RC -r DRAG3.RC DRAG3.RES

DRAG3.OBJ:                      DRAG3.C \
                                DRAG3RC.H
        ICC -C+ -Kb+ -Ss+ DRAG3.C
```

**DRAG3.DEF**

```
NAME DRAG3 WINDOWAPI

DESCRIPTION 'Drag example 3
Copyright (c) 1995 by Larry Salomon
All rights reserved.'

STACKSIZE 16384
```

This sample was based on DRAG1, allowing the target to render the data so that the sample is not burdened with details not necessary to the discussion.

The first differences that you will note are the use of WM_PICKUP instead of WM_BEGINDRAG to begin the operation and the processing of the DM_DROPNOTIFY as the signal of the completion of the operation.

```
case  WM_PICKUP :
      case  DM_DROPNOTIFY :
      case  DM_ENDCONVERSATION :
         return  doSource(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);
```

Also, since the user must specify to the application that the operation is to be completed or cancelled, the WM_CONTEXTMENU, WM_MENUEND, and WM_COMMAND messages are processed to handle the user interface.

```
case  DM_DRAGOVER :
      case  DM_DRAGLEAVE :
      case  DM_DROP :
      case  DM_DROPHELP :
      case  MYM_DEWDROP :
      case  WM_CONTEXTMENU :
      case  WM_COMMAND :
         return  doTarget(hwndClient,
                          ulMsg,
                          mpParm1,
                          mpParm2);
```

The real work is done in *doSource* and *doTarget*, as was the case in the earlier samples.

```
case  WM_PICKUP :
         {
            RECTL            rclWindow;
            FILE             *pfFile;
            DRAGITEM         diItem;
            DRAGIMAGE        diImage;
            BOOL             bSuccess;

            if (DrgQueryDragStatus() == DGS_LAZYDRAGINPROGRESS)
```

```
            {
               return  MRFROMSHORT(FALSE);
            }  /* endif                */
```

Note how we check for a lazy-drag-in-progress and return immediately if this is true. This was done to keep the sample simple. The processing of WM_PICKUP then continues as it did for WM_BEGINDRAG except that we call *DrgLazyDrag* instead of *DrgDrag*.

```
bSuccess = DrgLazyDrag(hwndClient,
                       pciInfo->pdiDrag,
                       &diImage,
                       1L,
                       NULL);
```

From the target's perspective, we need to provide an interface to the user to allow them to complete or cancel the operation. This is done via the WM_CONTEXTMENU, WM_MENUEND, and WM_COMMAND messages.

```
case  WM_CONTEXTMENU :
   {
      POINTL            ptlPoint;
      RECTL             rclWindow;
      HWND              hwndMenu;

      if (DrgQueryDragStatus() == DGS_LAZYDRAGINPROGRESS)
      {
         WinQueryPointerPos(HWND_DESKTOP,
                            &ptlPoint);

         WinQueryWindowRect(hwndClient,
                            &rclWindow);

         if (ptlPoint.x < rclWindow.xRight/2)
         {
            return  MRFROMSHORT(FALSE);
         }  /* endif                */

         hwndMenu = WinLoadMenu(HWND_OBJECT,
                                NULLHANDLE,
                                M_LAZYDRAG);

         WinPopupMenu(HWND_DESKTOP,
                      hwndClient,
                      hwndMenu,
                      ptlPoint.x,
                      ptlPoint.y,
                      0,
                      PU_MOUSEBUTTON1|PU_KEYBOARD);
      }   /* endif                */
   }
   break;
case  WM_MENUEND :
   if (SHORT1FROMMP(mpParm1) == FID_MENU)
   {
      WinDestroyWindow(HWNDFROMMP(mpParm2));
   }   /* endif                */
   break;
case  WM_COMMAND :
   switch (SHORT1FROMMP(mpParm1))
   {
      case  MI_DROP :
         {
            POINTL           ptlPoint;
```

```
            WinQueryPointerPos(HWND_DESKTOP,
                               &ptlPoint);
            DrgLazyDrop(hwndClient,
                        DO_DEFAULT,
                        &ptlPoint);
         }
         break;
      case  MI_CANCELDRAG :
         DrgCancelLazyDrag();
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);
   }   /* endswitch             */
   break;
```

It should be pretty obvious that we are simply providing a popup menu for the user to select one of two choices—drop or cancel—and handling each choice appropriately.

Everything else about this sample is as it was in DRAG1, which demonstrates the ease with which a programmer can switch between using one mode or the other.

Before we close this topic, a question must be asked: how does the target specify whether or not a set of objects that were picked up can be dropped on it or not? In modal drag and drop, you receive the DM_DRAGOVER and DM_DRAGLEAVE to allow for user feedback, but these messages are not sent automatically by the system when a lazy drag operation is in progress. IBM's documentation states that these messages are sent when the user presses a key indicating that intention to drop the object, but nowhere do they state what this mythical key is. It is the opinion of the authors that this "key" is a concept and not an actual key on the keyboard, and we chose to implement the "key" concept as a popup menu. It is then, therefore, that the target determines the validity of the operation and acts appropriately.

# Chapter 21

# Value Set

A value set is a control that provides a way for a user to select from several graphically illustrated choices. Only one choice can be selected at a time. A value set can use icons, bitmaps, colors, text, or numbers. However, it is optimal to use only graphical images and/or short text; other controls should be used if a choice of only text or numbers is offered. The value set is designed to show setting choices, not action choices; if an action choice needs to to be designated, a push button or menu should be used. A value set must contain at least two items. A value set choice that is unavailable should be disabled; if a value set has text choices, a letter for each choice should be designated as a mnemonic. A value set can be used as a tool palette also; however, the pointer should be changed to represent the current "tool" selected. For instance, if a "paint" tool is selected, the cursor could be changed to represent a paintbrush. See Figure 21.1 for an example of a value set.

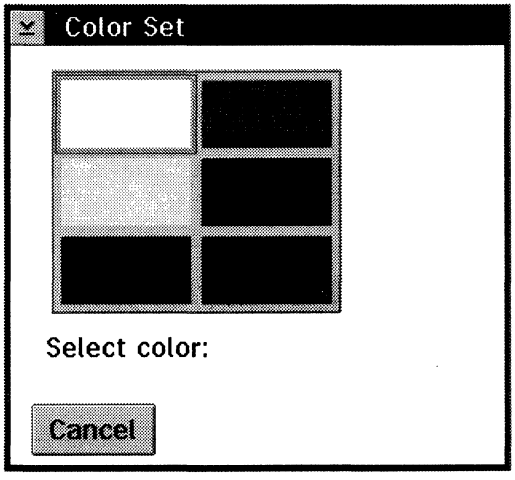



Figure 21.1 Example of the value set control.

## Value Set Styles

Table 21.1 lists the available value set styles.

Table 21.1 Value Set Styles

| Style | Description |
|---|---|
| VS_BITMAP | Default all value set items to bitmaps. |
| VS_ICON | Default all value set items to icons. |
| VS_TEXT | Default all value set items to text strings. |
| VS_RGB | Default all value set items to RGB values. |
| VS_COLORINDEX | Default all value set items to color indices. |
| VS_BORDER | Add a border to the value set control. |
| VS_ITEMBORDER | Add a border around each value set item. |
| VS_SCALEBITMAPS | Scale bitmaps to fit in cell region. |
| VS_RIGHTTOLEFT | Support right-to-left ordering. |
| VS_OWNERDRAW | Owner-drawn value set control. |

The following example program shows the creation of a value set control with the style VS_COLORINDEX.

**VALUE.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "value.h"
#define  CLS_CLIENT    "MyClass"
#define  UM_UPDATE   ( WM_USER )
static LONG alColors[] =
{

   CLR_BLUE,
   CLR_PINK,
   CLR_GREEN,
   CLR_CYAN,
   CLR_YELLOW,
   CLR_NEUTRAL,
   CLR_DARKGRAY,
   CLR_DARKBLUE,
   CLR_DARKRED,
   CLR_DARKPINK,
   CLR_DARKGREEN,
   CLR_DARKCYAN
} ;

typedef          struct
{
   SHORT            sColor;
   HWND             hwndDlg;
} WNDDATA,*PWNDDATA;

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

VOID DisplayError(CHAR *pszText);
VOID ProcessSelect(HWND hwndDlg,MPARAM mpParm2);

INT main(VOID)
{
```

```
HAB             habAnchor;
HMQ             hmqQueue;
ULONG           ulFlags;
HWND            hwndFrame;
BOOL            bLoop;
QMSG            qmMsg;
LONG            lWidth,lHeight;

habAnchor = WinInitialize(0);
hmqQueue = WinCreateMsgQueue(habAnchor,
                             0);

WinRegisterClass(habAnchor,
                 CLS_CLIENT,
                 ClientWndProc,
                 0,
                 sizeof(PVOID));

ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
   FCF_MENU|FCF_TASKLIST;

/*************************************************************/
/* create frame window                                       */
/*************************************************************/

hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                               0,
                               &ulFlags,
                               CLS_CLIENT,
                               "Value Set Example",
                               0,
                               NULLHANDLE,
                               ID_FRAME,
                               NULL);

/*************************************************************/
/* get screen height and width                               */
/*************************************************************/

lWidth = WinQuerySysValue(HWND_DESKTOP,
                          SV_CXSCREEN);

lHeight = WinQuerySysValue(HWND_DESKTOP,
                           SV_CYSCREEN);

/*************************************************************/
/* if failed, display error, and set to default value        */
/*************************************************************/

if (!lWidth || !lHeight)
{
   DisplayError("WinQuerySysValue failed");
   lWidth = 640;
   lHeight = 480;
}

if (hwndFrame != NULLHANDLE)
{

   /*********************************************************/
   /* set window position                                   */
   /*********************************************************/
```

```
      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      10,
                      10,
                      lWidth/10*8,
                      lHeight/10*8,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PWNDDATA         pwdData;

   switch (ulMsg)
   {

      case  WM_CREATE :

         /*****************************************************/
         /* allocate window word                              */
         /*****************************************************/

         pwdData = (PWNDDATA)malloc(sizeof(WNDDATA));
         if (pwdData == NULL)
         {
            DisplayError("Unable to allocate window word");
            return  MRFROMSHORT(TRUE);
         }                               /* endif                 */

         WinSetWindowPtr(hwndClient,
                         QWL_USER,
                         pwdData);
         pwdData->sColor = -1;
         break;

      case  WM_COMMAND :
         pwdData = WinQueryWindowPtr(hwndClient,
                                     QWL_USER);
         switch (SHORT1FROMMP(mpParm1))
         {
            case  IDM_DISPLAY :

               /*********************************************/
               /* load up dialog, if dialog already exists,    */
               /* just make it visible                         */
               /*********************************************/
```

```
            if (!pwdData->hwndDlg)
               pwdData->hwndDlg = WinLoadDlg(HWND_DESKTOP,
                                             hwndClient,
                                             DlgProc,
                                             NULLHANDLE,
                                             IDD_VALUE,
                                             NULL);
            else
               WinSetWindowPos(pwdData->hwndDlg,
                               HWND_TOP,
                               0,
                               0,
                               0,
                               0,
                               SWP_SHOW|SWP_ACTIVATE);
            break;
         case  IDM_EXIT :

            /***********************************************/
            /* close up window                             */
            /***********************************************/

            WinPostMsg(hwndClient,
                       WM_CLOSE,
                       MPVOID,
                       MPVOID);
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }
      break;

   case  WM_DESTROY :

      /*****************************************************/
      /* free up memory                                    */
      /*****************************************************/

      pwdData = WinQueryWindowPtr(hwndClient,
                                  QWL_USER);
      if (pwdData)
         free(pwdData);
      break;

   case  UM_UPDATE :

      /*****************************************************/
      /* user message indicates end-user selected new color */
      /* in value set, window needs to repaint itself with  */
      /* new color                                         */
      /*****************************************************/

      pwdData = WinQueryWindowPtr(hwndClient,
                                  QWL_USER);
      if (!pwdData)
      {
         DisplayError("WinQueryWindowPtr failed");
         break;
      }
      pwdData->sColor = SHORT1FROMMP(mpParm1);
      WinInvalidateRect(hwndClient,
                        NULL,
                        FALSE);
      WinUpdateWindow(hwndClient);
```

```
         break;

      case  WM_PAINT :
         {
            HPS               hpsPaint;
            RECTL             rclPaint;
            SHORT             sColor;

            /*************************************************/
            /* variable to indicate whether to paint or not  */
            /*************************************************/

            BOOL              bPaint = FALSE;

            pwdData = WinQueryWindowPtr(hwndClient,
                                        QWL_USER);

            /*************************************************/
            /* paint the entire client with the dropped color */
            /*************************************************/

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     &rclPaint);
            GpiErase(hpsPaint);

            /*************************************************/
            /* do some error checking                        */
            /*************************************************/

            if (pwdData)
            {
               if (pwdData->sColor >= 0)
               {
                  bPaint = TRUE;
                  sColor = pwdData->sColor;
               }
            }
            if (bPaint)
               WinFillRect(hpsPaint,
                           &rclPaint,
                           alColors[sColor]);
            WinEndPaint(hpsPaint);
         }
         break;

      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                    /* endswitch             */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)
{

   switch (ulMsg)
   {
      case  WM_INITDLG :
         {
            SHORT             sColor;
            USHORT            usX;
            USHORT            usY;
            ULONG             ulReturn;
            MRESULT           mrReply;
```

```
      sColor = 0;

      /*************************************************/
      /* loop through the rows and columns to initialize */
      /* items                                         */
      /*************************************************/

      for (usX = 1; usX <= 3; usX++)
      {
         for (usY = 1; usY <= 4; usY++)
         {

            mrReply = WinSendDlgItemMsg(hwndDlg,
                                        IDV_VALUE,
                                        VM_SETITEM,
                                        MPFROM2SHORT(usX,
                                                     usY),
                                        MPFROMLONG(alColors
                                           [sColor++]));
            if (!LONGFROMMR(mrReply))
               DisplayError("WinSendDlgItemMsg failed");

         }                          /* endfor               */
      }                             /* endfor               */
   }                                /* end WM_INITDLG       */
   break;

case  WM_CONTROL :
   if (SHORT1FROMMP(mpParm1) == IDV_VALUE)
   {
      switch (SHORT2FROMMP(mpParm1))
      {
         case  VN_SELECT :
            {
               ProcessSelect(hwndDlg,
                             mpParm2);
               break;
            }
         default  :
            return  WinDefDlgProc(hwndDlg,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

      }                             /* endswitch            */
   }                                /* end if == IDV_VALUE  */

   break;                           /* end WM_CONTROL       */
case  WM_COMMAND :
   switch (SHORT1FROMMP(mpParm1))
   {
      case  DID_CANCEL :
         WinDismissDlg(hwndDlg,
                       FALSE);
         break;

      default  :
         return  WinDefDlgProc(hwndDlg,
                               ulMsg,
                               mpParm1,
                               mpParm2);
   }                                /* endswitch mpParm1    */

   break;                           /* end WM_COMMAND       */
default  :
```

```
        return  WinDefDlgProc(hwndDlg,
                              ulMsg,
                              mpParm1,
                              mpParm2);

   }                                    /* endswitch            */
   return  MRFROMSHORT(FALSE);

}

VOID DisplayError(CHAR *pszText)
{

   /*************************************************************/
   /* small function to display error string                    */
   /*************************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}

VOID ProcessSelect(HWND hwndDlg,MPARAM mpParm2)
{

   /*************************************************************/
   /* small function to get color that was selected             */
   /*************************************************************/

   HWND             hwndClient,hwndFrame;
   USHORT           usRow,usCol;
   SHORT            sColorIndex;
   BOOL             bSuccess;

   /*************************************************************/
   /* get row and column of selected item                       */
   /*************************************************************/

   usRow = SHORT1FROMMP(mpParm2);
   usCol = SHORT2FROMMP(mpParm2);

   /*************************************************************/
   /* calculate index into color array                          */
   /*************************************************************/

   sColorIndex = ((usRow-1)*4)+(usCol-1);
   if (sColorIndex < 0)
   {
      DisplayError("Invalid selected item");
      return ;
   }

   /*************************************************************/
   /* get the client window handle to post message              */
   /*************************************************************/

   hwndFrame = WinWindowFromID(HWND_DESKTOP,
                               ID_FRAME);
   if (!hwndFrame)
   {
      DisplayError("WinWindowFromID:1 failed");
```

```
      return ;
   }
   hwndClient = WinWindowFromID(hwndFrame,
                                FID_CLIENT);
   if (!hwndClient)
   {
      DisplayError("WinWindowFromID:2 failed");
      return ;
   }

   bSuccess = WinPostMsg(hwndClient,
                         UM_UPDATE,
                         MPFROMSHORT(sColorIndex),
                         MPVOID);

   if (!bSuccess)
      DisplayError("WinPostMsg failed");

   return ;

}
```

**VALUE.RC**

```
#include <os2.h>
#include "value.h"


MENU ID_FRAME
{
   SUBMENU "~File", -1
   {
      MENUITEM "~Display Dialog", IDM_DISPLAY
      MENUITEM SEPARATOR
      MENUITEM "E~xit", IDM_EXIT
   }
}



DLGTEMPLATE IDD_VALUE LOADONCALL MOVEABLE DISCARDABLE
{
    DIALOG  "Color Set", IDD_VALUE, 12, 12, 155, 105, WS_VISIBLE,
            FCF_SYSMENU | FCF_TITLEBAR
    {
        LTEXT           "Select color: ", -1, 11, 25, 102, 8
        VALUESET        IDV_VALUE, 13, 38, 91, 61, VS_COLORINDEX | VS_BORDER
                        CTLDATA 8, 0, 3, 4
        PUSHBUTTON      "Cancel", DID_CANCEL, 6, 2, 40, 14
    }
}
```

**VALUE.H**

```
#define IDD_VALUE               100
#define IDV_VALUE               101
#define IDM_DISPLAY             102
#define IDM_EXIT                103
#define ID_FRAME                104
```

**VALUE.MAK**

```
VALUE.EXE:                      VALUE.OBJ \
                                VALUE.RES
        LINK386 @<<
VALUE
VALUE
VALUE
OS2386
VALUE
<<
        RC VALUE.RES VALUE.EXE

VALUE.RES:                      VALUE.RC \
                                VALUE.H
        RC -r VALUE.RC VALUE.RES

VALUE.OBJ:                      VALUE.C \
                                VALUE.H
        ICC -C+ -Kb+ -Ss+ VALUE.C
```

**VALUE.DEF**

```
NAME VALUE WINDOWAPI

DESCRIPTION 'Value-set example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

# The VALUE.RC Resource File

The VALUE.RC file contains two items: a menu and a dialog with the value set control. The dialog is created with the following code.

```
DLGTEMPLATE IDD_VALUE LOADONCALL MOVEABLE DISCARDABLE
{
    DIALOG  "Color Set", IDD_VALUE, 12, 12, 155, 105, WS_VISIBLE,
            FCF_SYSMENU | FCF_TITLEBAR
    {
        LTEXT           "Select color: ", -1, 11, 25, 102, 8
        VALUESET        IDV_VALUE, 13, 38, 91, 61, VS_COLORINDEX |
          VS_BORDER
                        CTLDATA 8, 0, 3, 4
        PUSHBUTTON      "Cancel", DID_CANCEL, 6, 2, 40, 14
    }
}
```

The sixth parameter in the VALUESET statement is the combination of window and control styles. In this case, we specify VS_COLORINDEX, indicating that the choices of the value set are the indices into the color index table. We also use VS_BORDER, which draws a border around the value set. The last parameter is the CTLDATA statement. In this case, this represents the VSCDATA structure. The VSCDATA structure is defined as:

```
typedef struct _VSCDATA  /* vscd */
{
   ULONG   cbSize;       /* Size of control block            */
   USHORT  usRowCount;   /* Number of rows in value set      */
   USHORT  usColumnCount;/* Number of columns in value set   */
} VSCDATA;
typedef VSCDATA *PVSCDATA;
```

The CTLDATA key word sees each parameter as a SHORT, so a LONG is represented as two parameters. The first two parameters correspond to the *cbSize* structure member. They are specified in low-byte, high-byte order. The third parameter represents *usRowCount*. Our value set will contain three rows. The fourth parameter represents *usColumnCount*. Our value set will contain four columns.

A structure defined at the top of the program is used for the window word. It is:

```
typedef struct {
   SHORT       sColor ;
   HWND        hwndDlg ;
} WNDDATA, * PWNDDATA ;
```

In the structure, the first element SHORT *sColor* represents the currently selected color in the value set. The *hwndDlg* is the window handle for the dialog box.

Also, the array *alColor* is declared. This is the array of color index values that are used in the value set.

# Initializing the Value Set

```
SHORT            sColor;
USHORT           usX;
USHORT           usY;
ULONG            ulReturn;
MRESULT          mrReply;

sColor = 0;

for (usX = 1; usX <= 3; usX++)
{
   for (usY = 1; usY <= 4; usY++)
   {

      mrReply = WinSendDlgItemMsg(hwndDlg,
                                  IDV_VALUE,
                                  VM_SETITEM,
                                  MPFROM2SHORT(usX,
                                               usY),
                                  MPFROMLONG(alColors
                                       [sColor++]));
   }
 }
```

The value set initialization is a very simple process of sending a VM_SETITEM for each item in the value set. Because this value set is of style VS_COLORINDEX, *mpParm2* will contain a color index constant. We will use the CLR_* values in the *alColors* array. *mpParm1* is a collection of two SHORTS that make up the row and column of the item. Notice that there is no row or column 0; these values start at 1. All value set messages pertaining to a specific value set item are done by using the row and column of the item of interest.

By default, the first item in the value set is selected.

## Value Set Selection Notification

```
usRow = SHORT1FROMMP(mpParm2);
usCol = SHORT2FROMMP(mpParm2);

sColorIndex = ((usRow-1)*4)+(usCol-1);

hwndFrame = WinWindowFromID(HWND_DESKTOP,
                            ID_FRAME);
hwndClient = WinWindowFromID(hwndFrame,
                             FID_CLIENT);

bSuccess = WinPostMsg(hwndClient,
                      UM_UPDATE,
                      MPFROMSHORT(sColorIndex),
                      MPVOID);
```

The WM_CONTROL message is where the value set will indicate when a new color has been selected. We check for the notification code VN_SELECT from the WM_CONTROL message. The row number (starting with 1) is sent as the low order byte of *mpParm2*. The column number is sent as the high order byte of *mpParm2*. By doing some quick math, the index into the *alColor* array is determined. The next task is to notify the client window that a new selection has been made. This is done by posting the user-defined message, UM_UPDATE, to the client, with the color index sent in *mpParm1*.

## VALUE Paint Processing

```
HPS             hpsPaint;
RECTL           rclPaint;
SHORT           sColor;
BOOL            bPaint = FALSE;

pwdData = WinQueryWindowPtr(hwndClient,
                            QWL_USER);

hpsPaint = WinBeginPaint(hwndClient,
                         NULLHANDLE,
                         &rclPaint);
GpiErase(hpsPaint);

if (pwdData)
{
   if (pwdData->sColor >= 0)
   {
      bPaint = TRUE;
      sColor = pwdData->sColor;
   }
}
if (bPaint)
   WinFillRect(hpsPaint,
               &rclPaint,
               alColors[sColor]);
WinEndPaint(hpsPaint);
}
break;
```

The WM_PAINT message starts with *WinQueryWindowPtr* to retrieve the window word of the client window. Next the usual *WinBeginPaint* is called. *GpiErase* is used to erase the entire invalidated region. If the *sColor* variable in the *pwdData* structure is greater than 0, a color has been selected by the user. Remember, the variable was initially set to -1. A Boolean variable *bPaint* is used to indicate all is okay, so go ahead and paint. *WinFillRect* fills the invalidated region with the specified color, and *WinEndPaint* is called to release the presentation space.

# The User-defined Message UM_UPDATE

```
pwdData = WinQueryWindowPtr(hwndClient,
                            QWL_USER);
if (!pwdData)
{
   DisplayError("WinQueryWindowPtr failed");
   break;
}
pwdData->sColor = SHORT1FROMMP(mpParm1);
WinInvalidateRect(hwndClient,
                  NULL,
                  FALSE);
WinUpdateWindow(hwndClient);
```

The message UM_UPDATE is a user-defined message that is sent from the value set when a new value set item has been selected. This is the signal to the client to repaint itself. The index of the selected item is sent in *mpParm1*. This value is retrieved and stored in the *pwdData* structure so it is visible to the WM_PAINT processing. *WinInvalidateRect* is used to invalidate the entire client window, and *WinUpdateWindow* is used to force the update of the client window—in other words, generate a WM_PAINT message and process it, *now*!

```
BOOL WinUpdateWindow(HWND hwnd);
```

*WinUpdateWindow* has only one parameter—*hwnd*, which is the window handle of the window to update.

This potent approach is not always necessary, but the example program depends on a quick user notification of the new value set selection.

# Chapter 22

# Notebook

The notebook control is designed to provide the user with a visual organizer of information, similar to a real notebook with dividers. Information can be broken up into categories, with the major tabs representing category headings. Information can then be further broken up using minor tabs as the subcategory headings. The notebook consists of six major parts, as illustrated in Figure 22.1: the binding, status line, intersection of pages, forward/backward page buttons, major tabs, and minor tabs.

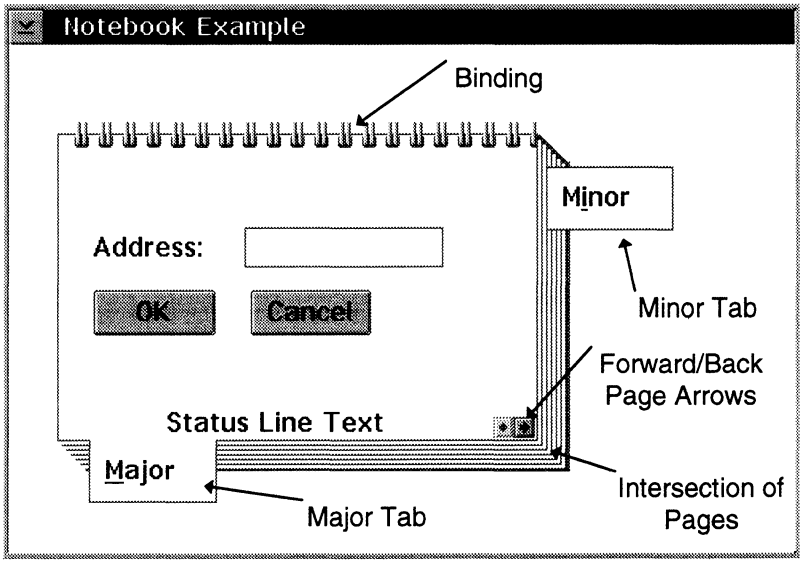



Figure 22.1 Drawing of a notebook.

A notebook should be used to offer the user a choice of settings or to present data that can be organized logically into categories or groups. Information that can be grouped together should be put into a single tabbed section. Major tabs can be placed at any of the four notebook sides; however, minor tabs always are placed perpendicular to the major tabs. Page buttons are provided to allow the user to page forward and backward between the notebook pages. Page buttons always are located in the corner that is flanked by the back pages. The binding can either be spiral-bound or solid-bound, depending on the specified style. A line of status text can be associated with each notebook page. If more than one page exists in a category, the status line should be used to indicate this to the user; for example, "Page 1 of 20." The status line can be left-justified, right-justified, or centered along the bottom of the notebook. The last part of the notebook

is the intersection of the back pages, used to design a landscape- or portrait-mode notebook. This feature gives the appearance of a three-dimensional notebook. This intersection can be located at any of the four corners. Figures 22.2 through 22.9 show the eight possible combinations of styles.

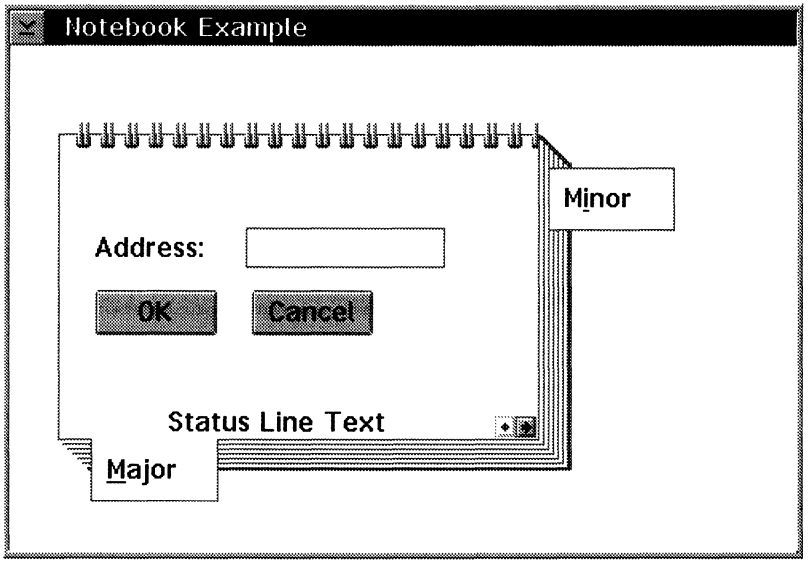



Figure 22.2 BKS_BACKPAGESBR | BKS_MAJORTABBOTTOM.

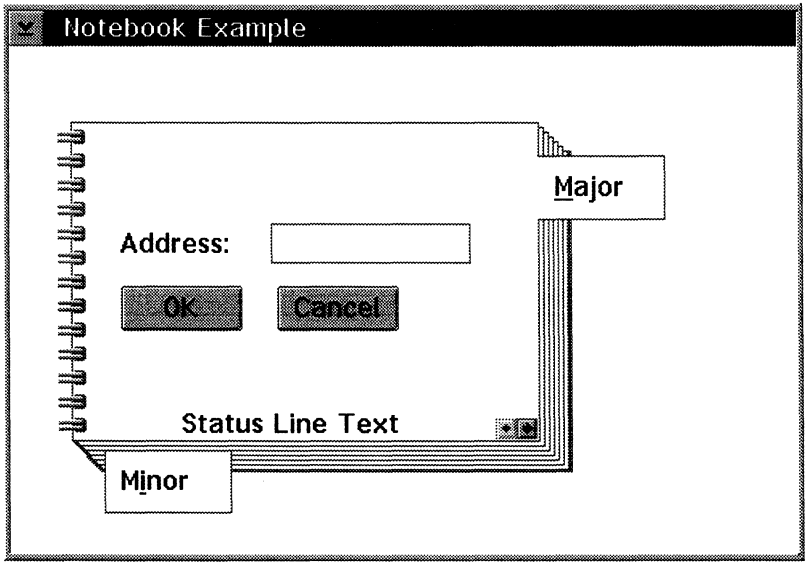



Figure 22.3 BKS_PAGESBR | BKS_MAJORTABRIGHT.

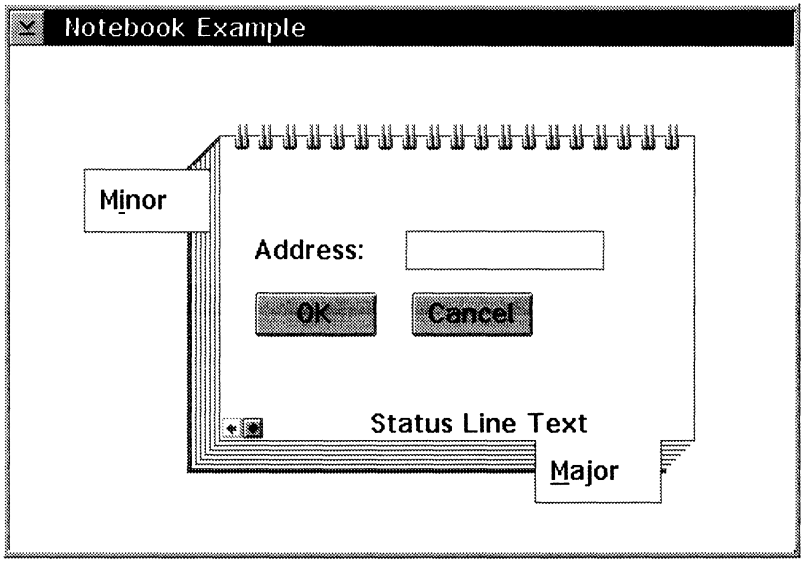



Figure 22.4 BKS_BACKPAGESBL | BKS_MAJORTABBOTTOM.

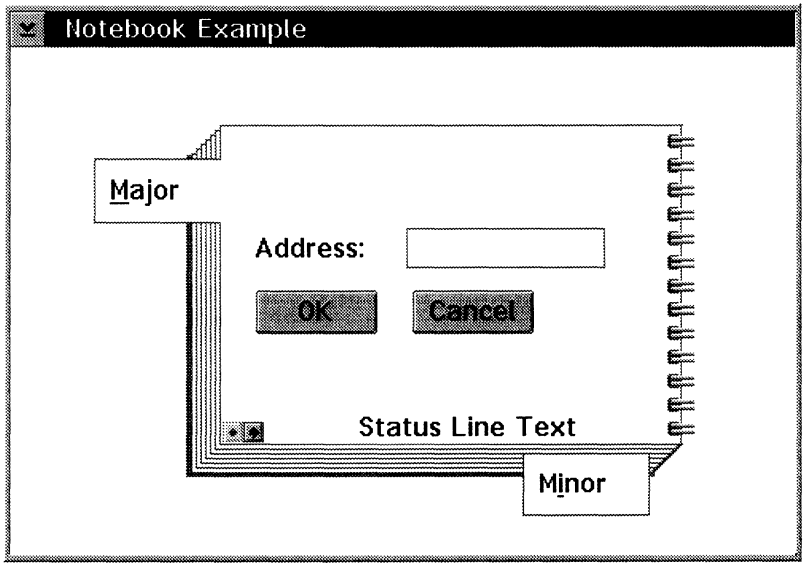



Figure 22.5 BKS_BACKPAGESBL | BKS_MAJORTABLEFT.

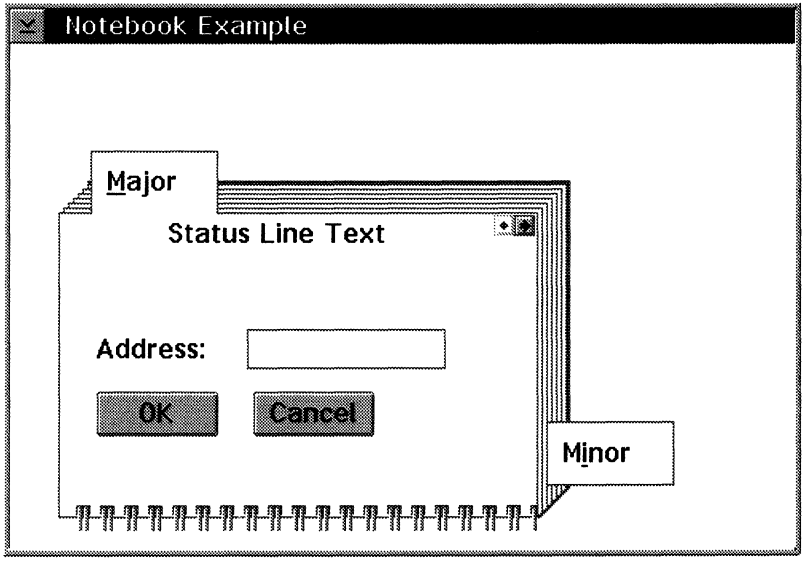



Figure 22.6 BKS_BACKPAGESTR | BKS_MAJORTABTOP.

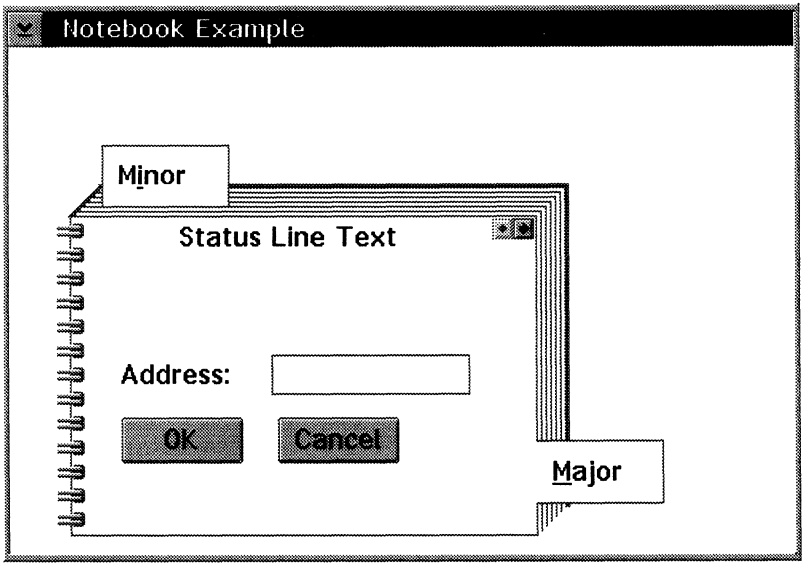



Figure 22.7 BKS_BACKPAGESTR | BKS_MAJORTABRIGHT.

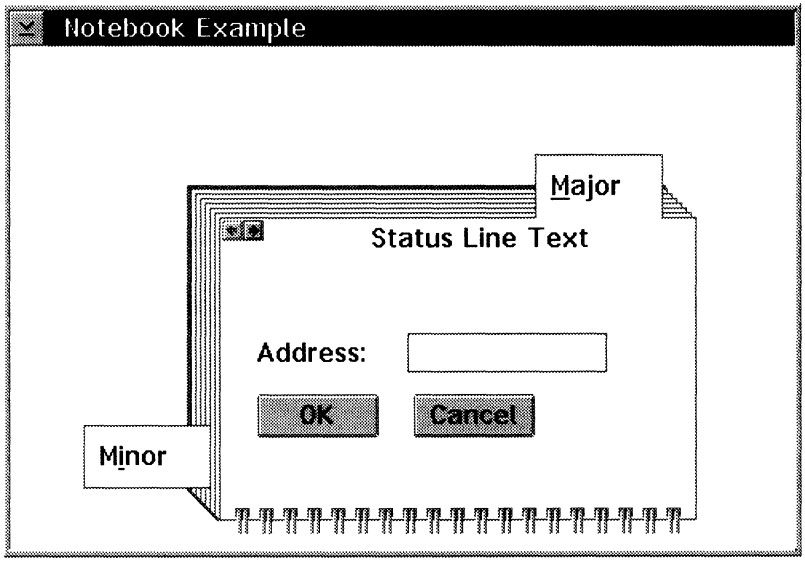



Figure 22.8 BKS_BACKPAGESTL | BKS_MAJORTABTOP.

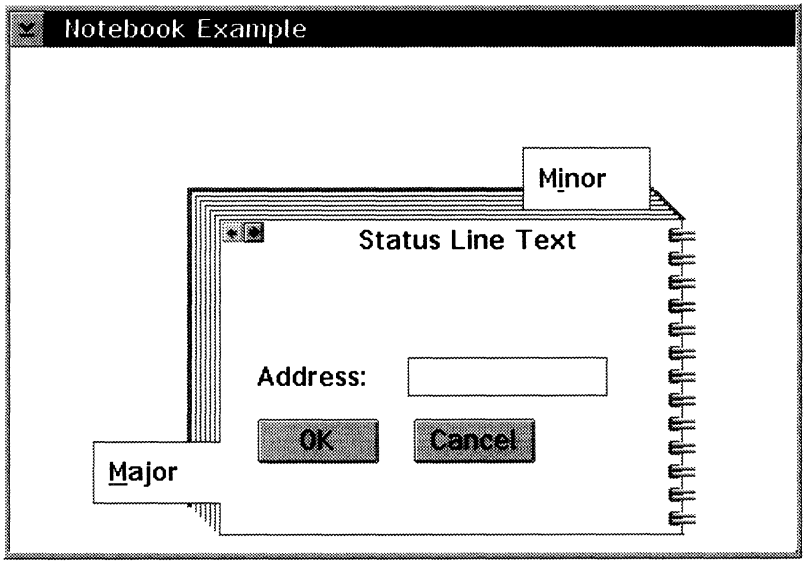



Figure 22.9 BKS_BACKPAGESTL | BKS_MAJORTABLEFT.

The following are the notebook window styles:

Table 22.1 Notebook Window Styles

| Style | Description |
|---|---|
| BKS_BACKPAGESBR | Intersection of pages is located at the bottom right corner. |
| BKS_BACKPAGESBL | Intersection of pages is located at the bottom left corner. |
| BKS_BACKPAGESTR | Intersection of pages is located at the top right corner. |
| BKS_BACKPAGESBR | Intersection of pages is located at the top left corner. |
| BKS_MAJORTABRIGHT | Major tabs are located on the right side. |
| BKS_MAJORTABLEFT | Major tabs are located on the left side. |
| BKS_MAJORTABTOP | Major tabs are located on the top side. |
| BKS_MAJORTABBOTTOM | Major tabs are located on the bottom side. |
| BKS_SQUARETABS | Notebook has squared-edge tabs. |
| BKS_ROUNDEDTABS | Notebook has rounded-edge tabs. |
| BKS_POLYGONTABS | Notebook has polygon-edge tabs. |
| BKS_SOLIDBIND | Notebook has a solid binding. |
| BKS_SPIRALBIND | Notebook has a spiral binding. |
| BKS_STATUSTEXTLEFT | Notebook has the status text left-justified. |
| BKS_STATUSTEXTRIGHT | Notebook has the status text right-justified. |
| BKS_STATUSTEXTCENTER | Notebook has the status text centered. |
| BKS_TABTEXTLEFT | Notebook has the tab text left-justified. |
| BKS_TABTEXTRIGHT | Notebook has the tab text right-justified. |
| BKS_TABTEXTCENTER | Notebook has the tab text centered. |

The major and minor tabs can be customized somewhat. They can be square or polygonal or have rounded corners. A tab can contain eithertext or bitmaps. The text can be left-justified, right-justified, or centered. If a bitmap is specified for the tab, the bitmap is sized automatically to fill the tab. The dimensions for the tab need to be set using the message BKM_SETDIMENSIONS. There is no automatic sizing of the tab for text.

# Notebook Pages

A notebook page is designed to be associated with a dialog box or window. When a new page is selected in the notebook, the notebook invalidates the new page, causing a WM_PAINT to be sent to the procedure associated with the newly selected page. When a notebook is created, the initialization should handle the insertion of any needed pages. If a page has a major or minor tab associated with it, this is specified in the BKM_INSERTPAGE. The following code segment shows how to insert a page.

```
ULONG ulPageID;
MRESULT mrReply;

mrReply = WinSendMsg( hwndNotebook,
        BKM_INSERTPAGE,
        (MPARAM)0,
        MPFROM2SHORT ( BKA_MAJOR | BKA_STATUSTEXTON,
        BKA_FIRST ));

ulPageID = LONGFROMMR( Reply );
```

If no major or minor tabs are specified, the new page becomes part of the current section. Each page has a *ulPageID* that is *returned* from the BKM_INSERTPAGE message. This ID is used extensively in the notebook messaging system.

The following example program illustrates the creation of a notebook.

## NOTEBOOK.C

```
#define  INCL_GPICONTROL
#define  INCL_GPILCIDS
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "notebook.h"
#define  CLS_CLIENT    "MyClass"
#define  UM_CREATEDONE (WM_USER + 1)
BOOL InitializeNotebook(HWND hwndNotebook,HWND hwndPage1,HWND
                        hwndPage2,LONG lCxChar,LONG lCyChar);

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

VOID DisplayError(CHAR *pszText);

INT main(VOID)
{

  HAB              habAnchor;
  HMQ              hmqQueue;
  ULONG            ulFlags;
  HWND             hwndFrame;
  BOOL             bLoop;
  QMSG             qmMsg;
  LONG             lWidth,lHeight;

  habAnchor = WinInitialize(0);
  hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

  WinRegisterClass(habAnchor,
                   CLS_CLIENT,
                   ClientWndProc,
                   0,
                   0);

  ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
     FCF_TASKLIST;

  /*************************************************************/
  /* create frame window                                       */
  /*************************************************************/

  hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                 0,
                                 &ulFlags,
                                 CLS_CLIENT,
                                 "Notebook Example",
                                 0,
                                 NULLHANDLE,
                                 0,
                                 NULL);

  /*************************************************************/
  /* get screen height and width                               */
  /*************************************************************/

  lWidth = WinQuerySysValue(HWND_DESKTOP,
                            SV_CXSCREEN);
```

```
  lHeight = WinQuerySysValue(HWND_DESKTOP,
                             SV_CYSCREEN);

  /*************************************************************/
  /* if failed, display error, and set to default value        */
  /*************************************************************/

  if (!lWidth || !lHeight)
  {
    DisplayError("WinQuerySysValue failed");
    lWidth = 640;
    lHeight = 480;
  }

  if (hwndFrame != NULLHANDLE)
  {

    /***********************************************************/
    /* set window position                                     */
    /***********************************************************/

    WinSetWindowPos(hwndFrame,
                    NULLHANDLE,
                    lWidth/8,
                    lHeight/8,
                    lWidth/8*6,
                    lHeight/8*6,
                    SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

    bLoop = WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
    while (bLoop)
    {
      WinDispatchMsg(habAnchor,
                     &qmMsg);
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
    }                                   /* endwhile              */
    WinDestroyWindow(hwndFrame);
  }                                     /* endif                 */
  WinDestroyMsgQueue(hmqQueue);
  WinTerminate(habAnchor);
  return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
  switch (ulMsg)
  {

    case  WM_CREATE :
      WinPostMsg(hwndWnd,
                 UM_CREATEDONE,
                 MPVOID,
                 MPVOID);
      break;

    case  UM_CREATEDONE :
      {
        HPS              hpsChar;
        FONTMETRICS      fmMetrics;
```

```
    LONG            lCxChar;
    LONG            lCyChar;
    HWND            hwndNotebook;
    HWND            hwndPage1;
    HWND            hwndPage2;
    RECTL           rclClient;

    hpsChar = WinGetPS(hwndWnd);
    GpiQueryFontMetrics(hpsChar,
                        sizeof(fmMetrics),
                        &fmMetrics);
    WinReleasePS(hpsChar);

    lCxChar = fmMetrics.lAveCharWidth;
    lCyChar = fmMetrics.lMaxBaselineExt;

    WinQueryWindowRect(hwndWnd,
                       &rclClient);

   hwndNotebook = WinCreateWindow(hwndWnd,
                                  WC_NOTEBOOK,
                                  "",
                                  BKS_SPIRALBIND|
                                     BKS_SQUARETABS|
                                     BKS_STATUSTEXTCENTER,
                                  0,
                                  0,
                                  rclClient.xRight,
                                  rclClient.yTop,
                                  hwndWnd,
                                  HWND_TOP,
                                  ID_NOTEBOOK,
                                  NULL,
                                  NULL);

    if (!hwndNotebook)
      DisplayError("WinCreateWindow failed");

    hwndPage1 = WinLoadDlg(hwndWnd,
                           hwndWnd,
                           DlgProc,
                           NULLHANDLE,
                           IDD_PERSONAL,
                           NULL);

    hwndPage2 = WinLoadDlg(hwndWnd,
                           hwndWnd,
                           DlgProc,
                           NULLHANDLE,
                           IDD_OS2,
                           NULL);

    InitializeNotebook(hwndNotebook,
                       hwndPage1,
                       hwndPage2,
                       lCxChar,
                       lCyChar);

    WinShowWindow(hwndNotebook,
                  TRUE);

    WinSetFocus(HWND_DESKTOP,
                WinWindowFromID(hwndPage1,
                                IDE_NAME));
  }
  break;
```

```
case  WM_SIZE :
  {
    HWND              hwndNotebook;

    hwndNotebook = WinWindowFromID(hwndWnd,
                                   ID_NOTEBOOK);
    WinSetWindowPos(hwndNotebook,
                    NULLHANDLE,
                    0,
                    0,
                    SHORT1FROMMP(mpParm2),
                    SHORT2FROMMP(mpParm2),
                    SWP_SIZE|SWP_SHOW);
  }
  break;

case  WM_CONTROL :
  switch (SHORT1FROMMP(mpParm1))
  {
    case  ID_NOTEBOOK :
      switch (SHORT2FROMMP(mpParm1))
      {
        case  BKN_PAGESELECTED :
          {
            PPAGESELECTNOTIFY ppsnSelect;
            HWND              hwndPage;
            USHORT            usDlgId;
            MRESULT           mrReply;

            ppsnSelect = PVOIDFROMMP(mpParm2);

            mrReply = WinSendMsg(ppsnSelect->hwndBook,
                                 BKM_QUERYPAGEWINDOWHWND,
                                 MPFROMLONG
                                    (ppsnSelect->ulPageIdNew),
                                 0);
            hwndPage = (HWND)PVOIDFROMMR(mrReply);

            usDlgId = WinQueryWindowUShort(hwndPage,
                                           QWS_ID);

            if (usDlgId == IDD_PERSONAL)
            {
              WinSetFocus(HWND_DESKTOP,
                          WinWindowFromID(hwndPage,
                                          IDE_NAME));
            }
            else
            {
              WinSetFocus(HWND_DESKTOP,
                          WinWindowFromID(hwndPage,
                                          IDE_TEAMOS2));
            }                           /* endif                  */
          }
          break;

        default  :
          return  WinDefWindowProc(hwndWnd,
                                   ulMsg,
                                   mpParm1,
                                   mpParm2);

      }                                 /* endswitch              */
      break;
    default  :
```

```
        return  WinDefWindowProc(hwndWnd,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2);

      }                                 /* endswitch              */
      break;

    case  WM_PAINT :
      {
        HPS             hpsPaint;

        hpsPaint = WinBeginPaint(hwndWnd,
                                 NULLHANDLE,
                                 NULLHANDLE);
        GpiErase(hpsPaint);
        WinEndPaint(hpsPaint);
      }
      break;

    default  :
      return  WinDefWindowProc(hwndWnd,
                               ulMsg,
                               mpParm1,
                               mpParm2);

  }                                     /* endswitch              */
  return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)
{
  switch (ulMsg)
  {

    case  WM_COMMAND :

      switch (SHORT1FROMMP(mpParm1))
      {

        case  DID_OK :
        case  DID_CANCEL :
          break;

        default  :
          return  WinDefDlgProc(hwndDlg,
                                ulMsg,
                                mpParm1,
                                mpParm2);

      }                                 /* endswitch              */
      break;

    default  :
      return  WinDefDlgProc(hwndDlg,
                            ulMsg,
                            mpParm1,
                            mpParm2);

  }                                     /* endswitch              */
  return  MRFROMSHORT(FALSE);
}
```

```
BOOL InitializeNotebook(HWND hwndNotebook,HWND hwndPage1,HWND
                        hwndPage2,LONG lCxChar,LONG lCyChar)
{
  ULONG            ulIdPage1;
  ULONG            ulIdPage2;
  ULONG            ulWidth;
  CHAR             achPage1Text[64];
  CHAR             achPage2Text[64];

  ulIdPage1 = LONGFROMMR(WinSendMsg(hwndNotebook,
                                    BKM_INSERTPAGE,
                                    0,
                                    MPFROM2SHORT(BKA_MAJOR|
                                       BKA_STATUSTEXTON,
                                                BKA_FIRST)));

  ulIdPage2 = LONGFROMMR(WinSendMsg(hwndNotebook,
                                    BKM_INSERTPAGE,
                                    0,
                                    MPFROM2SHORT(BKA_MAJOR|
                                       BKA_STATUSTEXTON,
                                                BKA_LAST)));

  WinSendMsg(hwndNotebook,
             BKM_SETSTATUSLINETEXT,
             MPFROMLONG(ulIdPage1),
             MPFROMP("Personal Information for This User"));

  WinSendMsg(hwndNotebook,
             BKM_SETSTATUSLINETEXT,
             MPFROMLONG(ulIdPage2),
             MPFROMP("TEAM OS / 2 Information"" for this Location")
     );

  strcpy(achPage1Text,
         " ~Personal");
  strcpy(achPage2Text,
         " ~TEAMOS2");

  ulWidth = (max(strlen(achPage1Text),
                 strlen(achPage2Text))+6)*lCxChar;

  WinSendMsg(hwndNotebook,
             BKM_SETDIMENSIONS,
             MPFROM2SHORT(ulWidth,
                          lCyChar*2),
             MPFROMSHORT(BKA_MAJORTAB));

  WinSendMsg(hwndNotebook,
             BKM_SETTABTEXT,
             MPFROMLONG(ulIdPage1),
             MPFROMP(achPage1Text));

  WinSendMsg(hwndNotebook,
             BKM_SETTABTEXT,
             MPFROMLONG(ulIdPage2),
             MPFROMP(achPage2Text));

  WinSendMsg(hwndNotebook,
             BKM_SETPAGEWINDOWHWND,
             MPFROMLONG(ulIdPage1),
             MPFROMHWND(hwndPage1));

  WinSendMsg(hwndNotebook,
             BKM_SETPAGEWINDOWHWND,
             MPFROMLONG(ulIdPage2),
             MPFROMHWND(hwndPage2));
```

```
  WinSendMsg(hwndNotebook,
             BKM_SETNOTEBOOKCOLORS,
             MPFROMLONG(CLR_BLUE),
             MPFROMSHORT(BKA_FOREGROUNDMAJORCOLORINDEX));

  return  TRUE;
}

VOID DisplayError(CHAR *pszText)
{

  /************************************************************/
  /* small function to display error string                   */
  /************************************************************/

  WinAlarm(HWND_DESKTOP,
           WA_ERROR);
  WinMessageBox(HWND_DESKTOP,
                HWND_DESKTOP,
                pszText,
                "Error!",
                0,
                MB_OK|MB_ERROR);

  return ;

}
```

## NOTEBOOK.RC

```
#include <os2.h>
#include "notebook.h"

DLGTEMPLATE IDD_OS2 LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG  "TEAMOS2 Information", IDD_OS2, 5, 0, 265, 126, NOT FS_DLGBORDER
            PRESPARAMS PP_BACKGROUNDCOLORINDEX, 0xFFFFFFFEL
    BEGIN
        GROUPBOX        "TEAMOS2 Information", -1, 0, 26, 213, 101
        LTEXT           "TEAMOS2 Name:", -1, 6, 106, 74, 8
        LTEXT           "User Group:", -1, 6, 85, 53, 8
        LTEXT           "Level of OS/2 Experience", -1, 59, 67, 109, 8
        ENTRYFIELD      "", IDE_TEAMOS2, 90, 102, 108, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_USERGROUP, 90, 85, 108, 8, ES_MARGIN
        AUTORADIOBUTTON "OS/2 Beginner", IDR_BEGIN, 68, 52, 77, 10,
                        WS_TABSTOP
        AUTORADIOBUTTON "OS/2 Intermediate", IDR_INTERMEDIATE, 68, 39, 91,
                        10, WS_TABSTOP
        AUTORADIOBUTTON "OS/2 Expert", IDR_EXPERT, 68, 27, 65, 10,
                        WS_TABSTOP
    END
END

DLGTEMPLATE IDD_PERSONAL LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG  "Personal Information", IDD_PERSONAL, 5, 0, 213, 126, NOT
            FS_DLGBORDER
            PRESPARAMS PP_BACKGROUNDCOLORINDEX, 0xFFFFFFFEL
    BEGIN
        GROUPBOX        "Personal Information", -1, 0, 26, 213, 101
        LTEXT           "Name:", -1, 6, 106, 29, 8
        LTEXT           "Address:", -1, 6, 85, 39, 8
        LTEXT           "City:", -1, 6, 63, 19, 8
        LTEXT           "State:", -1, 134, 63, 26, 8
        LTEXT           "Zip:", -1, 6, 40, 18, 8
```

```
        LTEXT           "Phone:", -1, 134, 40, 31, 8
        ENTRYFIELD      "", IDE_NAME, 55, 107, 145, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_ADDRESS, 55, 85, 145, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_CITY, 31, 65, 96, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_STATE, 169, 65, 31, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_ZIP, 31, 42, 96, 8, ES_MARGIN
        ENTRYFIELD      "", IDE_PHONE, 169, 42, 31, 8, ES_MARGIN
    END
END
```

## NOTEBOOK.H

```
#define ID_NOTEBOOK                 256
#define IDD_PERSONAL                257
#define IDD_OS2                     258

#define IDE_NAME                    512
#define IDE_ADDRESS                 513
#define IDE_CITY                    514
#define IDE_STATE                   515
#define IDE_ZIP                     516
#define IDE_PHONE                   517

#define IDE_TEAMOS2                 528
#define IDE_USERGROUP               529
#define IDR_BEGIN                   530
#define IDR_INTERMEDIATE            531
#define IDR_EXPERT                  532
```

## NOTEBOOK.MAK

```
NOTEBOOK.EXE:                   NOTEBOOK.OBJ \
                                NOTEBOOK.RES
        LINK386 @<<
NOTEBOOK
NOTEBOOK
NOTEBOOK
OS2386
NOTEBOOK
<<
        RC NOTEBOOK.RES NOTEBOOK.EXE

NOTEBOOK.RES:                   NOTEBOOK.RC \
                                NOTEBOOK.H
        RC -r NOTEBOOK.RC NOTEBOOK.RES

NOTEBOOK.OBJ:                   NOTEBOOK.C \
                                NOTEBOOK.H
        ICC -C+ -Kb+ -Ss+ NOTEBOOK.C
```

## NOTEBOOK.DEF

```
NAME NOTEBOOK WINDOWAPI

DESCRIPTION 'Notebook example
Copyright (c) 1992-1995 by Kathleen Panov.
All rights reserved.'

STACKSIZE 16384
```

## Flipping Pages

In the WM_CONTROL message processing, the BKN_PAGESELECTED notification code is sent each time a new page is selected in the notebook. We'll use this message as a signal to set the focus to the specified dialog control for the selected page. The BKN_PAGESELECTED notification code returns a pointer to the PAGESELECTNOTIFY structure. The structure looks like this:

```
typedef struct _PAGESELECTNOTIFY     /* pgsntfy */
   {
      HWND    hwndBook;             /* Notebook window handle    */
      ULONG   ulPageIdCur;          /* Previous top page id      */
      ULONG   ulPageIdNew;          /* New top Page id           */
   } PAGESELECTNOTIFY;
   typedef PAGESELECTNOTIFY *PPAGESELECTNOTIFY;
```

The item we are interested in is the new top page ID, *ulPageIdNew*. This value is used to query the window handle of the new page.

```
ppsnSelect = PVOIDFROMMP(mpParm2);

mrReply = WinSendMsg(ppsnSelect->hwndBook,
                     BKM_QUERYPAGEWINDOWHWND,
                     MPFROMLONG
                        (ppsnSelect->ulPageIdNew),
                     0);
hwndPage = (HWND)PVOIDFROMMR(mrReply);
```

Once we have the window handle, we query for the ID of the new top page. If the ID belongs to the dialog, IDD_PERSONAL, we set the focus to the first entry field, IDE_NAME. Otherwise, we know the dialog is the TEAMOS2 dialog, and we set the focus to the first entry field in that dialog, IDE_TEAMOS2.

```
usDlgId = WinQueryWindowUShort(hwndPage,
                               QWS_ID);

if (usDlgId == IDD_PERSONAL)
{
  WinSetFocus(HWND_DESKTOP,
              WinWindowFromID(hwndPage,
                              IDE_NAME));
}
else
{
  WinSetFocus(HWND_DESKTOP,
              WinWindowFromID(hwndPage,
                              IDE_TEAMOS2));
}                       /* endif                    */
```

## Creating a Notebook

The notebook is created using *WinCreateWindow* after the client area has been created.

```
hwndNotebook = WinCreateWindow(hwndWnd,
                               WC_NOTEBOOK,
                               "",
                               BKS_SPIRALBIND|
                                  BKS_SQUARETABS|
                                  BKS_STATUSTEXTCENTER,
                               0,
                               0,
                               rclClient.xRight,
                               rclClient.yTop,
                               hwndWnd,
                               HWND_TOP,
                               ID_NOTEBOOK,
                               NULL,
                               NULL);
```

When the UM_CREATEDONE message is received, the first thing we do is query the font size using *GpiQueryFontMetrics*. This function is covered in Chapter 10. This information is passed to the notebook initialization routine. The window size is found using *WinQueryWindowRect*. For more information on this function, see Chapter 10. Once these values are found, the actual notebook window is created using *WinCreateWindow*. *hwndWnd*, the client window handle, is specified as the parent in the first parameter of *WinCreateWindow*. The class style is WC_NOTEBOOK. The third parameter is window text, in this case "". The fourth parameter is the notebook style. The style flags specified here are BKS_SPIRALBIND | BKS_SQUARETABS | BKS_STATUSTEXTCENTER. This creates a spiral binding on the notebook, squared-edge tabs, and the status text centered. The next four parameters, 0, 0, *rclClient.xRight*, and *rclClient.yTop*, designate a notebook position of row 0, column 0 on the client window, a width equal to the client window width, and a height equal to the client window height. The ninth parameter, *hwndWnd*, is the owner window. The next parameter is window Z-order. Our notebook is to be placed on top of all the other windows, so HWND_TOP is specified. The next parameter is resource ID, ID_NOTEBOOK. The last two parameters are class-specific data and presentation data. In this instance there is none, so NULL is specified for both. Notice the notebook is created invisible and made visible after initialization is complete. This is a very good technique to use, especially if the window or dialog initialization is lengthy. The notebook initialization is done with the user function *InitializeNotebook* that is covered in the next section.

The next step is to load the two dialog boxes to be placed in the notebook. The first one is *hwndPage1*. It contains such basic information as "Name" and "Address." The next dialog, *hwndPage2*, contains OS/2-specific information from the user.

Last, we make the notebook window visible using *WinShowWindow*, and set the focus to the first entry field on the first page with the function *WinSetFocus*. These functions are discussed in Chapter 10.

## *InitializeNotebook*

This function performs all the initialization required for our notebook. The first messages sent to the notebook are BKM_INSERTPAGE.

```
ulIdPage1 = LONGFROMMR(WinSendMsg(hwndNotebook,
                                  BKM_INSERTPAGE,
                                  0,
                                  MPFROM2SHORT(BKA_MAJOR|
                                     BKA_STATUSTEXTON,
                                              BKA_FIRST)));
```

```
ulIdPage2 = LONGFROMMR(WinSendMsg(hwndNotebook,
                                  BKM_INSERTPAGE,
                                  0,
                                  MPFROM2SHORT(BKA_MAJOR|
                                     BKA_STATUSTEXTON,
                                              BKA_LAST)));
```

*mpParm1* in the BKM_INSERTPAGE message, in this case 0, is used to indicate a page ID to be used as a reference point for *mpParm2*. Specifying BKA_FIRST or BKA_LAST as page location alleviates the need for a page ID reference. If, for example, BKA_NEXT had been specified, *mpParm1* would have to contain the ID of the page the inserted page is to be placed next to. *mpParm2* contains two SHORTS. The first SHORT is the notebook page styles. In both of the preceding cases, a major tab is placed on the page, and status text is enabled at the bottom of the page. The second SHORT is the page insertion order. In this example, BKA_FIRST is specified. The page ID of the newly created pages is returned in the Reply and is stored in *ulIDPage1* and *ulIDPage2*.

```
WinSendMsg(hwndNotebook,
           BKM_SETSTATUSLINETEXT,
           MPFROMLONG(ulIdPage1),
           MPFROMP("Personal Information for This User"));
```

Since the status text is enabled, the next step is to copy text into it. Each section of the notebook only has one page, so information about that page will suffice. The message BKM_SETSTATUSLINETEXT is used to place text on the status line. *mpParm1* of the message indicates the page ID where the text is to be placed, and *mpParm2* is the text string itself.

```
ulWidth = ( max ( strlen ( achPage1Text ) ,
   strlen ( achPage2Text )) + 6 ) * lCxChar ;

WinSendMsg ( hwndNotebook,
             BKM_SETDIMENSIONS,
             MPFROM2SHORT ( ulWidth, lCyChar * 2 ) ,
             MPFROMSHORT ( BKA_MAJORTAB )) ;
```

After the status text is taken care of, the tabs need to have their dimensions set so that the tab text will not be truncated. The BKM_SETDIMENSIONS message is used to set the size. *mpParm1* is two SHORTs. The first short is tab width. The variable, *ulWidth*, is determined by calculating the maximum number of characters on the tab by the width of each character, *lCxChar*. The second SHORT is the height. For this parameter, *lCyChar* * 2 is used. *mpParm2* is the part for which the dimensions are being set, in this case BKA_MAJORTAB.

```
WinSendMsg ( hwndNotebook,
             BKM_SETTABTEXT,
             MPFROMLONG ( ulIdPage1 ) ,
             MPFROMP ( achPage1Text )) ;
```

The tab text is set for each major tab using the message BKM_SETTABTEXT. *mpParm1* is the page ID on which the tab is located. *mpParm2* is the text string that is placed on the tab.

```
WinSendMsg ( hwndNotebook,
             BKM_SETPAGEWINDOWHWND,
             MPFROMLONG ( ulIdPage1 ) ,
             MPFROMHWND ( hwndPage1 )) ;
```

Right now our notebook has two blank pages in it. The next step is to associate something with each page. A window or a dialog box can be associated with a page. Only one item can be associated with each page. The notebook itself is not designed for painting, rather just as a kind of "display device" for windows or dialog boxes.

The message BKM_SETPAGEWINDOWHWND associates a window handle with a page. *mpParm1* is the ID of the page on which the window is to be placed. *mpParm2* is the window handle that will be placed on the specified page of the notebook.

```
WinSendMsg ( hwndNotebook,
             BKM_SETNOTEBOOKCOLORS,
             MPFROMLONG ( CLR_BLUE ) ,
             MPFROMSHORT ( BKA_FOREGROUNDMAJORCOLORINDEX )) ;
```

The last *WinSendMsg* call sends the message BKM_SETNOTEBOOKCOLORS to change the presentation parameters on the notebook control. *mpParm1* of the message is the color or color index to use. *mpParm2* is the appropriate presentation parameter. Notice that these are different from the normal window presentation parameters. This example uses the BKA_FOREGROUNDMAJORCOLORINDEX to change the foreground colors on the major tabs. When using indices into the color palette, only the CLR_ values are acceptable. When specifying the color, RGB representation should be used.

The notebook control was added to OS/2 2.0 and has become very widely used by both new 32-bit applications and the operating system itself. A notebook should be used to present settings for an object or to present data that can be divided by groups. A notebook should not be used when a very small group of data is represented; in such a case, regular dialog boxes should be used.

# Chapter 23

# Containers

It was a happy occasion when Tupperware containers were invented. Not only could leftover meatloaf be stored in them, but so could crayons, plants, or almost anything else you desired. The container didn't know about the specifics of the items you stored, nor did it care; it simply stored the items.

OS/2 also has a container that has a similar purpose: to store items. It doesn't care if the items are employee names or sales statistics or the batting averages of the 1929 Yankees. The items to be stored are defined by the application. Additionally, the container control supports multiple views of the objects, in concordance with the CUA 1991 specification. Multiple-object selection methods are supported as well as direct editing of text and drag and drop. In short, the container can do anything save wash your windows or butter your bread.

This extreme amount of functionality and flexibility is not without its price, unfortunately. The container is a very complex control that demands a fair amount of initialization, and almost every message sent to and from the container references a structure or two. This chapter discusses container basics and develops a couple of applications to demonstrate the concepts discussed; the more advanced topics will be left to the reader.

## Container Styles

Table 23.1 describes the container styles and their meanings.

Table 23.1 Container Styles

| Style | Description |
|---|---|
| CCS_EXTENDSEL | Specifies that the extended selection model is to be used according to the CUA '91 guidelines. |
| CCS_MULTIPLESEL | Specifies that one or more items can be selected at any time. |
| CCS_SINGLESEL | Specifies that only a single item may be selected at any time. This is the default. |
| CCS_AUTOPOSITION | Specifies that the container should position items automatically when one of a specific set of events occurs. This is valid for icon view only. |
| CCS_MINIRECORDCORE | Specifies that the object records are of the type MINIRECORDCORE (instead of RECORDCORE). |
| CCS_READONLY | Specifies that no text should be editable. |
| CCS_VERIFYPOINTERS | Specifies that the container should verify that all pointers used belong to the object list. It does not validate the accessibility of the pointers. This should be used only during debugging, since it affects the performance of the container. |

## LPs or 45s?

The basic data unit of a container is a structure that describes the state of an individual item within the container. Depending on whether the CCS_MINIRECORDCORE style bit is specified, this is either a RECORDCORE or MINIRECORDCORE structure. There are advantages to using either; the former requires more setup but is more flexible, while the latter requires less setup but is more limiting. (Here we use the RECORDCORE structure in our discussions but we use the MINIRECORDCORE structure in the samples.) Additional bytes at the end of the record can be specified when the record is allocated. Thus, typically a structure would be defined by the programmer, whose first field is the RECORDCORE structure; the structure would be typecast to the appropriate structure type for messages sent to or from the container.

```
typedef struct _ITEMINFO {
   MINIRECORDCORE mrcRecord;
   CHAR achItem[256];
   ULONG ulUnitsSold;
   float fRevenue;
} ITEMINFO, *PITEMINFO;
```

Programmers always should be sure to specify the style bit that corresponds to the type of object record they decide to use.

Records are allocated using the CM_ALLOCRECORD message with the extra bytes needed beyond the RECORDCORE structure specified in the first parameter and the number of records to allocate specified in the second parameter. Obviously, for performance reasons, allocating one record at a time should be avoided. Instead, if at all possible, the number of records needed should be determined and allocated in one call. If more than one record is allocated, the head of a linked list of records is returned, with the link specified in the *preccNextRecord* field. Note that allocating memory for the records is not equivalent to inserting the records into the container. This is done using the CM_INSERTRECORD message and, as before, should be done with as many records as possible to increase performance.

The CM_INSERTRECORD message requires the first parameter to contain the head of the linked list of the (one or more) records to insert. The second parameter points to a RECORDINSERT structure.

```
typedef struct _RECORDINSERT {
   ULONG cb;
   PRECORDCORE pRecordOrder;
   PRECORDCORE pRecordParent;
   ULONG fInvalidateRecord;
   ULONG zOrder;
   ULONG cRecordsInsert;
} RECORDINSERT;
```

*cb* is the size of the structure in bytes. *pRecordOrder* specifies the record after which the record(s) are to be inserted. CMA_FIRST or CMA_END also can be specified to indicate that the record(s) should go at the front or end of the record list. *pRecordParent* specifies the parent record and can be NULL to indicate a top-level record. This field is valid only for tree view. *fInvalidateRecord* is TRUE if the records are to be invalidated (and thus redrawn) after being inserted. *zOrder* specifies the Z-order of the record and can be either CMA_TOP or CMA_BOTTOM to specify the top and bottom of the Z-order. *cRecordsInsert* specifies the number of records that are being inserted.

## Half Full or Half Empty?

We stated before that the container supports multiple views of its objects. This is a perfect time to elaborate because it introduces us to the CNRINFO structure, which is used to control a variety of container characteristics.

```
typedef struct _CNRINFO {
ULONG cb;
   PVOID pSortRecord;
   PFIELDINFO pFieldInfoLast;
   PFIELDINFO pFieldInfoObject;
   PSZ pszCnrTitle;
   ULONG flWindowAttr;
   POINTL ptlOrigin;
   ULONG cDelta;
   ULONG cRecords;
   SIZEL slBitmapOrIcon;
   SIZEL slTreeBitmapOrIcon;
   HBITMAP hbmExpanded;
   HBITMAP hbmCollapsed;
   HPOINTER hptrExpanded;
   HPOINTER hptrCollapsed;
   LONG cyLineSpacing;
   LONG cxTreeIndent;
   LONG cxTreeLine;
   ULONG cFields;
   LONG xVertSplitbar;
} CNRINFO;
```

The CNRINFO structure contains a large number of fields. Note that not every one of them needs to be initialized. Instead, only the needed fields are initialized; fields which were initialized are cited as a combination of flags specified in the second parameter of the CM_SETCNRINFO message. To change the view to icon view, for example:

```
CNRINFO ciInfo;

ciInfo.cb=sizeof(CNRINFO);
ciInfo.flWindowAttr=CV_ICON;
```

```
        WinSendMsg(pcdData->hwndCnr,
                   CM_SETCNRINFO,
                   MPFROMP(&ciInfo),
                   MPFROMLONG(CMA_FLWINDOWATTR));
```

Since we're talking about views of an object, let's look at the various combinations of view flags to specify the different view types. Table 23.2 provides a list of view flags.

Table 23.2 View Flags

| Views | Description |
|---|---|
| CV_ICON | Specifies that the icon or bitmap should be displayed with the text below it. |
| CV_NAME | Specifies that the icon should be displayed with the text to the right. This can be combined with the CV_FLOW flag. |
| CV_TEXT | Specifies that the text alone should be displayed. This can be combined with the CV_FLOW flag. |
| CV_TREE | Used for records that have children. Three view types can be used with the tree view. (See "Tree View" on page 409.) The tree view shows a hierarchical view of the data. |
| CV_DETAIL | The details view shows data in a columnar format. This is discussed in more detail later in this chapter on page 409 . |
| CV_FLOW | The CV_FLOW flag specifies that, once a column is filled, the list should continue in an adjacent column. |

The following sections look at each view type in detail.

## Icon, Name, and Text Views

The icon view is perhaps the most widely known because it is the default view for the folders on the desktop. It consists of an icon or bitmap representing the object, with text directly beneath it. The text can be "directly edited"—the user can, using the mouse and/or keyboard, directly edit the text. (The application controls whether the container retains the changes.)

If the container was created with the CCS_AUTOPOSITION style, the objects are arranged automatically whenever any of the following events occur:

- The window size changes
- Container items are inserted, removed, sorted, invalidated, or filtered
- The font or font size changes
- The window title text changes

This arranging occurs as if the container were sent a CM_ARRANGE message.

The name view consists of the icon or bitmap representing the object with the text immediately to the right. As with the icon view, the text can be edited directly. If CV_FLOW is not specified, objects are arranged vertically in a single column. If CV_FLOW is specified, a new column is created to the right if the objects extend beyond the bottom of the container.

The text view consists of the text only, and the objects are arranged in the same manner as the name view, with the same semantics regarding the specification of the CV_FLOW flag.

The following application illustrates these three views of a container's contents.

**CONTAIN1.C**

```
#define  INCL_WINPOINTERS
#define  INCL_WINSTDCNR
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "contain1.h"
#define  CLS_CLIENT     "SampleClass"
#define  MAX_YEARS      10
#define  MAX_MONTHS     12
#define  MAX_COLUMNS    4
#define  CX_SPLITBAR    120
typedef struct _CLIENTDATA
{
   HWND              hwndCnr;

   HPOINTER          hptrIcon;
} CLIENTDATA,*PCLIENTDATA;

typedef struct   _SALESINFO
{
   MINIRECORDCORE    mrcStd;
   ULONG             ulNumUnits;
   float             fSales;
   PCHAR             pchSales;
} SALESINFO,*PSALESINFO;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   BOOL              bLoop;
   QMSG              qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU|FCF_MENU;
```

```
   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Container Sample",
                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                 /* endwhile                  */
      WinDestroyWindow(hwndFrame);
   }                                    /* endif                     */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

VOID initSalesInfo(PCLIENTDATA pcdData,PSALESINFO psiSales,USHORT
                   usIndex)
{
   PCHAR            pchPos;

   psiSales->mrcStd.cb = sizeof(MINIRECORDCORE);

   psiSales->mrcStd.pszIcon = malloc(256);
   if (psiSales->mrcStd.pszIcon != NULL)
   {
      sprintf(psiSales->mrcStd.pszIcon,
              "Year 19%02d",
              usIndex+84);

   }                                    /* endif                     */
   psiSales->mrcStd.hptrIcon = pcdData->hptrIcon;
   psiSales->ulNumUnits = usIndex *usIndex;
   psiSales->fSales = (float)psiSales->ulNumUnits *9.95;

   psiSales->pchSales = malloc(16);
   if (psiSales->pchSales != NULL)
   {
      sprintf(psiSales->pchSales,
              "$%-10.2f",
              psiSales->fSales);

      pchPos = psiSales->pchSales;
      while (!isspace(*pchPos) && (*pchPos != 0))
      {
         pchPos++;
      }                                 /* endwhile                  */
      *pchPos = 0;
   }                                    /* endif                     */
   return ;
}
```

```
VOID freeYearInfo(PCLIENTDATA pcdData)
{
   PSALESINFO       psiYear;

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                CM_QUERYRECORD,
                                                0,
                                                MPFROM2SHORT
                                                   (CMA_FIRST,
                                                    CMA_ITEMORDER)));

   while (psiYear != NULL)
   {
      if (psiYear->mrcStd.pszIcon != NULL)
      {
         free(psiYear->mrcStd.pszIcon);
      }                                 /* endif                    */
      if (psiYear->pchSales != NULL)
      {
         free(psiYear->pchSales);
      }                                 /* endif                    */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                   CM_QUERYRECORD,
                                                   MPFROMP(psiYear),
                                                   MPFROM2SHORT
                                                      (CMA_NEXT,
                                                       CMA_ITEMORDER)
                                                       ));
   }                                    /* endwhile                 */
   return ;
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCLIENTDATA      pcdData;

   pcdData = (PCLIENTDATA)WinQueryWindowPtr(hwndClient,
                                            0);

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
            ULONG             ulExtra;
            RECORDINSERT      riRecord;
            PSALESINFO        psiYears;
            PSALESINFO        psiCYear;
            USHORT            usIndex;

            pcdData = (PCLIENTDATA)calloc(1,
                                          sizeof(CLIENTDATA));
            if (pcdData == NULL)
            {
               WinAlarm(HWND_DESKTOP,
                        WA_ERROR);
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                             "No memory is available",
                             "Error",
                             0,
                             MB_ICONEXCLAMATION|MB_OK);
               return  MRFROMSHORT(TRUE);
            }                           /* endif                    */
```

```
        WinSetWindowPtr(hwndClient,
                        0,
                        pcdData);

        pcdData->hwndCnr = NULLHANDLE;
        pcdData->hptrIcon = NULLHANDLE;

        pcdData->hwndCnr = WinCreateWindow(hwndClient,
                                           WC_CONTAINER,
                                           "",
                                           CCS_MINIRECORDCORE|
                                              CCS_EXTENDSEL|
                                              WS_VISIBLE,
                                           0,
                                           0,
                                           0,
                                           0,
                                           hwndClient,
                                           HWND_TOP,
                                           WND_CONTAINER,
                                           NULL,
                                           NULL);

        if (pcdData->hwndCnr == NULLHANDLE)
        {
           free(pcdData);
           WinAlarm(HWND_DESKTOP,
                    WA_ERROR);
           WinMessageBox(HWND_DESKTOP,
                         HWND_DESKTOP,
                         "Cannot create container",
                         "Error",
                         0,
                         MB_ICONEXCLAMATION|MB_OK);
           return  MRFROMSHORT(TRUE);
        }                          /* endif                        */

        pcdData->hptrIcon = WinLoadPointer(HWND_DESKTOP,
                                           NULLHANDLE,
                                           ICO_ITEM);

        ulExtra = sizeof(SALESINFO)-sizeof(MINIRECORDCORE);

        riRecord.cb = sizeof(RECORDINSERT);
        riRecord.pRecordOrder = (PRECORDCORE)CMA_END;
        riRecord.fInvalidateRecord = FALSE;
        riRecord.zOrder = CMA_TOP;

        psiYears = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->
           hwndCnr,
                                                      CM_ALLOCRECORD
                                                         ,
                                                         MPFROMLONG
                                                         (ulExtra),
                                                         MPFROMSHORT
                                                         (MAX_YEARS)
                                                         ));

        psiCYear = psiYears;

        for (usIndex = 0; usIndex < MAX_YEARS; usIndex++)
        {
           initSalesInfo(pcdData,
                         psiCYear,
                         usIndex);

           riRecord.pRecordParent = NULL;
           riRecord.cRecordsInsert = 1;
```

```
            WinSendMsg(pcdData->hwndCnr,
                       CM_INSERTRECORD,
                       MPFROMP(psiCYear),
                       MPFROMP(&riRecord));

            psiCYear = (PSALESINFO)
               psiCYear->mrcStd.preccNextRecord;
         }                              /* endfor                    */
         WinSendMsg(hwndClient,
                    WM_COMMAND,
                    MPFROMSHORT(MI_ICON),
                    0);
      }
      break;
   case  WM_DESTROY :
      freeYearInfo(pcdData);

      if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinDestroyWindow(pcdData->hwndCnr);
      }                              /* endif                     */
      if (pcdData->hptrIcon != NULLHANDLE)
      {
         WinDestroyPointer(pcdData->hptrIcon);
      }                              /* endif                     */
      free(pcdData);
      break;
   case  WM_SIZE :
      if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinSetWindowPos(pcdData->hwndCnr,
                         NULLHANDLE,
                         0,
                         0,
                         SHORT1FROMMP(mpParm2),
                         SHORT2FROMMP(mpParm2),
                         SWP_MOVE|SWP_SIZE);
      }                              /* endif                     */
      break;
   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  MI_ICON :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_ICON;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));

               WinSendMsg(pcdData->hwndCnr,
                          CM_ARRANGE,
                          NULL,
                          NULL);
            }
            break;

         case  MI_NAMEFLOWED :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_NAME|CV_FLOW;
```

```
            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));

            WinSendMsg(pcdData->hwndCnr,
                       CM_ARRANGE,
                       NULL,
                       NULL);

         }
         break;

      case  MI_TEXTFLOWED :
         {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_TEXT|CV_FLOW;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));

            WinSendMsg(pcdData->hwndCnr,
                       CM_ARRANGE,
                       NULL,
                       NULL);
         }
         break;

      case  MI_EXIT :
         WinPostMsg(hwndClient,
                    WM_CLOSE,
                    0,
                    0);
         break;

      case  MI_RESUME :
         break;

      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                              /* endswitch                  */
   break;

case  WM_PAINT :
   {
      HPS               hpsPaint;
      RECTL             rclPaint;

      hpsPaint = WinBeginPaint(hwndClient,
                               NULLHANDLE,
                               &rclPaint);
      WinFillRect(hpsPaint,
                  &rclPaint,
                  SYSCLR_WINDOW);
      WinEndPaint(hpsPaint);
   }
   break;

default  :
```

```
        return  WinDefWindowProc(hwndClient,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2);

   }                                  /* endswitch                */
   return  MRFROMSHORT(FALSE);
}
```

## CONTAIN1.RC

```
#include <os2.h>
#include "contain1.h"

ICON ICO_ITEM CONTAIN1.ICO

MENU RES_CLIENT
{
   SUBMENU "~Views", M_VIEWS
   {
      MENUITEM "~Icon", MI_ICON
      MENUITEM "~Name/flowed", MI_NAMEFLOWED
      MENUITEM "Te~xt/flowed", MI_TEXTFLOWED
   }
   SUBMENU "E~xit", M_EXIT
   {
      MENUITEM "E~xit", MI_EXIT
      MENUITEM "~Resume", MI_RESUME
   }
}
```

## CONTAIN1.H

```
#define RES_CLIENT              256
#define WND_CONTAINER           257
#define ICO_ITEM                258
#define M_VIEWS                 320
#define MI_ICON                 321
#define MI_DETAIL               322
#define MI_TREE                 323
#define MI_NAMEFLOWED           324
#define MI_TEXTFLOWED           325
#define M_EXIT                  336
#define MI_EXIT                 337
#define MI_RESUME               338
```

## CONTAIN1.MAK

```
CONTAIN1.EXE:                    CONTAIN1.OBJ \
                                 CONTAIN1.RES
        LINK386 @<<
CONTAIN1
CONTAIN1
CONTAIN1
OS2386
CONTAIN1
<<
        RC CONTAIN1.RES CONTAIN1.EXE
```

```
CONTAIN1.RES:                    CONTAIN1.RC \
                                 CONTAIN1.H
        RC -r CONTAIN1.RC CONTAIN1.RES

CONTAIN1.OBJ:                    CONTAIN1.C \
                                 CONTAIN1.H
        ICC -C+ -Kb+ -Ss+ CONTAIN1.C
```

**CONTAIN1.DEF**

```
NAME  CONTAIN1  WINDOWAPI

DESCRIPTION 'First container example
            Copyright 1992 by Larry Salomon
            All rights reserved.'
STACKSIZE 32768
```

The code should be easy to digest. First, the records are allocated using the CM_ALLOCRECORD structure.

```
psiYears = ( PSALESINFO ) PVOIDFROMMR (
                    WinSendMsg ( pcdData -> hwndCnr,
                                 CM_ALLOCRECORD,
                                 MPFROMLONG ( ulExtra ) ,
                                 MPFROMSHORT ( MAX_YEARS ))) ;
```

Then the allocated records are initialized by calling the *initSalesInfo* function; after each record is initialized, it is inserted using the *riRecord* structure that was initialized earlier.

```
psiCYear = psiYears ;

for ( usIndex = 0 ; usIndex < MAX_YEARS ; usIndex ++ ) {
   initSalesInfo ( pcdData, psiCYear, usIndex ) ;

   riRecord.pRecordParent = NULL ;
   riRecord.cRecordsInsert = 1 ;

   WinSendMsg ( pcdData -> hwndCnr,
                CM_INSERTRECORD,
                MPFROMP ( psiCYear ) ,
                MPFROMP ( &riRecord )) ;

   psiCYear =
     ( PSALESINFO ) psiCYear -> mrcStd.preccNextRecord ;
} /* endfor */
```

It is true that the source code should "practice what we preach" in terms of inserting more than one record at a time to increase performance, but simplicity was deemed more important to allow better understanding of the code.

Finally, the container is switched into icon view by sending ourselves a WM_COMMAND message to simulate the selection of the corresponding menu item.

```
WinSendMsg ( hwndClient,
             WM_COMMAND,
             MPFROMSHORT ( MI_ICON ) ,
             0 ) ;
```

The WM_COMMAND code to switch between container views is rather simple as well. For space reasons, here we present only the code for switching to icon view.

```
case MI_ICON:
   {
      CNRINFO ciInfo ;

      ciInfo.cb = sizeof ( CNRINFO ) ;
      ciInfo.flWindowAttr = CV_ICON ;

      WinSendMsg ( pcdData -> hwndCnr,
                   CM_SETCNRINFO,
                   MPFROMP ( &ciInfo ) ,
                   MPFROMLONG ( CMA_FLWINDOWATTR )) ;

      WinSendMsg ( pcdData -> hwndCnr,
                   CM_ARRANGE,
                   NULL,
                   NULL ) ;
   }
   break ;
```

# Tree View

The tree view is next in the list in order of complexity. It offers three different variations, which are described in Table 23.3.

Table 23.3 Tree View Variations

| View | Description |
|---|---|
| Tree icon view | Objects in the tree are represented by icons or bitmaps with the text to the right. If an item is expandable, a separate bitmap is drawn to the left of the object. This view is specified by adding the CV_ICON and CV_TREE flags to the *flWindowAttr* field. |
| Tree name view | This is the same as the tree icon view except that an object's expandability is shown on the icon or bitmap of the object, and not as a separate bitmap; the TREEITEMDESC structure contains the bitmap or icon handles for both expanded and collapsed views. The caveat here is that the TREEITEMDESC structure is pointed to by the RECORDCORE structure but *not* by the MINIRECORDCORE structure. This view is specified by adding the CV_NAME and CV_TREE flags to the *flWindowAttr* field. |
| Tree text view | Objects in the tree are represented by text only. The feedback on the expandability of an object is represented by a separate bitmap to the left of the text. This view is specified by adding the CV_TEXT and CV_TREE flags to the *flWindowAttr* field. |

In addition to specifying the view type, the amount of space (in pels) for indentation and the thickness of the tree lines may be specified when CA_TREELINE is specified. The indentation and thickness are specified in the *cxTreeIndent* and *cxTreeLine* fields of the CNRINFO structure, respectively. If a value less than 0 is specified for either field, the default for that field is used.

# Details View

The details view is by far the most difficult of the five view types to program, but its ability to show a lot of information at once overshadows this complexity. This view supports the following data types: bitmap/icon, string, unsigned long integer, date, and time. For the latter three, national language support (NLS) is enabled, meaning that the proper thousands separator character is used, the time information is

ordered correctly, and so on. There is no support for decimal types, so any decimals will have to be converted to their string equivalents to display numbers of this type.

The major item of interest when using the details view is the FIELDINFO structure, which describes a single column that is displayed in this view. As with the object records, memory for the FIELDINFO structures is allocated via a message: CM_ALLOCDETAILFIELDINFO. The first parameter specifies the number of FIELDINFO structures to allocate, and a pointer to the first structure is returned. As with CM_ALLOCRECORD, this is the head of a linked list of structures if more than one is allocated and the link to the next record is specified in the *pNextFieldInfo* field.

```
typedef struct _FIELDINFO {
   ULONG cb;
   ULONG flData;
   ULONG flTitle;
   PVOID pTitleData;
   ULONG offStruct;
   PVOID pUserData;
   struct _FIELDINFO *pNextFieldInfo;
   ULONG cxWidth;
} FIELDINFO;
```

*cb* specifies the size of the structure in bytes. *flData* specifies the type of the data in this field and any associated attributes of the column via one or more CFA_ constants listed in Table 23.4.

Table 23.4 CFA_ Constants

| Constant | Description |
|---|---|
| CFA_LEFT | Specifies that the data is to be horizontally aligned left. |
| CFA_RIGHT | Specifies that the data is to be horizontally aligned right. |
| CFA_CENTER | Specifies that the data is to be horizontally centered. |
| CFA_TOP | Specifies that the data is to be vertically aligned top. |
| CFA_VCENTER | Specifies that the data is to be vertically centered. |
| CFA_BOTTOM | Specifies that the data is to be vertically aligned bottom. |
| CFA_INVISIBLE | Specifies that the column is not to be shown. |
| CFA_BITMAPORICON | Specifies that *offStruct* points to a bitmap or icon handle to be displayed in the column, depending on the current setting of *flWindowAttr* in the CNRINFO structure last used to set the container attributes. |
| CFA_SEPARATOR | Specifies that there should be a vertical separator to the right of the column. |
| CFA_HORZSEPARATOR | (*flTitle* only) Specifies that the column title should have a horizontal separator dividing it from the data. |
| CFA_STRING | Specifies that offStruct points to a pointer to a string to be displayed in the column. |
| CFA_OWNER | Specifies that the column is to be owner-drawn. |
| CFA_DATE | Specifies that offStruct points to a CDATE structure. |
| CFA_TIME | Specifies that offStruct points to a CTIME structure. |
| CFA_FIREADONLY | Specifies that the column data should be read-only. |
| CFA_FITITLEREADONLY | (*flTitle* only) Specifies that the column should be read-only. |
| CFA_ULONG | Specifies that *offStruct* points to a ULONG. |

*flTitle* specifies attributes about the heading for this column and is also a combination of CFA_ constants. *pTitleData* points to the column title data; this is a a bitmap or icon if CFA_BITMAPORICON is specified in *flTitle*; otherwise it is a pointer to a string. *offStruct* specifies the offset from the beginning of the

RECORDCORE structure to where the data resides. *pUserData* points to any application-specific data for this column. *pNextFieldInfo* points to the next FIELDINFO structure in the linked list. *cxWidth* specifies the width of the column. If 0, the column will be autosized to be the width of its widest element.

The fields *cb*, *pNextFieldInfo*, and *cxWidth* are initialized by the container in the CM_ALLOCDETAILINFO processing. The application is responsible for initializing the remaining fields.

**Gotcha!**
If *flData* specifies CFA_STRING, then *offStruct* specifies the offset of the *pointer to the text* and not the text itself.

**Gotcha!**
The column heading data is not copied into the container's workspace. Thus they must be global, static, or dynamically allocated data.

**Gotcha!**
A common mistake when specifying CFA_DATE or CFA_TIME for a column is to improperly convert an FDATE structure to a CDATE structure and an FTIME structure to a CTIME structure.

## Splitbars

Details view also provides the option of having a single splitbar between columns. A splitbar is a vertical bar that can be moved with the mouse. This is useful if the data displayed in a column extends beyond the space available. If a splitbar is used, horizontal scrollbars are displayed on the bottom of the container for each subsection bounded by a container edge or a splitbar.

As might be expected, a splitbar is added to the details view using the CM_SETCNRINFO message. The *pFieldInfoLast* and *xVertSplitbar* fields are initialized in the CNRINFO structure. The former points to the FIELDINFO structure to the immediate left of the splitbar; and the latter specifies where the splitbar is to be positioned initially. After initializing these fields, the CM_SETCNRINFO message is sent, specifying CMA_PFIELDINFOLAST | CMA_XVERTSPLITBAR as the second parameter.

The following sample application adds tree and details view to the last sample application. Additionally, it demonstrates the use of a splitbar in the details view.

**CONTAIN2.C**

```
#define  INCL_WINPOINTERS
#define  INCL_WINSTDCNR
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "contain2.h"

#define  CLS_CLIENT     "SampleClass"
#define  MAX_YEARS      10
#define  MAX_MONTHS     12
#define  MAX_COLUMNS    4
#define  CX_SPLITBAR    120
typedef struct _CLIENTDATA
{
   HWND              hwndCnr;

   HPOINTER          hptrIcon;
} CLIENTDATA,*PCLIENTDATA;

typedef struct   _SALESINFO
{
   MINIRECORDCORE   mrcStd;
   ULONG            ulNumUnits;
   float            fSales;
   PCHAR            pchSales;
} SALESINFO,*PSALESINFO;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU|FCF_MENU;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Container Sample",
                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   if (hwndFrame != NULLHANDLE)
```

```
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile                  */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                     */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

VOID initSalesInfo(PCLIENTDATA pcdData,PSALESINFO psiParent,
                   PSALESINFO psiSales,USHORT usIndex)
{
   PCHAR            pchPos;

   psiSales->mrcStd.cb = sizeof(MINIRECORDCORE);

   psiSales->mrcStd.pszIcon = malloc(256);
   if (psiSales->mrcStd.pszIcon != NULL)
   {
      if (psiParent != NULL)
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Month %d",
                 usIndex+1);
      }
      else
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Year 19%02d",
                 usIndex+84);
      }                                  /* endif                     */
   }                                     /* endif                     */
   psiSales->mrcStd.hptrIcon = pcdData->hptrIcon;

   if (psiParent != NULL)
   {
      psiSales->ulNumUnits = psiParent->ulNumUnits/12;
   }
   else
   {
      psiSales->ulNumUnits = usIndex *usIndex;
   }                                     /* endif                     */
   psiSales->fSales = (float)psiSales->ulNumUnits *9.95;

   psiSales->pchSales = malloc(16);
   if (psiSales->pchSales != NULL)
   {
      sprintf(psiSales->pchSales,
              "$%-10.2f",
              psiSales->fSales);

      pchPos = psiSales->pchSales;
      while (!isspace(*pchPos) && (*pchPos != 0))
```

```
      {
         pchPos++;
      }                                  /* endwhile                  */
      *pchPos = 0;
   }                                     /* endif                     */
   return ;
}

VOID freeYearInfo(PCLIENTDATA pcdData)
{
   PSALESINFO       psiYear;
   PSALESINFO       psiMonth;

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                CM_QUERYRECORD,
                                                0,
                                                MPFROM2SHORT
                                                   (CMA_FIRST,
                                                    CMA_ITEMORDER)));
   while (psiYear != NULL)
   {
      psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                    CM_QUERYRECORD,
                                                    MPFROMP(psiYear),
                                                    MPFROM2SHORT
                                                    (CMA_FIRSTCHILD,
                                                     CMA_ITEMORDER)))
         ;
      while (psiMonth != NULL)
      {
         if (psiMonth->mrcStd.pszIcon != NULL)
         {
            free(psiMonth->mrcStd.pszIcon);
         }                               /* endif                     */
         if (psiMonth->pchSales != NULL)
         {
            free(psiMonth->pchSales);
         }                               /* endif                     */
         psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg
            (pcdData->hwndCnr,
             CM_QUERYRECORD,
             MPFROMP(psiMonth),
             MPFROM2SHORT(CMA_NEXT,
                          CMA_ITEMORDER)));
      }                                  /* endwhile                  */
      if (psiYear->mrcStd.pszIcon != NULL)
      {
         free(psiYear->mrcStd.pszIcon);
      }                                  /* endif                     */
      if (psiYear->pchSales != NULL)
      {
         free(psiYear->pchSales);
      }                                  /* endif                     */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                   CM_QUERYRECORD,
                                                   MPFROMP(psiYear),
                                                   MPFROM2SHORT
                                                      (CMA_NEXT,
                                                       CMA_ITEMORDER)
                                                       ));
   }                                     /* endwhile                  */
   return ;
}

VOID initColumns(PCLIENTDATA pcdData)
{
   CNRINFO          ciInfo;
   PFIELDINFO       pfiCurrent;
   PFIELDINFO       pfiInfo;
```

```
PFIELDINFO       pfiLefty;
FIELDINFOINSERT  fiiInfo;

pfiInfo = (PFIELDINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                             CM_ALLOCDETAILFIELDINFO
                                                ,
                                               MPFROMLONG
                                               (MAX_COLUMNS),
                                               0));

pfiCurrent = pfiInfo;

pfiCurrent->flData = CFA_BITMAPORICON|CFA_HORZSEPARATOR|CFA_CENTER
   |CFA_SEPARATOR;

pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
pfiCurrent->pTitleData = "Icon";
pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                    mrcStd.hptrIcon);

pfiCurrent = pfiCurrent->pNextFieldInfo;
pfiCurrent->flData = CFA_STRING|CFA_CENTER|CFA_HORZSEPARATOR;

pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
pfiCurrent->pTitleData = "Year";
pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                    mrcStd.pszIcon);

pfiLefty = pfiCurrent;

pfiCurrent = pfiCurrent->pNextFieldInfo;
pfiCurrent->flData = CFA_ULONG|CFA_CENTER|CFA_HORZSEPARATOR|
   CFA_SEPARATOR;

pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
pfiCurrent->pTitleData = "Units Sold";
pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                    ulNumUnits);

pfiCurrent = pfiCurrent->pNextFieldInfo;
pfiCurrent->flData = CFA_STRING|CFA_RIGHT|CFA_HORZSEPARATOR;

pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
pfiCurrent->pTitleData = "Sales";
pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                    pchSales);

fiiInfo.cb = sizeof(fiiInfo);
fiiInfo.pFieldInfoOrder = (PFIELDINFO)CMA_FIRST;
fiiInfo.cFieldInfoInsert = MAX_COLUMNS;
fiiInfo.fInvalidateFieldInfo = TRUE;

WinSendMsg(pcdData->hwndCnr,
           CM_INSERTDETAILFIELDINFO,
           MPFROMP(pfiInfo),
           MPFROMP(&fiiInfo));

memset(&ciInfo,
       0,
       sizeof(ciInfo));
ciInfo.cb = sizeof(CNRINFO);
ciInfo.pFieldInfoLast = pfiLefty;
ciInfo.xVertSplitbar = CX_SPLITBAR;

WinSendMsg(pcdData->hwndCnr,
           CM_SETCNRINFO,
           MPFROMP(&ciInfo),
```

```
            MPFROMLONG(CMA_PFIELDINFOLAST|CMA_XVERTSPLITBAR));

   return ;
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCLIENTDATA      pcdData;

   pcdData = (PCLIENTDATA)WinQueryWindowPtr(hwndClient,
                                            0);

   switch (ulMsg)
   {

      case  WM_CREATE :
         {
            ULONG             ulExtra;
            RECORDINSERT      riRecord;
            PSALESINFO        psiYears;
            PSALESINFO        psiCYear;
            USHORT            usIndex1;
            PSALESINFO        psiMonths;
            PSALESINFO        psiCMonth;
            USHORT            usIndex2;

            pcdData = (PCLIENTDATA)malloc(sizeof(CLIENTDATA));

            if (pcdData == NULL)
            {
               WinAlarm(HWND_DESKTOP,
                        WA_ERROR);
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                             "No memory is available",
                             "Error",
                             0,
                             MB_ICONEXCLAMATION|MB_OK);
               return  MRFROMSHORT(TRUE);
            }                           /* endif                    */

            WinSetWindowPtr(hwndClient,
                            0,
                            pcdData);

            pcdData->hwndCnr = NULLHANDLE;
            pcdData->hptrIcon = NULLHANDLE;

            pcdData->hwndCnr = WinCreateWindow(hwndClient,
                                               WC_CONTAINER,
                                               "",
                                               CCS_MINIRECORDCORE|
                                                  CCS_EXTENDSEL|
                                                  WS_VISIBLE,
                                               0,
                                               0,
                                               0,
                                               0,
                                               hwndClient,
                                               HWND_TOP,
                                               WND_CONTAINER,
                                               NULL,
                                               NULL);

            if (pcdData->hwndCnr == NULLHANDLE)
            {
               free(pcdData);
```

```
      WinAlarm(HWND_DESKTOP,
               WA_ERROR);
      WinMessageBox(HWND_DESKTOP,
                    HWND_DESKTOP,
                    "Cannot create container",
                    "Error",
                    0,
                    MB_ICONEXCLAMATION|MB_OK);
      return  MRFROMSHORT(TRUE);
   }                          /* endif                    */

   pcdData->hptrIcon = WinLoadPointer(HWND_DESKTOP,
                                      NULLHANDLE,
                                      ICO_ITEM);

   ulExtra = sizeof(SALESINFO)-sizeof(MINIRECORDCORE);

   riRecord.cb = sizeof(RECORDINSERT);
   riRecord.pRecordOrder = (PRECORDCORE)CMA_END;
   riRecord.fInvalidateRecord = FALSE;
   riRecord.zOrder = CMA_TOP;

   psiYears = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->
      hwndCnr,
                                                 CM_ALLOCRECORD
                                                    ,
                                                    MPFROMLONG
                                                    (ulExtra),
                                                    MPFROMSHORT
                                                    (MAX_YEARS)
                                                    ));

   psiCYear = psiYears;

   for (usIndex1 = 0; usIndex1 < MAX_YEARS; usIndex1++)
   {
      initSalesInfo(pcdData,
                    NULL,
                    psiCYear,
                    usIndex1);

      riRecord.pRecordParent = NULL;
      riRecord.cRecordsInsert = 1;

      WinSendMsg(pcdData->hwndCnr,
                 CM_INSERTRECORD,
                 MPFROMP(psiCYear),
                 MPFROMP(&riRecord));

      psiMonths = (PSALESINFO)PVOIDFROMMR(WinSendMsg
         (pcdData->hwndCnr,
          CM_ALLOCRECORD,
          MPFROMLONG(ulExtra),
          MPFROMSHORT(MAX_MONTHS)));

      psiCMonth = psiMonths;

      for (usIndex2 = 0; usIndex2 < MAX_MONTHS; usIndex2++)
      {

         initSalesInfo(pcdData,
                       psiCYear,
                       psiCMonth,
                       usIndex2);

         psiCMonth = (PSALESINFO)
            psiCMonth->mrcStd.preccNextRecord;
```

```
            }                           /* endfor                    */
            riRecord.pRecordParent = (PRECORDCORE)psiCYear;
            riRecord.cRecordsInsert = MAX_MONTHS;

            WinSendMsg(pcdData->hwndCnr,
                       CM_INSERTRECORD,
                       MPFROMP(psiMonths),
                       MPFROMP(&riRecord));

            psiCYear = (PSALESINFO)
               psiCYear->mrcStd.preccNextRecord;
         }                              /* endfor                    */
         initColumns(pcdData);

         WinSendMsg(hwndClient,
                    WM_COMMAND,
                    MPFROMSHORT(MI_ICON),
                    0);
      }
      break;

   case  WM_DESTROY :
      freeYearInfo(pcdData);

      if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinDestroyWindow(pcdData->hwndCnr);
      }                                 /* endif                     */
      if (pcdData->hptrIcon != NULLHANDLE)
      {
         WinDestroyPointer(pcdData->hptrIcon);
      }                                 /* endif                     */
      free(pcdData);
      break;

   case  WM_SIZE :
      if (pcdData->hwndCnr != NULLHANDLE)
      {

         WinSetWindowPos(pcdData->hwndCnr,
                         NULLHANDLE,
                         0,
                         0,
                         SHORT1FROMMP(mpParm2),
                         SHORT2FROMMP(mpParm2),
                         SWP_MOVE|SWP_SIZE);
      }                                 /* endif                     */
      break;

   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {

         case  MI_ICON :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_ICON;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));

               WinSendMsg(pcdData->hwndCnr,
                          CM_ARRANGE,
                          NULL,
                          NULL);
```

```
        }
        break;

    case  MI_DETAIL :
        {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_DETAIL|
               CA_DETAILSVIEWTITLES;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));
        }
        break;

    case  MI_TREE :
        {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_TREE|CV_ICON|CA_TREELINE;
            ciInfo.cxTreeIndent = -1;
            ciInfo.cxTreeLine = -1;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));
        }
        break;

    case  MI_NAMEFLOWED :
        {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_NAME|CV_FLOW;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));

            WinSendMsg(pcdData->hwndCnr,
                       CM_ARRANGE,
                       NULL,
                       NULL);
        }
        break;

    case  MI_TEXTFLOWED :
        {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_TEXT|CV_FLOW;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));
```

```
                WinSendMsg(pcdData->hwndCnr,
                           CM_ARRANGE,
                           NULL,
                           NULL);
             }
             break;

          case  MI_EXIT :
             WinPostMsg(hwndClient,
                        WM_CLOSE,
                        0,
                        0);
             break;

          case  MI_RESUME :
             break;

          default  :
             return  WinDefWindowProc(hwndClient,
                                      ulMsg,
                                      mpParm1,
                                      mpParm2);

       }                                 /* endswitch                */
       break;

    case  WM_PAINT :
       {
          HPS              hpsPaint;
          RECTL            rclPaint;

          hpsPaint = WinBeginPaint(hwndClient,
                                   NULLHANDLE,
                                   &rclPaint);

          WinFillRect(hpsPaint,
                      &rclPaint,
                      SYSCLR_WINDOW);
          WinEndPaint(hpsPaint);
       }
       break;

    default  :
       return  WinDefWindowProc(hwndClient,
                                ulMsg,
                                mpParm1,
                                mpParm2);

  }                                      /* endswitch                */
  return  MRFROMSHORT(FALSE);
}
```

**CONTAIN2.RC**

```
#include <os2.h>
#include "contain2.h"

ICON ICO_ITEM CONTAIN2.ICO

MENU RES_CLIENT
{
   SUBMENU "~Views", M_VIEWS
   {
      MENUITEM "~Icon", MI_ICON
      MENUITEM "~Detail", MI_DETAIL
      MENUITEM "~Tree", MI_TREE
```

```
      MENUITEM "~Name/flowed", MI_NAMEFLOWED
      MENUITEM "Te~xt/flowed", MI_TEXTFLOWED
   }
   SUBMENU "E~xit", M_EXIT
   {
      MENUITEM "E~xit", MI_EXIT
      MENUITEM "~Resume", MI_RESUME
   }
}
```

**CONTAIN2.H**

```
#define RES_CLIENT              256
#define WND_CONTAINER           257
#define ICO_ITEM                258
#define M_VIEWS                 320
#define MI_ICON                 321
#define MI_DETAIL               322
#define MI_TREE                 323
#define MI_NAMEFLOWED           324
#define MI_TEXTFLOWED           325
#define M_EXIT                  336
#define MI_EXIT                 337
#define MI_RESUME               338
```

**CONTAIN2.MAK**

```
CONTAIN2.EXE:                    CONTAIN2.OBJ \
                                 CONTAIN2.RES
        LINK386 @<<
CONTAIN2
CONTAIN2
CONTAIN2
OS2386
CONTAIN2
<<
        RC CONTAIN2.RES CONTAIN2.EXE

CONTAIN2.RES:                    CONTAIN2.RC \
                                 CONTAIN2.H
        RC -r CONTAIN2.RC CONTAIN2.RES

CONTAIN2.OBJ:                    CONTAIN2.C \
                                 CONTAIN2.H
        ICC -C+ -Kb+ -Ss+ CONTAIN2.C
```

**CONTAIN2.DEF**

```
NAME  CONTAIN2  WINDOWAPI

DESCRIPTION 'Second container example
            Copyright 1992 by Larry Salomon
            All rights reserved.'
STACKSIZE 32768
```

As before, we allocate a number of records using the CM_ALLOCRECORD structure; within the loop to initialize each record (which represents a year of sales figures), we allocate 12 records to represent each month, initialize these records, and insert them into the container, specifying the year record previously

inserted as the parent. This establishes a hierarchical structure that we may observe by placing the container in tree view.

```
psiMonths = ( PSALESINFO ) PVOIDFROMMR (
                    WinSendMsg ( pcdData -> hwndCnr,
                                 CM_ALLOCRECORD,
                                 MPFROMLONG ( ulExtra ) ,
                                 MPFROMSHORT ( MAX_MONTHS ))) ;
psiCMonth = psiMonths ;
for ( usIndex2 = 0 ; usIndex2 < MAX_MONTHS ;
         usIndex2 ++ ) {
   initSalesInfo ( pcdData,
                   psiCYear,
                   psiCMonth,
                   usIndex2 ) ;

   psiCMonth =
      ( PSALESINFO ) psiCMonth->mrcStd.preccNextRecord;
} /* endfor */

riRecord.pRecordParent = ( PRECORDCORE ) psiCYear ;
riRecord.cRecordsInsert = MAX_MONTHS ;

WinSendMsg ( pcdData -> hwndCnr,
             CM_INSERTRECORD,
             MPFROMP ( psiMonths ) ,
             MPFROMP ( &riRecord )) ;
```

Finally, we call *initColumns* to set up the detail view. It allocates a fixed number of FIELDINFO structures by sending a CM_ALLOCDETAILFIELDINFO message to the container.

```
pfiInfo = ( PFIELDINFO ) PVOIDFROMMR (
             WinSendMsg ( pcdData -> hwndCnr,
                          CM_ALLOCDETAILFIELDINFO,
                          MPFROMLONG ( MAX_COLUMNS ) ,
                          0 )) ;
```

Each FIELDINFO structure is then initialized, and then all of the FIELDINFO structures are inserted.

```
pfiCurrent -> flData = CFA_BITMAPORICON |
                       CFA_HORZSEPARATOR |
                       CFA_CENTER |
                       CFA_SEPARATOR ;

pfiCurrent -> flTitle = CFA_STRING | CFA_CENTER ;
pfiCurrent -> pTitleData = "Icon" ;
pfiCurrent -> offStruct = FIELDOFFSET ( SALESINFO,
                                        mrcStd.hptrIcon ) ;

pfiCurrent = pfiCurrent -> pNextFieldInfo ;
   :
   :
fiiInfo.cb = sizeof ( fiiInfo ) ;
fiiInfo.pFieldInfoOrder = ( PFIELDINFO ) CMA_FIRST ;
fiiInfo.cFieldInfoInsert = MAX_COLUMNS ;
fiiInfo.fInvalidateFieldInfo = TRUE ;

WinSendMsg ( pcdData -> hwndCnr,
             CM_INSERTDETAILFIELDINFO,
             MPFROMP ( pfiInfo ) ,
             MPFROMP ( &fiiInfo )) ;
```

Finally, the splitbar is initialized by sending the CM_SETCNRINFO message.

```
memset ( &ciInfo, 0, sizeof ( ciInfo )) ;
ciInfo.cb = sizeof ( CNRINFO ) ;
ciInfo.pFieldInfoLast = pfiLefty ;
ciInfo.xVertSplitbar = CX_SPLITBAR ;

WinSendMsg ( pcdData -> hwndCnr,
             CM_SETCNRINFO,
             MPFROMP ( &ciInfo ) ,
             MPFROMLONG ( CMA_PFIELDINFOLAST |
                          CMA_XVERTSPLITBAR )) ;
```

## Of Emphasis and Pop-ups

Object emphasis is a visual cue to the user that something about the object is different from the norm. *Cursored*, *selected*, *in-use*, and *source* emphasis are the four types defined by the container. Of these four types, defined in Table 23.5, only the first two are set automatically by the container. The latter two must be explicitly set by the application via the CM_SETRECORDEMPHASIS message.

Table 23.5 Emphasis Types

| Emphasis | Description |
|---|---|
| Cursored | Set whenever the input focus belongs to the object. This is shown as a dotted-line rectangle around the object. |
| Selected | Set whenever the object was selected using the mouse button or the spacebar. The selection style of the container how records previously selected behave when a new record is selected. This is shown as an inverted background around the object. |
| In-use | Set whenever the object is defined to be in use by the application. This is shown as a crosshatch pattern in the background of the object. |
| Source | Set whenever the object is a source of some action. This record also could be in the selected state, but doing so is not required. This is shown as a dashed-line rectangle with rounded corners around the object. |

**Gotcha!**
While the CM_SETRECORDEMPHASIS has CRA_ constants for the cursored, selected, and in-use emphasis types, the Toolkit header files in OS/2 2.x are missing the source emphasis constant. CRA_SOURCE should be defined as 0x00004000L. This has been fixed in the OS/2 Warp Toolkit. It also should be noted that the entire container can have source emphasis, in which case NULL should be specified for the record pointer.

The following sample removes the action bar from the window and instead uses pop-up menus to provide the actions available to the user.

**CONTAIN3.C**

```
#define  INCL_WINFRAMEMGR
#define  INCL_WINMENUS
#define  INCL_WINPOINTERS
#define  INCL_WINSTDCNR
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "CONTAIN3.H"
#define  CLS_CLIENT     "SampleClass"
#define  MAX_YEARS      10
#define  MAX_MONTHS     12
#define  MAX_COLUMNS    4
#define  CX_SPLITBAR    120
//------------------------------------------------------
// For the GA 2.0 toolkit, CRA_SOURCE is not defined,
// but it should be.
//------------------------------------------------------

#ifndef  CRA_SOURCE
#define  CRA_SOURCE     0x00004000L
#endif
typedef struct _CLIENTDATA
{
   HWND             hwndCnr;

   HPOINTER         hptrIcon;
   HWND             hwndMenu;
   BOOL             bCnrSelected;
} CLIENTDATA,*PCLIENTDATA;

typedef struct   _SALESINFO
{
   MINIRECORDCORE   mrcStd;
   BOOL             bEmphasized;
   ULONG            ulNumUnits;
   float            fSales;
   PCHAR            pchSales;
} SALESINFO,*PSALESINFO;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU;
```

```
   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Container Sample",
                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                 /* endwhile                  */
      WinDestroyWindow(hwndFrame);
   }                                    /* endif                     */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

VOID initSalesInfo(PCLIENTDATA pcdData,PSALESINFO psiParent,
                   PSALESINFO psiSales,USHORT usIndex)
{
   PCHAR            pchPos;

   psiSales->mrcStd.cb = sizeof(MINIRECORDCORE);

   psiSales->mrcStd.pszIcon = malloc(256);
   if (psiSales->mrcStd.pszIcon != NULL)
   {

      if (psiParent != NULL)
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Month %d",
                 usIndex+1);
      }
      else
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Year 19%02d",
                 usIndex+84);
      }                                 /* endif                     */
   }                                    /* endif                     */
   psiSales->mrcStd.hptrIcon = pcdData->hptrIcon;
   psiSales->bEmphasized = FALSE;

   if (psiParent != NULL)
   {
      psiSales->ulNumUnits = psiParent->ulNumUnits/12;
   }
   else
   {
```

```
      psiSales->ulNumUnits = usIndex *100;
   }                                    /* endif                    */
   psiSales->fSales = (float)psiSales->ulNumUnits *9.95;

   psiSales->pchSales = malloc(16);

   if (psiSales->pchSales != NULL)
   {
      sprintf(psiSales->pchSales,
              "$%-10.2f",
              psiSales->fSales);

      pchPos = psiSales->pchSales;
      while (!isspace(*pchPos) && (*pchPos != 0))
      {
         pchPos++;
      }                                 /* endwhile                 */
      *pchPos = 0;
   }                                    /* endif                    */
   return ;
}

VOID emphasizeRecs(HWND hwndCnr,BOOL bEmphasize)
{
   SHORT             sFlag;
   PSALESINFO        psiYear;

   sFlag = ((bEmphasize)?CRA_SELECTED:CRA_SOURCE);

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                CM_QUERYRECORDEMPHASIS
                                                    ,
                                                MPFROMP(CMA_FIRST),
                                                MPFROMSHORT(sFlag)));

   while (psiYear != NULL)
   {
      if (bEmphasize)
      {
         WinSendMsg(hwndCnr,
                    CM_SETRECORDEMPHASIS,
                    MPFROMP(psiYear),
                    MPFROM2SHORT(TRUE,
                                 CRA_SOURCE));

         psiYear->bEmphasized = TRUE;
      }
      else
      {
         WinSendMsg(hwndCnr,
                    CM_SETRECORDEMPHASIS,
                    MPFROMP(psiYear),
                    MPFROM2SHORT(FALSE,
                                 CRA_SOURCE));

         psiYear->bEmphasized = FALSE;
      }                                 /* endif                    */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                   CM_QUERYRECORDEMPHASIS
                                                       ,
                                                   MPFROMP(psiYear),
                                                   MPFROMSHORT(sFlag)
                                                   ));
   }                                    /* endwhile                 */
   return ;
}

VOID freeCnrInfo(HWND hwndCnr)
{
```

```
   PSALESINFO       psiYear;
   PSALESINFO       psiMonth;

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                CM_QUERYRECORD,
                                                0,
                                                MPFROM2SHORT
                                                   (CMA_FIRST,
                                                    CMA_ITEMORDER)));
   while (psiYear != NULL)
   {
      psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                    CM_QUERYRECORD,
                                                    MPFROMP(psiYear),
                                                    MPFROM2SHORT
                                                    (CMA_FIRSTCHILD,
                                                     CMA_ITEMORDER)))
         ;
      while (psiMonth != NULL)
      {
         if (psiMonth->mrcStd.pszIcon != NULL)
         {
            free(psiMonth->mrcStd.pszIcon);
         }                                /* endif                    */
         if (psiMonth->pchSales != NULL)
         {
            free(psiMonth->pchSales);
         }                                /* endif                    */
         psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                       CM_QUERYRECORD
                                                          ,
                                                       MPFROMP
                                                       (psiMonth),
                                                       MPFROM2SHORT
                                                       (CMA_NEXT,
                                                        CMA_ITEMORDER
                                                        )));
      }                                   /* endwhile                 */
      if (psiYear->mrcStd.pszIcon != NULL)
      {
         free(psiYear->mrcStd.pszIcon);
      }                                   /* endif                    */
      if (psiYear->pchSales != NULL)
      {
         free(psiYear->pchSales);
      }                                   /* endif                    */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                   CM_QUERYRECORD,
                                                   MPFROMP(psiYear),
                                                   MPFROM2SHORT
                                                      (CMA_NEXT,
                                                       CMA_ITEMORDER)
                                                       ));
   }                                      /* endwhile                 */
   return ;
}

VOID initColumns(PCLIENTDATA pcdData)
{
   CNRINFO          ciInfo;
   PFIELDINFO       pfiCurrent;
   PFIELDINFO       pfiInfo;
   PFIELDINFO       pfiLefty;
   FIELDINFOINSERT  fiiInfo;
```

```
   pfiInfo = (PFIELDINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                CM_ALLOCDETAILFIELDINFO
                                                      ,
                                                MPFROMLONG
                                                (MAX_COLUMNS),
                                                0));

   pfiCurrent = pfiInfo;

   pfiCurrent->flData = CFA_BITMAPORICON|CFA_HORZSEPARATOR|CFA_CENTER
      |CFA_SEPARATOR;

   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Icon";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       mrcStd.hptrIcon);

   pfiCurrent = pfiCurrent->pNextFieldInfo;

   pfiCurrent->flData = CFA_STRING|CFA_CENTER|CFA_HORZSEPARATOR;

   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Year";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       mrcStd.pszIcon);

   pfiLefty = pfiCurrent;

   pfiCurrent = pfiCurrent->pNextFieldInfo;
   pfiCurrent->flData = CFA_ULONG|CFA_CENTER|CFA_HORZSEPARATOR|
      CFA_SEPARATOR;

   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Units Sold";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       ulNumUnits);

   pfiCurrent = pfiCurrent->pNextFieldInfo;
   pfiCurrent->flData = CFA_STRING|CFA_RIGHT|CFA_HORZSEPARATOR;

   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Sales";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       pchSales);

   fiiInfo.cb = sizeof(fiiInfo);
   fiiInfo.pFieldInfoOrder = (PFIELDINFO)CMA_FIRST;
   fiiInfo.cFieldInfoInsert = MAX_COLUMNS;
   fiiInfo.fInvalidateFieldInfo = TRUE;

   WinSendMsg(pcdData->hwndCnr,
              CM_INSERTDETAILFIELDINFO,
              MPFROMP(pfiInfo),
              MPFROMP(&fiiInfo));

   memset(&ciInfo,
          0,
          sizeof(ciInfo));
   ciInfo.cb = sizeof(CNRINFO);
   ciInfo.pFieldInfoLast = pfiLefty;
   ciInfo.xVertSplitbar = CX_SPLITBAR;

   WinSendMsg(pcdData->hwndCnr,
              CM_SETCNRINFO,
              MPFROMP(&ciInfo),
              MPFROMLONG(CMA_PFIELDINFOLAST|CMA_XVERTSPLITBAR));
   return ;
}
```

```
MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCLIENTDATA      pcdData;

   pcdData = (PCLIENTDATA)WinQueryWindowPtr(hwndClient,
                                            0);

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
            MENUITEM          miItem;
            ULONG             ulStyle;
            ULONG             ulExtra;
            RECORDINSERT      riRecord;
            PSALESINFO        psiYears;
            PSALESINFO        psiCYear;
            USHORT            usIndex1;
            PSALESINFO        psiMonths;
            PSALESINFO        psiCMonth;
            USHORT            usIndex2;

            pcdData = (PCLIENTDATA)malloc(sizeof(CLIENTDATA));
            if (pcdData == NULL)
            {
               WinAlarm(HWND_DESKTOP,
                        WA_ERROR);
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                             "No memory is available",
                             "Error",
                             0,
                             MB_ICONEXCLAMATION|MB_OK);
               return  MRFROMSHORT(TRUE);
            }                           /* endif                    */

            WinSetWindowPtr(hwndClient,
                            0,
                            pcdData);

            pcdData->hwndCnr = NULLHANDLE;
            pcdData->hptrIcon = NULLHANDLE;
            pcdData->hwndMenu = NULLHANDLE;
            pcdData->bCnrSelected = FALSE;

            pcdData->hwndCnr = WinCreateWindow(hwndClient,
                                               WC_CONTAINER,
                                               "",
                                               CCS_MINIRECORDCORE|
                                                  CCS_EXTENDSEL|
                                                  WS_VISIBLE,
                                               0,
                                               0,
                                               0,
                                               0,
                                               hwndClient,
                                               HWND_TOP,
                                               WND_CONTAINER,
                                               NULL,
                                               NULL);

            if (pcdData->hwndCnr == NULLHANDLE)
            {
               free(pcdData);
```

```
    WinAlarm(HWND_DESKTOP,
             WA_ERROR);
    WinMessageBox(HWND_DESKTOP,
                  HWND_DESKTOP,
                  "Cannot create container",
                  "Error",
                  0,
                  MB_ICONEXCLAMATION|MB_OK);
    return  MRFROMSHORT(TRUE);
}                            /* endif                        */

pcdData->hptrIcon = WinLoadPointer(HWND_DESKTOP,
                                   NULLHANDLE,
                                   ICO_ITEM);

pcdData->hwndMenu = WinLoadMenu(hwndClient,
                                NULLHANDLE,
                                RES_CLIENT);
WinSendMsg(pcdData->hwndMenu,
           MM_QUERYITEM,
           MPFROM2SHORT(M_VIEWS,
                        TRUE),
           MPFROMP(&miItem));

ulStyle = WinQueryWindowULong(miItem.hwndSubMenu,
                              QWL_STYLE);

ulStyle |= MS_CONDITIONALCASCADE;
WinSetWindowULong(miItem.hwndSubMenu,
                  QWL_STYLE,
                  ulStyle);

WinSendMsg(miItem.hwndSubMenu,
           MM_SETDEFAULTITEMID,
           MPFROMSHORT(MI_ICON),
           0);

ulExtra = sizeof(SALESINFO)-sizeof(MINIRECORDCORE);

riRecord.cb = sizeof(RECORDINSERT);
riRecord.pRecordOrder = (PRECORDCORE)CMA_END;
riRecord.fInvalidateRecord = FALSE;
riRecord.zOrder = CMA_TOP;

psiYears = (PSALESINFO)PVOIDFROMMR(WinSendMsg(pcdData->
   hwndCnr,
                                                CM_ALLOCRECORD
                                                   ,
                                                   MPFROMLONG
                                                   (ulExtra),
                                                   MPFROMSHORT
                                                   (MAX_YEARS)
                                                   ));

psiCYear = psiYears;

for (usIndex1 = 0; usIndex1 < MAX_YEARS; usIndex1++)
{

   initSalesInfo(pcdData,
                 NULL,
                 psiCYear,
                 usIndex1);

   riRecord.pRecordParent = NULL;
   riRecord.cRecordsInsert = 1;
```

```
            WinSendMsg(pcdData->hwndCnr,
                       CM_INSERTRECORD,
                       MPFROMP(psiCYear),
                       MPFROMP(&riRecord));

            psiMonths = (PSALESINFO)PVOIDFROMMR(WinSendMsg
               (pcdData->hwndCnr,
                CM_ALLOCRECORD,
                MPFROMLONG(ulExtra),
                MPFROMSHORT(MAX_MONTHS)));

            psiCMonth = psiMonths;

            for (usIndex2 = 0; usIndex2 < MAX_MONTHS; usIndex2++)
            {

               initSalesInfo(pcdData,
                             psiCYear,
                             psiCMonth,
                             usIndex2);

               psiCMonth = (PSALESINFO)
                  psiCMonth->mrcStd.preccNextRecord;
            }                          /* endfor                    */
            riRecord.pRecordParent = (PRECORDCORE)psiCYear;
            riRecord.cRecordsInsert = MAX_MONTHS;

            WinSendMsg(pcdData->hwndCnr,
                       CM_INSERTRECORD,
                       MPFROMP(psiMonths),
                       MPFROMP(&riRecord));

            psiCYear = (PSALESINFO)
               psiCYear->mrcStd.preccNextRecord;
         }                             /* endfor                    */
         initColumns(pcdData);

         WinSendMsg(hwndClient,
                    WM_COMMAND,
                    MPFROMSHORT(MI_ICON),
                    0);
      }
      break;

   case  WM_DESTROY :
      freeCnrInfo(pcdData->hwndCnr);

      if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinDestroyWindow(pcdData->hwndCnr);
      }                             /* endif                        */
      if (pcdData->hptrIcon != NULLHANDLE)
      {
         WinDestroyPointer(pcdData->hptrIcon);
      }                             /* endif                        */
      free(pcdData);
      break;

   case  WM_SIZE :
      if (pcdData->hwndCnr != NULLHANDLE)
      {

         WinSetWindowPos(pcdData->hwndCnr,
                         NULLHANDLE,
                         0,
                         0,
                         SHORT1FROMMP(mpParm2),
```

```
                        SHORT2FROMMP(mpParm2),
                        SWP_MOVE|SWP_SIZE);
      }                                 /* endif                    */
      break;

   case  WM_MENUEND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  FID_MENU :
            if (pcdData->bCnrSelected)
            {
               WinSendMsg(pcdData->hwndCnr,
                          CM_SETRECORDEMPHASIS,
                          0,
                          MPFROM2SHORT(FALSE,
                                       CRA_SOURCE));
               pcdData->bCnrSelected = FALSE;
            }
            else
            {
               emphasizeRecs(pcdData->hwndCnr,
                             FALSE);
            }                           /* endif                    */
            break;

         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

      }                                 /* endswitch                */
      break;

   case  WM_CONTROL :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  WND_CONTAINER :
            switch (SHORT2FROMMP(mpParm1))
            {
               case  CN_CONTEXTMENU :
                  {
                     PSALESINFO       psiSales;
                     POINTL           ptlMouse;

                     psiSales = (PSALESINFO)PVOIDFROMMP(mpParm2);
                     if (psiSales != NULL)
                     {
                        if ((psiSales->mrcStd.flRecordAttr
                           &CRA_SELECTED) == 0)
                        {
                           WinSendMsg(pcdData->hwndCnr,
                                      CM_SETRECORDEMPHASIS,
                                      MPFROMP(psiSales),
                                      MPFROM2SHORT(TRUE,
                                                   CRA_SOURCE));
                           psiSales->bEmphasized = TRUE;
                        }
                        else
                        {
                           emphasizeRecs(pcdData->hwndCnr,
                                         TRUE);
                        }               /* endif                    */
                     }
                     else
                     {
```

```
                        WinSendMsg(pcdData->hwndCnr,
                                   CM_SETRECORDEMPHASIS,
                                   0,
                                   MPFROM2SHORT(TRUE,
                                                CRA_SOURCE));
                        pcdData->bCnrSelected = TRUE;
                     }              /* endif                        */
                     WinQueryPointerPos(HWND_DESKTOP,
                                        &ptlMouse);
                     WinMapWindowPoints(HWND_DESKTOP,
                                        hwndClient,
                                        &ptlMouse,
                                        1);
                  WinPopupMenu(hwndClient,
                               hwndClient,
                               pcdData->hwndMenu,
                               ptlMouse.x,
                               ptlMouse.y,
                               M_VIEWS,
                               PU_HCONSTRAIN|PU_VCONSTRAIN|
                                  PU_KEYBOARD|PU_MOUSEBUTTON1|
                                  PU_MOUSEBUTTON2|PU_NONE);
                  }
                  break;

               default  :
                  return  WinDefWindowProc(hwndClient,
                                           ulMsg,
                                           mpParm1,
                                           mpParm2);

            }                       /* endswitch                    */
            break;

         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

      }                             /* endswitch                    */
      break;

   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  MI_ICON :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_ICON;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));

               WinSendMsg(pcdData->hwndCnr,
                          CM_ARRANGE,
                          NULL,
                          NULL);
            }
            break;

         case  MI_DETAIL :
            {
```

```
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_DETAIL|
               CA_DETAILSVIEWTITLES;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));
         }
         break;

      case  MI_TREE :
         {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_TREE|CV_ICON|CA_TREELINE;
            ciInfo.cxTreeIndent = -1;
            ciInfo.cxTreeLine = -1;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));
         }
         break;

      case  MI_NAMEFLOWED :
         {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_NAME|CV_FLOW;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));

            WinSendMsg(pcdData->hwndCnr,
                       CM_ARRANGE,
                       NULL,
                       NULL);
         }
         break;

      case  MI_TEXTFLOWED :
         {
            CNRINFO          ciInfo;

            ciInfo.cb = sizeof(CNRINFO);
            ciInfo.flWindowAttr = CV_TEXT|CV_FLOW;

            WinSendMsg(pcdData->hwndCnr,
                       CM_SETCNRINFO,
                       MPFROMP(&ciInfo),
                       MPFROMLONG(CMA_FLWINDOWATTR));

            WinSendMsg(pcdData->hwndCnr,
                       CM_ARRANGE,
                       NULL,
                       NULL);
         }
         break;

      case  MI_EXIT :
```

```
            WinPostMsg(hwndClient,
                       WM_CLOSE,
                       0,
                       0);
            break;

         case  MI_RESUME :
            break;

         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

      }                                 /* endswitch                */
      break;

   case  WM_PAINT :
      {
         HPS              hpsPaint;
         RECTL            rclPaint;

         hpsPaint = WinBeginPaint(hwndClient,
                                  NULLHANDLE,
                                  &rclPaint);

         WinFillRect(hpsPaint,
                     &rclPaint,
                     SYSCLR_WINDOW);
         WinEndPaint(hpsPaint);
      }
      break;

   default  :
      return  WinDefWindowProc(hwndClient,
                               ulMsg,
                               mpParm1,
                               mpParm2);

}                                       /* endswitch                */
return  MRFROMSHORT(FALSE);
}
```

## CONTAIN3.RC

```
#include <os2.h>
#include "contain3.h"

ICON ICO_ITEM CONTAIN3.ICO

MENU RES_CLIENT
{
   SUBMENU "~Views   ", M_VIEWS
   {
      MENUITEM "~Icon", MI_ICON
      MENUITEM "~Detail", MI_DETAIL
      MENUITEM "~Tree", MI_TREE
      MENUITEM "~Name/flowed", MI_NAMEFLOWED
      MENUITEM "Te~xt/flowed", MI_TEXTFLOWED
   }
   SUBMENU "E~xit", M_EXIT
   {
      MENUITEM "E~xit", MI_EXIT
      MENUITEM "~Resume", MI_RESUME
   }
}
```

## CONTAIN3.H

```
#define RES_CLIENT              256
#define WND_CONTAINER           257
#define ICO_ITEM                258
#define M_VIEWS                 320
#define MI_ICON                 321
#define MI_DETAIL               322
#define MI_TREE                 323
#define MI_NAMEFLOWED           324
#define MI_TEXTFLOWED           325
#define M_EXIT                  336
#define MI_EXIT                 337
#define MI_RESUME               338
```

## CONTAIN3.MAK

```
CONTAIN3.EXE:                    CONTAIN3.OBJ \
                                 CONTAIN3.RES
        LINK386 $(LINKOPTS) @<<
CONTAIN3
CONTAIN3
CONTAIN3
OS2386
CONTAIN3
<<
        RC CONTAIN3.RES CONTAIN3.EXE

CONTAIN3.RES:                    CONTAIN3.RC \
                                 CONTAIN3.H
        RC -r CONTAIN3.RC CONTAIN3.RES

CONTAIN3.OBJ:                    CONTAIN3.C \
                                 CONTAIN3.H
        ICC -C+ -Kb+ -Ss+ CONTAIN3.C
```

**CONTAIN3.DEF**

```
NAME  CONTAIN3  WINDOWAPI

DESCRIPTION 'Third container example
            Copyright 1992 by Larry Salomon
            All rights reserved.'
STACKSIZE 32768
```

**Gotcha!**
The container has a bug in that it will send the WM_CONTROL / CN_CONTEXTMENU notification if the mouse was used to request the pop-up, but it will not send this message if the keyboard is used.

The WM_CONTROL notification specifies the record under the mouse when the pop-up menu was request. If there was no record, NULL is specified instead.

```
psiSales = ( PSALESINFO ) PVOIDFROMMP ( mpParm2 ) ;
if ( psiSales != NULL ) {
   if (( psiSales -> mrcStd.flRecordAttr &
     CRA_SELECTED ) == 0 )
   {
      WinSendMsg ( pcdData -> hwndCnr,
                                 CM_SETRECORDEMPHASIS,
                                 MPFROMP ( psiSales ) ,
                  MPFROM2SHORT ( TRUE, CRA_SOURCE )) ;
      psiSales -> bEmphasized = TRUE ;
   } else {
      emphasizeRecs ( pcdData -> hwndCnr, TRUE ) ;
   } /* endif */
} else {
   WinSendMsg ( pcdData -> hwndCnr,
                CM_SETRECORDEMPHASIS,
                0,
                MPFROM2SHORT ( TRUE, CRA_SOURCE )) ;
   pcdData -> bCnrSelected = TRUE ;
} /* endif */
```

The records are selected using the CM_SETRECORDEMPHASIS message; this message sets the appropriate bit in the *flRecordAttr* field and redraws the record. Conceivably this could be done explicitly, but why go through the extra work? The method of determining which records are given source emphasis follows that of the Workplace Shell, and can be summarized in the following manner:

- If there is a record under the mouse and it is selected, give all selected records source emphasis.
- If there is a record under the mouse and it is not selected, give it source emphasis only.
- If there are no records under the mouse, give the entire container source emphasis.

**Gotcha!**

The documentation does not state how the container is given source emphasis. This is done by specifying NULL for the record pointer in *mpParm1*. The container does not keep track of whether it has source emphasis or not and blindly draws this emphasis using the XOR method. Thus, if two CM_SETRECORDEMPHASIS messages are sent, both specifying that source emphasis is to be removed from the container, no visible difference will be seen.

After the records have been given source emphasis in the appropriate manner, the pointer position is determined and the menu is popped up via the *WinPopupMenu* message.

```
WinQueryPointerPos ( HWND_DESKTOP, &ptlMouse ) ;
WinMapWindowPoints ( HWND_DESKTOP,
                     hwndClient,
                     &ptlMouse,
                     1 ) ;

WinPopupMenu ( hwndClient,
               hwndClient,
               pcdData -> hwndMenu,
               ptlMouse.x,
               ptlMouse.y,
               M_VIEWS,
               PU_HCONSTRAIN |
               PU_VCONSTRAIN |
               PU_KEYBOARD |
               PU_MOUSEBUTTON1 |
               PU_MOUSEBUTTON2 |
               PU_NONE ) ;
```

# Direct Editing

As stated earlier, the user can edit directly with a mouse click. The application must be aware of this possibility and be able to process this event properly. When the user selects the proper combination of mouse clicks or keystrokes, the container sends the application a WM_CONTROL message with a CN_BEGINEDIT notification code. The data in the second parameter is a pointer to the CNREDITDATA structure.

```
typedef struct _CNREDITDATA {
   ULONG cb;
   HWND hwndCnr;
   PRECORDCORE pRecord;
   PFIELDINFO pFieldInfo;
   PSZ *ppszText;
   ULONG cbText;
   ULONG id;
} CNREDITDATA;
```

*cb* is the size of the structure in bytes. *hwndCnr* is the handle of the container window. *pRecord* is a pointer to the RECORDCORE structure of the object being edited. If the container titles are being edited, this field is NULL. *pFieldInfo* is a pointer to the FIELDINFO structure if the current view is detail view and the column titles are not being edited. Otherwise, this field is NULL. *ppszText* points to the pointer to the current text if the notification code is CN_BEGINEDIT or CN_REALLOCPSZ. For CN_ENDEDIT notification, this points to the pointer to the new text. *cbText* specifies the number of bytes in the text. *id* is the identifier of the window being edited and is a CID_ constant.

The CN_BEGINEDIT notification allows the application to perform any preedit processing, such as setting a limit on the text length. After the user direct editing, the container sends a CN_REALLOCPSZ notification to the container's owner before copying the new text into the application's text string to allow adjust the buffer size if needed. Finally, a CN_ENDEDIT notification is sent to allow any postedit processing to be done.

**Gotcha!**
The application must return TRUE from the CN_REALLOCPSZ notification, or else the container will discard the editing changes.

## Of Sorting and Filtering

The final, great abilities we will look at are *sorting* and *filtering* records, which are done with a little assistance from the application. *Sorting* is a concept that programmers should be familiar with; *filtering*, however, might not be so familiar. Its idea is analogous to a strainer that would be used when cooking. Items that meet the criteria demanded by the strainer (that they are smaller than a defined threshold) can continue on their merry way. Items that do not, may not. "Continuing" in the sense of the container is the visibility state of the record. If the record meets the threshold, it remains visible; if it doesn't, it is hidden. It should be noted that filtered records are not deleted—they simply aren't shown. Defining the threshold such that all records will meet it will reshow all of the records.

The sorting and filtering callback functions are defined in the following manner. For sorting, we have:

```
SHORT EXPENTRY pfnSort(PRECORDCORE prcFirst,
                       PRECORDCORE prcSecond,
                       PVOID pvData)
```

For filtering, we have:

```
BOOL EXPENTRY pfnSort(PRECORDCORE prcRecord,
                      PVOID pvData)
```

Of course, if the container was created with the CCS_MINIRECORDCORE style, the RECORDCORE pointers are instead MINIRECORDCORE pointers.

The sorting function behaves just like *strcmp*—if the first record is "less than" the second, a negative number should be returned; if the first record is "equal to" the second, 0 should be returned; if the first record is "greater than" the second, a positive number should be returned. The container takes care of the rest.

Filtering is just as easy—if the record meets the criteria and should remain visible, TRUE should be returned. Otherwise, return FALSE.

The following sample illustrates both sorting and filtering.

**CONTAIN4.C**

```
#define  INCL_WINFRAMEMGR
#define  INCL_WINMENUS
#define  INCL_WINPOINTERS
#define  INCL_WINSTDCNR
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "CONTAIN4.H"
#define  CLS_CLIENT     "SampleClass"
#define  MAX_YEARS      10
#define  MAX_MONTHS     12
#define  MAX_COLUMNS    4
#define  CX_SPLITBAR    120
//------------------------------------------------
// For the GA 2.0 toolkit, CRA_SOURCE is not defined,
// but it should be.
//------------------------------------------------

#ifndef  CRA_SOURCE
#define  CRA_SOURCE     0x00004000L
#endif
typedef struct _CLIENTDATA
{
   HWND              hwndCnr;

   HPOINTER          hptrIcon;
   HWND              hwndMenu;
   BOOL              bCnrSelected;
} CLIENTDATA,*PCLIENTDATA;

typedef struct   _SALESINFO
{
   MINIRECORDCORE    mrcStd;
   BOOL              bEmphasized;
   ULONG             ulNumUnits;
   float             fSales;
   PCHAR             pchSales;
} SALESINFO,*PSALESINFO;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   BOOL              bLoop;
   QMSG              qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    sizeof(PVOID));
```

```
   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Container Sample",
                                  0L,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                 /* endwhile                  */
      WinDestroyWindow(hwndFrame);
   }                                    /* endif                     */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

VOID initSalesInfo(PCLIENTDATA pcdData,PSALESINFO psiParent,
                   PSALESINFO psiSales,USHORT usIndex)
{
   PCHAR            pchPos;

   psiSales->mrcStd.cb = sizeof(MINIRECORDCORE);

   psiSales->mrcStd.pszIcon = malloc(256);
   if (psiSales->mrcStd.pszIcon != NULL)
   {
      if (psiParent != NULL)
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Month %d",
                 usIndex+1);
      }
      else
      {
         sprintf(psiSales->mrcStd.pszIcon,
                 "Year 19%02d",
                 usIndex+84);
      }                                 /* endif                     */
   }                                    /* endif                     */
   psiSales->mrcStd.hptrIcon = pcdData->hptrIcon;
   psiSales->bEmphasized = FALSE;

   if (psiParent != NULL)
   {
      psiSales->ulNumUnits = psiParent->ulNumUnits/12;
   }
```

```
   else
   {
      psiSales->ulNumUnits = rand()%100;
   }                                    /* endif                        */
   psiSales->fSales = (float)psiSales->ulNumUnits *9.95;

   psiSales->pchSales = malloc(16);
   if (psiSales->pchSales != NULL)
   {
      sprintf(psiSales->pchSales,
              "$%-10.2f",
              psiSales->fSales);

      pchPos = psiSales->pchSales;
      while (!isspace(*pchPos) && (*pchPos != 0))
      {
         pchPos++;
      }                                 /* endwhile                     */
      *pchPos = 0;
   }                                    /* endif                        */
   return ;
}

VOID emphasizeRecs(HWND hwndCnr,BOOL bEmphasize)
{
   SHORT            sFlag;
   PSALESINFO       psiYear;

   sFlag = ((bEmphasize)?CRA_SELECTED:CRA_SOURCE);

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                CM_QUERYRECORDEMPHASIS
                                                   ,
                                                MPFROMP(CMA_FIRST),
                                                MPFROMSHORT(sFlag)));

   while (psiYear != NULL)
   {
      if (bEmphasize)
      {
         WinSendMsg(hwndCnr,
                    CM_SETRECORDEMPHASIS,
                    MPFROMP(psiYear),
                    MPFROM2SHORT(TRUE,
                                 CRA_SOURCE));

         psiYear->bEmphasized = TRUE;
      }
      else
      {
         WinSendMsg(hwndCnr,
                    CM_SETRECORDEMPHASIS,
                    MPFROMP(psiYear),
                    MPFROM2SHORT(FALSE,
                                 CRA_SOURCE));

         psiYear->bEmphasized = FALSE;
      }                                 /* endif                        */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                   CM_QUERYRECORDEMPHASIS
                                                      ,
                                                   MPFROMP(psiYear),
                                                   MPFROMSHORT(sFlag)
                                                   ));
   }                                    /* endwhile                     */
   return ;
}
```

```
VOID freeCnrInfo(HWND hwndCnr)
{
   PSALESINFO       psiYear;
   PSALESINFO       psiMonth;

   psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                CM_QUERYRECORD,
                                                0L,
                                                MPFROM2SHORT
                                                   (CMA_FIRST,
                                                    CMA_ITEMORDER)));
   while (psiYear != NULL)
   {
      psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                    CM_QUERYRECORD,
                                                    MPFROMP(psiYear),
                                                    MPFROM2SHORT
                                                    (CMA_FIRSTCHILD,
                                                     CMA_ITEMORDER)))
         ;
      while (psiMonth != NULL)
      {
         if (psiMonth->mrcStd.pszIcon != NULL)
         {
            free(psiMonth->mrcStd.pszIcon);
         }                                  /* endif                    */
         if (psiMonth->pchSales != NULL)
         {
            free(psiMonth->pchSales);
         }                                  /* endif                    */
         psiMonth = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                       CM_QUERYRECORD
                                                          ,
                                                       MPFROMP
                                                       (psiMonth),
                                                       MPFROM2SHORT
                                                       (CMA_NEXT,
                                                        CMA_ITEMORDER
                                                        )));
      }                                     /* endwhile                 */
      if (psiYear->mrcStd.pszIcon != NULL)
      {
         free(psiYear->mrcStd.pszIcon);
      }                                     /* endif                    */
      if (psiYear->pchSales != NULL)
      {
         free(psiYear->pchSales);
      }                                     /* endif                    */
      psiYear = (PSALESINFO)PVOIDFROMMR(WinSendMsg(hwndCnr,
                                                   CM_QUERYRECORD,
                                                   MPFROMP(psiYear),
                                                   MPFROM2SHORT
                                                      (CMA_NEXT,
                                                       CMA_ITEMORDER)
                                                       ));
   }                                        /* endwhile                 */
   return ;
}

VOID initColumns(PCLIENTDATA pcdData)
{
   CNRINFO          ciInfo;
   PFIELDINFO       pfiCurrent;
   PFIELDINFO       pfiInfo;
   PFIELDINFO       pfiLefty;
   FIELDINFOINSERT  fiiInfo;
```

```
   pfiInfo = (PFIELDINFO)PVOIDFROMMR(WinSendMsg(pcdData->hwndCnr,
                                                CM_ALLOCDETAILFIELDINFO
                                                      ,
                                                MPFROMLONG
                                                (MAX_COLUMNS),
                                                0L));

   pfiCurrent = pfiInfo;

   pfiCurrent->flData = CFA_BITMAPORICON|CFA_HORZSEPARATOR|CFA_CENTER
      |CFA_SEPARATOR;
   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Icon";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       mrcStd.hptrIcon);

   pfiCurrent = pfiCurrent->pNextFieldInfo;
   pfiCurrent->flData = CFA_STRING|CFA_CENTER|CFA_HORZSEPARATOR;
   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Year";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       mrcStd.pszIcon);

   pfiLefty = pfiCurrent;

   pfiCurrent = pfiCurrent->pNextFieldInfo;
   pfiCurrent->flData = CFA_ULONG|CFA_CENTER|CFA_HORZSEPARATOR|
      CFA_SEPARATOR;

   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Units Sold";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       ulNumUnits);

   pfiCurrent = pfiCurrent->pNextFieldInfo;
   pfiCurrent->flData = CFA_STRING|CFA_RIGHT|CFA_HORZSEPARATOR;
   pfiCurrent->flTitle = CFA_STRING|CFA_CENTER;
   pfiCurrent->pTitleData = "Sales";
   pfiCurrent->offStruct = FIELDOFFSET(SALESINFO,
                                       pchSales);

   fiiInfo.cb = sizeof(fiiInfo);
   fiiInfo.pFieldInfoOrder = (PFIELDINFO)CMA_FIRST;
   fiiInfo.cFieldInfoInsert = MAX_COLUMNS;
   fiiInfo.fInvalidateFieldInfo = TRUE;

   WinSendMsg(pcdData->hwndCnr,
              CM_INSERTDETAILFIELDINFO,
              MPFROMP(pfiInfo),
              MPFROMP(&fiiInfo));

   memset(&ciInfo,
          0,
          sizeof(ciInfo));
   ciInfo.cb = sizeof(CNRINFO);
   ciInfo.pFieldInfoLast = pfiLefty;
   ciInfo.xVertSplitbar = CX_SPLITBAR;

   WinSendMsg(pcdData->hwndCnr,
              CM_SETCNRINFO,
              MPFROMP(&ciInfo),
              MPFROMLONG(CMA_PFIELDINFOLAST|CMA_XVERTSPLITBAR));
   return ;
}

SHORT EXPENTRY sortRecords(PSALESINFO psiFirst,PSALESINFO psiSecond,
                           PUSHORT pusSortBy)
{
   switch (*pusSortBy)
```

```
   {
      case  MI_SORTBYUNITS :
         if (psiFirst->ulNumUnits < psiSecond->ulNumUnits)
         {
            return -1;
         }
         else
            if (psiFirst->ulNumUnits == psiSecond->ulNumUnits)
            {
               return 0;
            }
            else
            {
               return 1;
            }                                  /* endif                    */
      case  MI_SORTBYYEAR :
         return  strcmp(psiFirst->mrcStd.pszIcon,
                        psiSecond->mrcStd.pszIcon);
      default  :
         return 0;
   }                                           /* endswitch                */
}

BOOL EXPENTRY filterRecords(PSALESINFO psiInfo,PUSHORT pusFilterBy)
{
   switch (*pusFilterBy)
   {
      case  MI_FILTER300DOLLARS :
         return (psiInfo->fSales > 300.0);
      case  MI_FILTER400DOLLARS :
         return (psiInfo->fSales > 400.0);
      case  MI_FILTER500DOLLARS :
         return (psiInfo->fSales > 500.0);
      case  MI_FILTERNONE :
         return  TRUE;
      default  :
         return  TRUE;
   }                                           /* endswitch                */
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCLIENTDATA      pcdData;

   pcdData = (PCLIENTDATA)WinQueryWindowPtr(hwndClient,
                                            0);

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
            MENUITEM         miItem;
            ULONG            ulStyle;
            ULONG            ulExtra;
            RECORDINSERT     riRecord;
            PSALESINFO       psiYears;
            PSALESINFO       psiCYear;
            USHORT           usIndex1;
            PSALESINFO       psiMonths;
            PSALESINFO       psiCMonth;
            USHORT           usIndex2;

            pcdData = (PCLIENTDATA)malloc(sizeof(CLIENTDATA));
            if (pcdData == NULL)
            {
```

```
            WinAlarm(HWND_DESKTOP,
                     WA_ERROR);
            WinMessageBox(HWND_DESKTOP,
                          HWND_DESKTOP,
                          "No memory is available",
                          "Error",
                          0,
                          MB_ICONEXCLAMATION|MB_OK);
            return  MRFROMSHORT(TRUE);
         }                              /* endif                    */

         WinSetWindowPtr(hwndClient,
                         0,
                         pcdData);

         pcdData->hwndCnr = NULLHANDLE;
         pcdData->hptrIcon = NULLHANDLE;
         pcdData->hwndMenu = NULLHANDLE;
         pcdData->bCnrSelected = FALSE;

         pcdData->hwndCnr = WinCreateWindow(hwndClient,
                                            WC_CONTAINER,
                                            "",
                                            CCS_MINIRECORDCORE|
                                               CCS_EXTENDSEL|
                                               WS_VISIBLE,
                                            0,
                                            0,
                                            0,
                                            0,
                                            hwndClient,
                                            HWND_TOP,
                                            WND_CONTAINER,
                                            NULL,
                                            NULL);
         if (pcdData->hwndCnr == NULLHANDLE)
         {
            free(pcdData);
            WinAlarm(HWND_DESKTOP,
                     WA_ERROR);
            WinMessageBox(HWND_DESKTOP,
                          HWND_DESKTOP,
                          "Cannot create container",
                          "Error",
                          0,
                          MB_ICONEXCLAMATION|MB_OK);
            return  MRFROMSHORT(TRUE);
         }                              /* endif                    */

         pcdData->hptrIcon = WinLoadPointer(HWND_DESKTOP,
                                            NULLHANDLE,
                                            ICO_ITEM);

         pcdData->hwndMenu = WinLoadMenu(hwndClient,
                                         NULLHANDLE,
                                         RES_CLIENT);
//-------------------------------------------------------------
// "Gotcha" note:  you cannot specify the MS_CONDITIONALCASCADE
// style in the .RC file.  You *must* do it this way.  Also,
// you must pad the pull-right text with spaces to account for
// the button that will be placed there.
//-------------------------------------------------------------

         WinSendMsg(pcdData->hwndMenu,
                    MM_QUERYITEM,
                    MPFROM2SHORT(M_VIEWS,
                                 TRUE),
                    MPFROMP(&miItem));
```

```
ulStyle = WinQueryWindowULong(miItem.hwndSubMenu,
                              QWL_STYLE);
ulStyle |= MS_CONDITIONALCASCADE;
WinSetWindowULong(miItem.hwndSubMenu,
                  QWL_STYLE,
                  ulStyle);

WinSendMsg(miItem.hwndSubMenu,
           MM_SETDEFAULTITEMID,
           MPFROMSHORT(MI_ICON),
           0L);

ulExtra = sizeof(SALESINFO)-sizeof(MINIRECORDCORE);

riRecord.cb = sizeof(RECORDINSERT);
riRecord.pRecordOrder = (PRECORDCORE)CMA_END;
riRecord.fInvalidateRecord = FALSE;
riRecord.zOrder = CMA_TOP;

psiYears = (PSALESINFO)PVOIDFROMMR(WinSendMsg
   (pcdData->hwndCnr,
    CM_ALLOCRECORD,
    MPFROMLONG(ulExtra),
    MPFROMSHORT(MAX_YEARS)));

psiCYear = psiYears;

for (usIndex1 = 0; usIndex1 < MAX_YEARS; usIndex1++)
{
   initSalesInfo(pcdData,
                 NULL,
                 psiCYear,
                 usIndex1);

   riRecord.pRecordParent = NULL;
   riRecord.cRecordsInsert = 1;

   WinSendMsg(pcdData->hwndCnr,
              CM_INSERTRECORD,
              MPFROMP(psiCYear),
              MPFROMP(&riRecord));

   psiMonths = (PSALESINFO)PVOIDFROMMR(WinSendMsg
      (pcdData->hwndCnr,
       CM_ALLOCRECORD,
       MPFROMLONG(ulExtra),
       MPFROMSHORT(MAX_MONTHS)));

   psiCMonth = psiMonths;

   for (usIndex2 = 0; usIndex2 < MAX_MONTHS; usIndex2++)
   {
      initSalesInfo(pcdData,
                    psiCYear,
                    psiCMonth,
                    usIndex2);

      psiCMonth = (PSALESINFO)
         psiCMonth->mrcStd.preccNextRecord;
   }                          /* endfor                    */
   riRecord.pRecordParent = (PRECORDCORE)psiCYear;
   riRecord.cRecordsInsert = MAX_MONTHS;

   WinSendMsg(pcdData->hwndCnr,
              CM_INSERTRECORD,
              MPFROMP(psiMonths),
              MPFROMP(&riRecord));
```

```
            psiCYear = (PSALESINFO)
               psiCYear->mrcStd.preccNextRecord;
         }                              /* endfor                    */
         initColumns(pcdData);

         WinSendMsg(hwndClient,
                    WM_COMMAND,
                    MPFROMSHORT(MI_ICON),
                    0);
      }
      break;
   case  WM_DESTROY :
      freeCnrInfo(pcdData->hwndCnr);

   if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinDestroyWindow(pcdData->hwndCnr);
      }                              /* endif                     */
      if (pcdData->hptrIcon != NULLHANDLE)
      {
         WinDestroyPointer(pcdData->hptrIcon);
      }                              /* endif                     */
      free(pcdData);
      break;
   case  WM_SIZE :
      if (pcdData->hwndCnr != NULLHANDLE)
      {
         WinSetWindowPos(pcdData->hwndCnr,
                         NULLHANDLE,
                         0,
                         0,
                         SHORT1FROMMP(mpParm2),
                         SHORT2FROMMP(mpParm2),
                         SWP_MOVE|SWP_SIZE);
      }                              /* endif                     */
      break;
   case  WM_MENUEND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  FID_MENU :
            if (pcdData->bCnrSelected)
            {
               WinSendMsg(pcdData->hwndCnr,
                          CM_SETRECORDEMPHASIS,
                          0L,
                          MPFROM2SHORT(FALSE,
                                       CRA_SOURCE));
               pcdData->bCnrSelected = FALSE;
            }
            else
            {
               emphasizeRecs(pcdData->hwndCnr,
                             FALSE);
            }                           /* endif                     */
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }                              /* endswitch                 */
      break;

   case  WM_CONTROL :

      switch (SHORT1FROMMP(mpParm1))
      {
         case  WND_CONTAINER :
```

```
            switch (SHORT2FROMMP(mpParm1))
            {
               case  CN_CONTEXTMENU :
                  {
                     PSALESINFO       psiSales;
                     POINTL           ptlMouse;

                     psiSales = (PSALESINFO)PVOIDFROMMP(mpParm2);
                     if (psiSales != NULL)
                     {
                        if ((psiSales->mrcStd.flRecordAttr
                           &CRA_SELECTED) == 0)
                        {
                           WinSendMsg(pcdData->hwndCnr,
                                      CM_SETRECORDEMPHASIS,
                                      MPFROMP(psiSales),
                                      MPFROM2SHORT(TRUE,
                                                   CRA_SOURCE));
                           psiSales->bEmphasized = TRUE;
                        }
                        else
                        {
                           emphasizeRecs(pcdData->hwndCnr,
                                         TRUE);
                        }           /* endif                    */
                     }
                     else
                     {
                        WinSendMsg(pcdData->hwndCnr,
                                   CM_SETRECORDEMPHASIS,
                                   0L,
                                   MPFROM2SHORT(TRUE,
                                                CRA_SOURCE));
                        pcdData->bCnrSelected = TRUE;
                     }              /* endif                    */
                     WinQueryPointerPos(HWND_DESKTOP,
                                        &ptlMouse);
                     WinMapWindowPoints(HWND_DESKTOP,
                                        hwndClient,
                                        &ptlMouse,
                                        1);
                     WinPopupMenu(hwndClient,
                                  hwndClient,
                                  pcdData->hwndMenu,
                                  ptlMouse.x,
                                  ptlMouse.y,
                                  M_VIEWS,
                                  PU_HCONSTRAIN|PU_VCONSTRAIN|
                                     PU_KEYBOARD|PU_MOUSEBUTTON1|
                                     PU_MOUSEBUTTON2|PU_NONE);
                  }
                  break;
               default  :
                  return  WinDefWindowProc(hwndClient,
                                           ulMsg,
                                           mpParm1,
                                           mpParm2);
            }                       /* endswitch                */
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }                             /* endswitch                */
      break;
```

```
   case  WM_COMMAND :

      switch (SHORT1FROMMP(mpParm1))
      {
         case  MI_ICON :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_ICON;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));

               WinSendMsg(pcdData->hwndCnr,
                          CM_ARRANGE,
                          NULL,
                          NULL);
            }
            break;
         case  MI_DETAIL :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_DETAIL|
                  CA_DETAILSVIEWTITLES;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));
            }
            break;
         case  MI_TREE :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_TREE|CV_ICON|CA_TREELINE;
               ciInfo.cxTreeIndent = -1;
               ciInfo.cxTreeLine = -1;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));
            }
            break;
         case  MI_NAMEFLOWED :
            {
               CNRINFO          ciInfo;

               ciInfo.cb = sizeof(CNRINFO);
               ciInfo.flWindowAttr = CV_NAME|CV_FLOW;

               WinSendMsg(pcdData->hwndCnr,
                          CM_SETCNRINFO,
                          MPFROMP(&ciInfo),
                          MPFROMLONG(CMA_FLWINDOWATTR));

               WinSendMsg(pcdData->hwndCnr,
                          CM_ARRANGE,
                          NULL,
                          NULL);
            }
            break;
```

```
        case  MI_TEXTFLOWED :
           {
              CNRINFO          ciInfo;

              ciInfo.cb = sizeof(CNRINFO);
              ciInfo.flWindowAttr = CV_TEXT|CV_FLOW;

              WinSendMsg(pcdData->hwndCnr,
                         CM_SETCNRINFO,
                         MPFROMP(&ciInfo),
                         MPFROMLONG(CMA_FLWINDOWATTR));

              WinSendMsg(pcdData->hwndCnr,
                         CM_ARRANGE,
                         NULL,
                         NULL);
           }
           break;
        case  MI_SORTBYUNITS :
           {
              USHORT           usId;

              usId = MI_SORTBYUNITS;

              WinSendMsg(pcdData->hwndCnr,
                         CM_SORTRECORD,
                         MPFROMP(sortRecords),
                         MPFROMP(&usId));
           }
           break;
        case  MI_SORTBYYEAR :
           {
              USHORT           usId;

              usId = MI_SORTBYYEAR;

              WinSendMsg(pcdData->hwndCnr,
                         CM_SORTRECORD,
                         MPFROMP(sortRecords),
                         MPFROMP(&usId));
           }
           break;
        case  MI_FILTER300DOLLARS :
           {
              USHORT           usId;

              usId = MI_FILTER300DOLLARS;

              WinSendMsg(pcdData->hwndCnr,
                         CM_FILTER,
                         MPFROMP(filterRecords),
                         MPFROMP(&usId));
           }
           break;
        case  MI_FILTER400DOLLARS :
           {
              USHORT           usId;

              usId = MI_FILTER400DOLLARS;

              WinSendMsg(pcdData->hwndCnr,
                         CM_FILTER,
                         MPFROMP(filterRecords),
                         MPFROMP(&usId));
           }
           break;
        case  MI_FILTER500DOLLARS :
```

```
            {
               USHORT           usId;

               usId = MI_FILTER500DOLLARS;

               WinSendMsg(pcdData->hwndCnr,
                          CM_FILTER,
                          MPFROMP(filterRecords),
                          MPFROMP(&usId));
            }
            break;
         case  MI_FILTERNONE :
            {
               USHORT           usId;

               usId = MI_FILTERNONE;

               WinSendMsg(pcdData->hwndCnr,
                          CM_FILTER,
                          MPFROMP(filterRecords),
                          MPFROMP(&usId));
            }
            break;
         case  MI_EXIT :
            WinPostMsg(hwndClient,
                       WM_CLOSE,
                       0L,
                       0L);
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }                              /* endswitch                    */
      break;

   case  WM_PAINT :
      {
         HPS              hpsPaint;
         RECTL            rclPaint;

         hpsPaint = WinBeginPaint(hwndClient,
                                  NULLHANDLE,
                                  &rclPaint);
         WinFillRect(hpsPaint,
                     &rclPaint,
                     SYSCLR_WINDOW);
         WinEndPaint(hpsPaint);
      }
      break;
   default  :
      return  WinDefWindowProc(hwndClient,
                               ulMsg,
                               mpParm1,
                               mpParm2);
}                                    /* endswitch                    */
return  MRFROMSHORT(FALSE);
}
```

## CONTAIN4.RC

```
#include <os2.h>
#include "contain4.h"

ICON ICO_ITEM CONTAIN4.ICO

MENU RES_CLIENT
{
   SUBMENU "~Views    ", M_VIEWS
   {
      MENUITEM "~Icon", MI_ICON
      MENUITEM "~Detail", MI_DETAIL
      MENUITEM "~Tree", MI_TREE
      MENUITEM "~Name/flowed", MI_NAMEFLOWED
      MENUITEM "Te~xt/flowed", MI_TEXTFLOWED
   }
   MENUITEM SEPARATOR
   SUBMENU "~Sort    ", M_SORT
   {
      MENUITEM "By ~units sold", MI_SORTBYUNITS
      MENUITEM "By ~year", MI_SORTBYYEAR
   }
   MENUITEM SEPARATOR
   SUBMENU "~Filter    ", M_FILTER
   {
      MENUITEM "Show $~300.00 or greater", MI_FILTER300DOLLARS
      MENUITEM "Show $~400.00 or greater", MI_FILTER400DOLLARS
      MENUITEM "Show $~500.00 or greater", MI_FILTER500DOLLARS
      MENUITEM "Show all records", MI_FILTERNONE
   }
   MENUITEM SEPARATOR
   MENUITEM "E~xit", MI_EXIT
}
```

## CONTAIN4.H

```
#define RES_CLIENT              256
#define WND_CONTAINER           257
#define ICO_ITEM                258
#define M_VIEWS                 320
#define MI_ICON                 321
#define MI_DETAIL               322
#define MI_TREE                 323
#define MI_NAMEFLOWED           324
#define MI_TEXTFLOWED           325
#define M_SORT                  336
#define MI_SORTBYUNITS          337
#define MI_SORTBYYEAR           338
#define M_FILTER                352
#define MI_FILTER300DOLLARS     353
#define MI_FILTER400DOLLARS     354
#define MI_FILTER500DOLLARS     355
#define MI_FILTERNONE           356
#define MI_EXIT                 368
```

**CONTAIN4.MAK**

```
ICCOPTS=-C+ -Gm- -Kb+ -Ss+
LINKOPTS=/MAP /A:16

CONTAIN4.EXE:                   CONTAIN4.OBJ \
                                CONTAIN4.RES
        LINK386 $(LINKOPTS) @<<
CONTAIN4
CONTAIN4
CONTAIN4
OS2386
CONTAIN4
<<
        RC CONTAIN4.RES CONTAIN4.EXE

CONTAIN4.RES:                   CONTAIN4.RC \
                                CONTAIN4.H
        RC -r CONTAIN4.RC CONTAIN4.RES

CONTAIN4.OBJ:                   CONTAIN4.C \
                                CONTAIN4.H
        ICC $(ICCOPTS) CONTAIN4.C
```

**CONTAIN4.DEF**

```
;-------------------------------------------
; CONTAIN4.DEF
;-------------------------------------------

NAME  CONTAIN4  WINDOWAPI

DESCRIPTION 'Fourth container example
Copyright 1992 by Larry Salomon
All rights reserved.'

CODE    MOVEABLE
DATA    MOVEABLE MULTIPLE

HEAPSIZE  10240
STACKSIZE 32768
```

By running this sample, it will be seen that two sort menu items are provided, sort by units solds and sort by revenue. The code actually to sort the records is quite simple.

```
case MI_SORTBYUNITS:
   {
      USHORT usId;

      usId=MI_SORTBYUNITS;

      WinSendMsg(pcdData->hwndCnr,
                 CM_SORTRECORD,
                 MPFROMP(sortRecords),
                 MPFROMP(&usId));
   }
   break;
```

As was said, this really is simple. The callback function is just as easy to understand.

```
SHORT EXPENTRY sortRecords(PSALESINFO psiFirst,
                           PSALESINFO psiSecond,
                           PUSHORT pusSortBy)
{
   switch (*pusSortBy) {
   case MI_SORTBYUNITS:
      if (psiFirst->ulNumUnits<psiSecond->ulNumUnits) {
         return -1;
      } else
      if (psiFirst->ulNumUnits==psiSecond->ulNumUnits) {
         return 0;
      } else {
         return 1;
      } /* endif */
   case MI_SORTBYYEAR:
      return strcmp(psiFirst->mrcStd.pszIcon,
                    psiSecond->mrcStd.pszIcon);
   default:
      return 0;
   } /* endswitch */
}
```

It checks to see by what the user requested the items to be sorted and then checks the appropriate field in the SALESINFO structure. That is all there is to sorting; there isn't anything difficult about it.

Filtering is even easier; the sample defines four filter choices: revenues greater than $300.00, greater than $400.00, greater than $500.00, and no filtering at all. Again, the code that actually filters the records is trivial.

```
case MI_FILTER300DOLLARS:
   {
      USHORT usId;

      usId=MI_FILTER300DOLLARS;

      WinSendMsg(pcdData->hwndCnr,
                 CM_FILTER,
                 MPFROMP(filterRecords),
                 MPFROMP(&usId));
   }
   break;
```

This is almost identical to the code that initiates the sorting. The callback is simpler than the sorting callback.

```
BOOL EXPENTRY filterRecords(PSALESINFO psiInfo,
                            PUSHORT pusFilterBy)
{
   switch (*pusFilterBy) {
   case MI_FILTER300DOLLARS:
      return (psiInfo->fSales>300.0);
   case MI_FILTER400DOLLARS:
      return (psiInfo->fSales>400.0);
   case MI_FILTER500DOLLARS:
      return (psiInfo->fSales>500.0);
   case MI_FILTERNONE:
      return TRUE;
   default:
      return TRUE;
   } /* endswitch */
}
```

It checks to see by what criteria the user wanted to filter the records and returns the appropriate value.

How much easier can it get?

## Where Does Direct Manipulation Fit In?

From what we can see of the container's capabilities, it was obviously designed to be an advanced control; thus, we would expect it to support direct manipulation (drag and drop). However, as Chapter 20 makes clear, direct manipulation is a very complex mechanism that could not possibly be supported entirely by the container. Instead, the container sends its owner a WM_CONTROL message with one of six notification codes specific to direct manipulation.

Table 23.6 Container Notification Codes

| Notification | Explanation |
|---|---|
| CN_DRAGAFTER | Sent after the container receives a DM_DRAGOVER message, the CA_ORDEREDTARGETEMPHASIS or CA_MIXEDTARGETEMPHASIS attribute is set in the CNRINFO structure, and the current view is name, text, or detail. |
| CN_DRAGLEAVE | Sent after the container receives a DM_DRAGLEAVE message. |
| CN_DRAGOVER | Sent after the container receives a DM_DRAGOVER message and the requirements for the CN_DRAGAFTER notification are not met. |
| CN_DROP | Sent after the container receives a DM_DROP message. |
| CN_DROPHELP | Sent after the container receives a DM_DROPHELP message. |
| CN_INITDRAG | Sent after the container receives a WM_BEGINDRAG message. |

Chapter 20 presents information about what is to be done when one of these notifications is received.

## Summary

The container control, while at times cumbersome to initialize and interact with, is a very useful addition to the library of standard controls provided with Presentation Manager. It is very flexible, providing many different viewing methods, and supports the CUA '91 user interface guidelines. With a little imagination and a great deal of programming, this control could greatly enhance the user interface of an application.

# Chapter 24

# Spin Buttons

A spin button is a button that will display a list of choices to the user. Up and down arrows are displayed to the right of the button; they are used to "spin" through the choices. Spin buttons should be used when the choices can be organized into some logical, consecutive order. For example, a list of days of the week would be a good use for a spin button. A spin button can be read-only, or it can be edited similar to an entry field.

## Spin Button Styles

Table 24.1 presents spin button styles.

Table 24.1 Spin Button Styles

| Style | Description |
|---|---|
| SPBS_ALLCHARACTERS | All characters are accepted into spin button. |
| SPBS_NUMERICONLY | Only the characters 0–9 are accepted into spin button. |
| SPBS_READONLY | No characters are allowed into spin button. |
| SPBS_MASTER | Spin button will have arrows displayed to the right. |
| SPBS_SERVANT | Spin button has no arrows but is attached to a set of spin buttons that share one set of arrows. |
| SPBS_JUSTLEFT | Left-justify the spin button text. |
| SPBS_JUSTRIGHT | Right-justify the spin button text. |
| SPBS_JUSTCENTER | Center the spin button text. |
| SPBS_NOBORDER | No border will be drawn around spin button. |
| SPBS_FASTSPIN | Spin button can skip over numbers, when arrows are held down. |
| SPBS_PADWITHZEROS | Pad the number with zeros. |

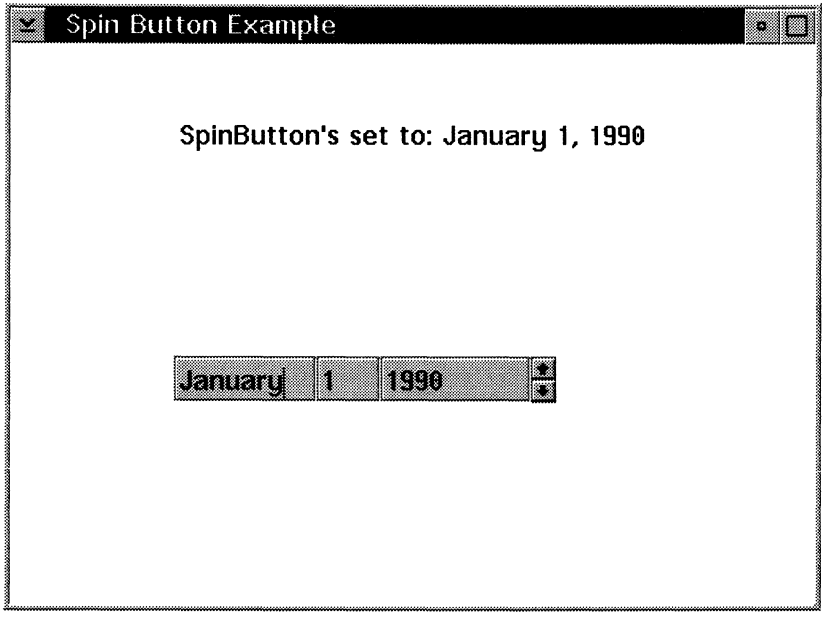



Figure 24.1 One master spin button with two slave spin buttons.

Figure 24.1 illustrates one master spin button with two slaves. A master spin button contains spin arrows, and the servant spin buttons do not. The master spin arrows control the spinning of the master button and the attached slaves. When the user spins the arrows, the button with the cursor is the button that will spin.

The following example program shows how to use a spin button in a program.

**SPIN.C**

```
#define  INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include "spin.h"
#define  CLS_CLIENT     "MyClass"
PCHAR achMonthArray[] =
{

   "January",
   "February",
   "March",
   "April",
   "May",
   "June",
   "July",
   "August",
   "September",
   "October",
   "November",
   "December"
} ;


PCHAR            achDayArray[] =
{
```

```
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "10",
    "11",
    "12",
    "13",
    "14",
    "15",
    "16",
    "17",
    "18",
    "19",
    "20",
    "21",
    "22",
    "23",
    "24",
    "25",
    "26",
    "27",
    "28",
    "29",
    "30",
    "31"
} ;

PCHAR            achYearArray[] =
{
    "1990",
    "1991",
    "1992",
    "1993",
    "1994",
    "1995",
    "1996",
    "1997",
    "1998",
    "1999",
    "2000"
} ;

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{
    HAB              habAnchor;
    HMQ              hmqQueue;
    ULONG            ulFlags;
    LONG             lHeight;
    LONG             lWidth;
    HWND             hwndFrame;
    HWND             hwndClient;
    BOOL             bLoop;
    QMSG             qmMsg;

    habAnchor = WinInitialize(0);
    hmqQueue = WinCreateMsgQueue(habAnchor,
                                 0);
```

```
   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW|CS_SYNCPAINT,
                    0);

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
      FCF_ACCELTABLE;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Spin Button Example",
                                  0,
                                  NULLHANDLE,
                                  ID_WINDOW,
                                  &hwndClient);

   if (hwndFrame != NULLHANDLE)
   {

      lHeight = WinQuerySysValue(HWND_DESKTOP,
                                 SV_CYSCREEN);
      lWidth = WinQuerySysValue(HWND_DESKTOP,
                                SV_CXSCREEN);

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      lWidth/4,
                      lHeight/4,
                      lWidth/2,
                      lHeight/2,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {

      case  WM_CREATE :
         {
            ULONG            ulMonthStyle;
            ULONG            ulDayStyle;
            ULONG            ulYearStyle;
```

```
LONG            lWidth;
LONG            lHeight;
LONG            xPosition;
LONG            yPosition;
LONG            yHeight;

ulMonthStyle = SPBS_SERVANT|SPBS_READONLY|
   SPBS_JUSTLEFT|SPBS_FASTSPIN|WS_VISIBLE;

ulDayStyle = SPBS_SERVANT|SPBS_READONLY|SPBS_JUSTLEFT
   |SPBS_FASTSPIN|WS_VISIBLE;

ulYearStyle = SPBS_MASTER|SPBS_READONLY|SPBS_JUSTLEFT
   |SPBS_FASTSPIN|WS_VISIBLE;

lHeight = WinQuerySysValue(HWND_DESKTOP,
                           SV_CYSCREEN)/2;
lWidth = WinQuerySysValue(HWND_DESKTOP,
                          SV_CXSCREEN)/2;

xPosition = lWidth/5;
yPosition = lHeight/3;
yHeight = 50;

WinCreateWindow(hwndWnd,
                WC_SPINBUTTON,
                NULL,
                ulMonthStyle,
                xPosition,
                yPosition,
                90,
                yHeight,
                hwndWnd,
                HWND_TOP,
                ID_SPINBUTTONMONTH,
                NULL,
                NULL);

WinCreateWindow(hwndWnd,
                WC_SPINBUTTON,
                NULL,
                ulDayStyle,
                xPosition+90,
                yPosition,
                40,
                yHeight,
                hwndWnd,
                HWND_TOP,
                ID_SPINBUTTONDAY,
                NULL,
                NULL);
```

```
        WinCreateWindow(hwndWnd,
                        WC_SPINBUTTON,
                        NULL,
                        ulYearStyle,
                        xPosition+90+40,
                        yPosition,
                        110,
                        yHeight,
                        hwndWnd,
                        HWND_TOP,
                        ID_SPINBUTTONYEAR,
                        NULL,
                        NULL);

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONMONTH,
                          SPBM_SETARRAY,
                          MPFROMP(achMonthArray),
                          MPFROMSHORT(12));

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONMONTH,
                          SPBM_SETMASTER,
                          MPFROMHWND(WinWindowFromID(hwndWnd,
                                                     ID_SPINBUTTONYEAR
                                                             )),
                          0);

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONDAY,
                          SPBM_SETARRAY,
                          MPFROMP(achDayArray),
                          MPFROMSHORT(31));

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONDAY,
                          SPBM_SETMASTER,
                          MPFROMHWND(WinWindowFromID(hwndWnd,
                                                     ID_SPINBUTTONYEAR
                                                             )),
                          0);

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONYEAR,
                          SPBM_SETARRAY,
                          MPFROMP(achYearArray),
                          MPFROMSHORT(11));

        WinSetFocus(HWND_DESKTOP,
                    WinWindowFromID(hwndWnd,
                                    ID_SPINBUTTONMONTH));
    }
    break;

case  WM_CONTROL :
    {
        USHORT            usID;
        USHORT            usNotifyCode;
        RECTL             rclWindow;

        usID = SHORT1FROMMP(mpParm1);
        usNotifyCode = SHORT2FROMMP(mpParm1);

        if (usID == ID_SPINBUTTONDAY || usID ==
           ID_SPINBUTTONMONTH || usID == ID_SPINBUTTONYEAR)
        {
```

```
            if (usNotifyCode == SPBN_ENDSPIN)
            {

               WinQueryWindowRect(hwndWnd,
                                  &rclWindow);
               rclWindow.yBottom = (rclWindow.yTop-
                  rclWindow.yBottom)/3*2;

               WinInvalidateRect(hwndWnd,
                                 &rclWindow,
                                 FALSE);
            }
            else
            {
               return  WinDefWindowProc(hwndWnd,
                                        ulMsg,
                                        mpParm1,

                                        mpParm2);
            }                          /* endif                */
         }
         else
         {
            return  WinDefWindowProc(hwndWnd,
                                     ulMsg,
                                     mpParm1,

                                     mpParm2);
         }                             /* endif                */
      }
      break;
   case  WM_COMMAND :
      {
         HWND              hwndActive;
         USHORT            usFocusID;

         if (SHORT1FROMMP(mpParm2) == CMDSRC_ACCELERATOR)
         {

            hwndActive = WinQueryFocus(HWND_DESKTOP);
            usFocusID = WinQueryWindowUShort(hwndActive,
                                             QWS_ID);

            if (SHORT1FROMMP(mpParm1) == IDK_TAB)
            {
               usFocusID++;

               if (usFocusID > LAST_CONTROL)
               {
                  usFocusID = FIRST_CONTROL;
               }                       /* endif                */
               hwndActive = WinWindowFromID(hwndWnd,
                                            usFocusID);
               WinSetFocus(HWND_DESKTOP,
                           hwndActive);

            }
            else

               if (SHORT1FROMMP(mpParm1) == IDK_BACKTAB)
               {
                  usFocusID--;

                  if (usFocusID < FIRST_CONTROL)
                  {
                     usFocusID = LAST_CONTROL;
                  }                    /* endif                */
```

```
                hwndActive = WinWindowFromID(hwndWnd,
                                             usFocusID);
                WinSetFocus(HWND_DESKTOP,
                            hwndActive);

            }                       /* endif                */
        }                           /* endif                */
    }
    break;
case  WM_PAINT :
    {
        HPS             hpsPaint;
        RECTL           rclBox;
        CHAR            achMonth[15];
        CHAR            achDay[3];
        CHAR            achYear[5];
        CHAR            achMsg[128];

        hpsPaint = WinBeginPaint(hwndWnd,
                                 NULLHANDLE,
                                 NULL);

        WinQueryWindowRect(hwndWnd,
                           &rclBox);

        rclBox.yBottom = (rclBox.yTop-rclBox.yBottom)/3*2;

        WinFillRect(hpsPaint,
                    &rclBox,
                    CLR_WHITE);

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONMONTH,
                          SPBM_QUERYVALUE,
                          MPFROMP(achMonth),
                          MPFROM2SHORT(sizeof(achMonth),
                                       SPBQ_DONOTUPDATE));

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONDAY,
                          SPBM_QUERYVALUE,
                          MPFROMP(achDay),
                          MPFROM2SHORT(sizeof(achDay),
                                       SPBQ_DONOTUPDATE));

        WinSendDlgItemMsg(hwndWnd,
                          ID_SPINBUTTONYEAR,
                          SPBM_QUERYVALUE,
                          MPFROMP(achYear),
                          MPFROM2SHORT(sizeof(achYear),
                                       SPBQ_DONOTUPDATE));

        sprintf(achMsg,
                "SpinButton's set to: %s %s, %s",
                achMonth,
                achDay,
                achYear);

        WinDrawText(hpsPaint,
                    -1,
                    achMsg,
                    &rclBox,
                    0,
                    0,
                    DT_CENTER|DT_VCENTER|DT_TEXTATTRS);

        WinEndPaint(hpsPaint);
    }
    break;
```

```
      case  WM_ERASEBACKGROUND :
         return  MRFROMSHORT(TRUE);

      default  :
         return  WinDefWindowProc(hwndWnd,
                                  ulMsg,
                                  mpParm1,

                                  mpParm2);
   }                                   /* endswitch            */
   return  MRFROMSHORT(FALSE);
}
```

## SPIN.RC

```
#include <os2.h>
#include "spin.h"

ACCELTABLE ID_WINDOW
{
    VK_TAB, IDK_TAB, VIRTUALKEY
    VK_BACKTAB, IDK_BACKTAB, VIRTUALKEY, SHIFT
}
```

## SPIN.H

```
#define ID_WINDOW                101
#define ID_SPINBUTTONMONTH       102
#define ID_SPINBUTTONDAY         103
#define ID_SPINBUTTONYEAR        104
#define IDK_TAB                  105
#define IDK_BACKTAB              106
#define FIRST_CONTROL            ID_SPINBUTTONMONTH
#define LAST_CONTROL             ID_SPINBUTTONYEAR
```

## SPIN.MAK

```
SPIN.EXE:                        SPIN.OBJ \
                                 SPIN.RES
        LINK386 @<<
SPIN
SPIN
SPIN
OS2386
SPIN
<<
        RC SPIN.RES SPIN.EXE

SPIN.RES:                        SPIN.RC \
                                 SPIN.H
        RC -r SPIN.RC SPIN.RES

SPIN.OBJ:                        SPIN.C \
                                 SPIN.H
        ICC -C+ -Kb+ -Ss+ SPIN.C
```

**SPIN.DEF**

```
NAME SPIN WINDOWAPI

DESCRIPTION 'Spin button example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

## Accelerator Keys

In this example, we create three spin buttons directly on the client window. However, the big drawback to using a client window and not a dialog box as the parent is that you lose a lot of the keyboard handling of the dialog box. The dialog box procedure automates the moving from control to control when the user hits the TAB and BACKTAB key. We want our spin buttons to do this also, so we will emulate the TAB key handling using accelerator keys.

Accelerator keys are a shortcut keystroke that causes some action to happen immediately. In Presentation Manager programming lingo, a WM_COMMAND message is posted whenever an accelerator key is pressed. Accelerator keys are covered in more detail in Chapter 12.

```
ACCELTABLE ID_WINDOW
{
    VK_TAB, IDK_TAB, VIRTUALKEY
    VK_BACKTAB, IDK_BACKTAB, VIRTUALKEY
}
```

Accelerator keys can be created dynamically or in a resource file. This example uses a resource file. Our resource file defines only two accelerator keys, VK_TAB and VK_BACKTAB.

## WM_CREATE Processing

In this example, we want to create the spin buttons directly on the client area of the window. The ideal time to create them is at the same time the window is created, in the WM_CREATE processing.

```
ulYearStyle =  SPBS_MASTER | SPBS_READONLY |
               SPBS_JUSTLEFT | SPBS_FASTSPIN |
               WS_VISIBLE ;
```

The variables *ulMonthStyle, ulDayStyle,* and *ulYearStyle* are used to hold the spin button styles. Each button is fairly similar. SPBS_READONLY indicates this spin button will be read-only. SPBS_JUSTLEFT will left-justify the spin button text. SPBS_FASTSPIN lets the user spin the buttons quickly by holding down the arrow keys. Two of the spin buttons will be servant spin buttons. The Year spin button will be the master, and the up and down arrows are located to the right of that button.

```
lHeight = WinQuerySysValue ( HWND_DESKTOP,
                             SV_CYSCREEN ) / 2 ;
lWidth  = WinQuerySysValue ( HWND_DESKTOP,
                             SV_CXSCREEN ) / 2 ;
```

The next step is to determine where we will place the spin buttons in the client area. In the WM_CREATE message, the client area has a size of 0, 0. This can make it very difficult to try to guess the size. However, in this case we can cheat. We know what proportion the client window is of the screen size; so we use the screen height and width, and divide by two.

```
xPosition = lWidth / 5;
yPosition = lHeight / 3;
yHeight = 50;
```

The x and y coordinates are calculated by using one-fifth the client area width, and one-third the client area height.

The spin buttons are created using *WinCreateWindow* with the class WC_SPINBUTTON.

```
WinSendDlgItemMsg ( hwndWnd,
                    ID_SPINBUTTONDAY,
                    SPBM_SETARRAY,
                    MPFROMP ( achDayArray ) ,
                    MPFROMSHORT ( 31 ) ) ;

WinSendDlgItemMsg ( hwndWnd,
                    ID_SPINBUTTONDAY,
                    SPBM_SETMASTER,
                    MPFROMHWND ( WinWindowFromID (
                                 hwndWnd,
                                 ID_SPINBUTTONYEAR )),
                    0 ) ;
```

The last step in creating the spin buttons is to initialize them. The buttons with the IDs ID_SPINBUTTONDAY and ID_SPINBUTTONMONTH need to be told exactly who their master is, since they are only servant spin buttons. The message SPBM_SETMASTER will do this. *mpParm1* is the master window handle, and *mpParm2* is not used. Each different button also has an array of data that needs to be associated with it. These arrays are defined in SPIN.C. To associate the array, we will send the spin button the message SPBM_SETARRAY. *mpParm1* is a pointer to the array, and *mpParm2* is the number of items in the array.

# WM_CONTROL Processing

The owner of control windows will receive a WM_CONTROL message when something important has happened. It just so happens that one of these messages will be able to tell the client window that the spin button has finished spinning. When that happens, we want to update the status string at the top of the client window.

```
usID = SHORT1FROMMP ( mpParm1 ) ;
usNotifyCode = SHORT2FROMMP ( mpParm1 ) ;

if ( usID == ID_SPINBUTTONDAY ||
   usID == ID_SPINBUTTONMONTH ||
   usID == ID_SPINBUTTONYEAR ) {

   if ( usNotifyCode == SPBN_ENDSPIN ) {

      WinQueryWindowRect ( hwndWnd, &rclWindow ) ;
      rclWindow.yBottom = ( rclWindow.yTop -
                            rclWindow.yBottom ) / 3 * 2 ;

      WinInvalidateRect ( hwndWnd,
                          &rclWindow,
                          FALSE ) ;
```

*mpParm1* in the WM_CONTROL message contains all the information we need to know about the spin buttons. The first SHORT is the ID of the control that sent the WM_CONTROL message. The second SHORT is a notification code that is specific to that type of control. It's a good idea to look at the IDs of the window sending the message in order to make sure you've got the right window. The only notification code that we're interested in is SPBN_ENDSPIN. If we receive that message, we want to make the client area repaint the status area. This area takes up the top third of the client window. First we find the rectangle we want to repaint, then we use *WinInvalidateRect* to force a repaint of that area.

# WM_COMMAND Processing

The WM_COMMAND processing is where we handle the processing of the accelerator keys.

```
if ( SHORT1FROMMP ( mpParm2 ) == CMDSRC_ACCELERATOR ) {

   hwndActive = WinQueryFocus ( HWND_DESKTOP ) ;
   usFocusID = WinQueryWindowUShort ( hwndActive,
                                      QWS_ID ) ;
```

The lower bytes of *mpParm2* contain the command type ID. This can contain values such as: CMDSRC_PUSHBUTTON, CMDSRC_MENU, CMDSRC_FONTDLG, CMDSRC_FILEDLG, CMDSRC_OTHER, or the value that we're interested in, CMDSRC_ACCELERATOR. The lower bytes of *mpParm1* contains the accelerator key command ID that we specified as the *cmd* element of the ACCEL structures. This information tells us whether the user hit the TAB key or BACKTAB key. *WinQueryFocus* is used to determine what spin button to use as a starting point. The window ID is retrieved using *WinQueryWindowUShort*. Our window IDs are consecutive numbers, so it is a simple matter to determine which spin button should have the focus next.

```
   if ( SHORT1FROMMP ( mpParm1 ) == IDK_TAB ) {
      usFocusID ++ ;

      if ( usFocusID > LAST_CONTROL ) {
          usFocusID = FIRST_CONTROL ;
      } /* endif */

      hwndActive = WinWindowFromID ( hwndWnd,
                                     usFocusID ) ;
      WinSetFocus ( HWND_DESKTOP, hwndActive ) ;

   } else
```

If the accelerator key was a TAB key, we're moving forward. If the current spin button is the last one in the chain, or if the window ID received is out of the bounds of the spin buttons, we set the variable *usFocusID* to the first spin button, or else we just increment *usFocusD*.

```
   if ( SHORT1FROMMP ( mpParm1 ) == IDK_BACKTAB ) {
      usFocusID -- ;

      if ( usFocusID < FIRST_CONTROL ) {
          usFocusID = LAST_CONTROL ;
      } /* endif */

      hwndActive = WinWindowFromID ( hwndWnd,
                                     usFocusID ) ;
      WinSetFocus ( HWND_DESKTOP, hwndActive ) ;

   } /* endif */
} /* endif */
```

The same logic, reversed, is used if the accelerator key is the BACKTAB. Once the new window ID is determined, *WinSetFocus* will set the keyboard focus to the new window.

## WM_PAINT Processing

The text displaying the current selection of the spin buttons is displayed in the top third of the window. *WinQueryWindowRect* determines the size of the window, and then *WinFillRect* fills this part of the window with the color white (CLR_WHITE), effectively erasing this part of this window.

```
WinSendDlgItemMsg ( hwndWnd,
                    ID_SPINBUTTONYEAR,
                    SPBM_QUERYVALUE,
                    MPFROMP ( achYear ) ,
                    MPFROM2SHORT ( sizeof ( achYear ) ,
                       SPBQ_DONOTUPDATE )) ;

sprintf ( achMsg,
          "SpinButton's set to: %s %s, %s",
          achMonth,
          achDay,
          achYear ) ;

WinDrawText ( hpsPaint,
              -1,
              achMsg,
              &rclBox,
              0,
              0,
              DT_CENTER | DT_VCENTER | DT_TEXTATTRS ) ;
```

The message SPBM_QUERYVALUE will determine what the spin buttons are currently set at. All three spin buttons are queried, and their values are returned in a character string. These strings are used to create one string that will be displayed in the text portion of the window. *WinDrawText* is used to display the text, centered both horizontally and vertically, in the text portion of the window.

# Chapter 25

# Sliders

A slider control is a control designed for two purposes: to let a user adjust some value on a graduated scale and to serve as a progress indicator of a process. The slider is similar in function to an air-conditioning thermostat. It can be adjustable or read-only. There are two kinds of sliders: a *linear slider* and a *circular slider*. The circular slider was included with MMPM/2 in earlier versions of OS/2, but in Warp it is included in the base operating system. Figure 25.1 illustrates the different linear slider components.

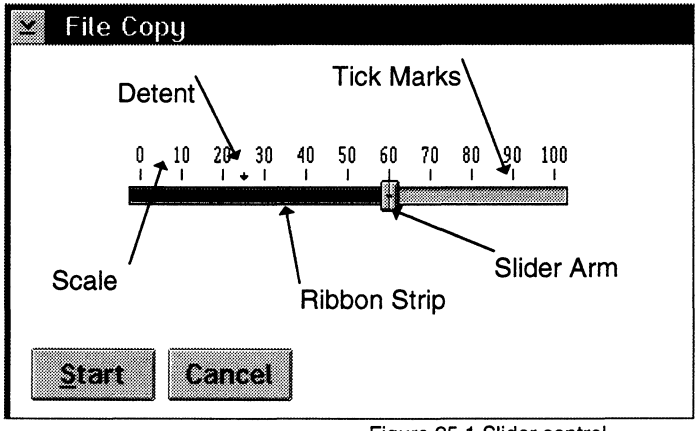



Figure 25.1 Slider control.

The slider arm is the "handle" that is used to select new values along the slider shaft. The arm can be dragged with a mouse or moved with the cursor keys.

The piece of color that sits to the right or left of the slider arm (depending on slider orientation) is called a ribbon strip.

The graduations marked along the slider shaft are called tick marks. They can be labeled with text or left blank.

A detent is a little arrow that marks some point of interest along the scale.

A slider scale can sit above or below the slider shaft, or a slider can use two scales. Table 25.1 presents the available slider styles.

# Linear Slider Styles

Table 25.1 Slider Styles

| Style | Description |
|---|---|
| SLS_HORIZONTAL | The default orientation of the slider. When is slider is of style SLS_HORIZONTAL, the slider arm will move left and right. The scale is placed above the shaft, below the shaft, or above and below the shaft. |
| SLS_VERTICAL | Positions the slider vertically. The arm will move up and down the shaft, and the scale(s) are placed vertically along the shaft, similar to a thermometer. |
| SLS_CENTER | Centers the slider in the slider window. This is the default. |
| SLS_BOTTOM | Positions the slider at the bottom of the slider window. Not valid for vertical sliders. |
| SLS_TOP | Positions the slider at the top of the slider window. Not valid for vertical sliders. |
| SLS_LEFT | Positions the slider at the left of the slider window. Not valid for horizontal sliders. |
| SLS_RIGHT | Positions the slider at the right of the slider window. Not valid for horizontal sliders. |
| SLS_PRIMARYSCALE1 | Positions the scale above the slider for a horizontal slider and to the right of the slider for a vertical slider. The increments and detents are also positioned correspondingly. This is the default style. |
| SLS_PRIMARYSCALE2 | The inverse of the previous style. Scales for horizontal sliders are placed on the bottom, and scales for vertical sliders are placed on the left. |
| SLS_HOMELEFT | Causes the slider arm to be placed on the left edge of the slider when it is in base, or 0, position. This style can be used only with horizontal sliders. |
| SLS_HOMERIGHT | Causes the slider arm to be placed on the right edge of the slider when it is in base, or 0, position. This style can be used only with horizontal sliders. |
| SLS_HOMEBOTTOM | Causes the slider arm to be placed on the bottom of the slider when it is in base, or 0, position. This style can be used only with vertical sliders. |
| SLS_HOMETOP | Causes the slider arm to be placed on the top of the slider when it is in base, or 0, position. This style can be used only with vertical sliders. |
| SLS_BUTTONSLEFT | Includes slider buttons that will be placed to the left of the slider. Clicking on the buttons moves the slider arm one position in the specified direction. This style can be used only with horizontal sliders. |
| SLS_BUTTONSRIGHT | Includes slider buttons that will be placed to the right of the slider. Clicking on the buttons moves the slider arm one position in the specified direction. This style can be used only with horizontal sliders. |
| SLS_BUTTONSBOTTOM | Includes slider buttons that will be placed on the bottom of the slider. Clicking on the buttons moves the slider arm one position in the specified direction. This style can be used only with vertical sliders. |
| SLS_BUTTONSTOP | Includes slider buttons that will be placed on the top of the slider. Clicking on the buttons moves the slider arm one position in the specified direction. This style can be used only with vertical sliders. |
| SLS_SNAPTOINCREMENT | Causes the slider arm to snap to the nearest scale increment as it is moved. |

| Style | Description |
|---|---|
| SLS_READONLY | Prevents the user from interacting with the slider. The slider will contain no slider buttons and no detents, and the slider arm is narrower than non-read-only sliders. |
| SLS_OWNDERDRAW | Causes WM_DRAWITEM messages to be sent to the application when the slider needs to be painted. |
| SLS_RIBBONSTRIP | Provides a ribbon strip in the middle of the slider shaft. |

## Creating a Linear Slider

A slider can be created either by using *WinCreateWindow* or by specifying a slider control in the resource file. The following code demonstrates using the function *WinCreateWindow* to create a slider.

```
SLDCDATA structSliderData;
HWND hwndSlider;
ULONG ulSliderStyle;

structSliderData.cbSize = sizeof( SLDCDATA);
/* size of control data structure */
structSliderData.usScale1Increments = 10;
/* number of increments on Scale 1 */
structSliderData.usScale1Spacing = 6;
/* number of pixels between Scale 1 increments */
structSliderData.usScale2Increments = 0;
/* number of increments on Scale2 */
structSliderData.usScale2Spacing = 0;
/* number of pixels between Scale 2 increments */
ulSliderStyle =  WS_VISIBLE  |  SLS_BUTTONSLEFT |  SLS_SNAPTOINCREMENT;

hwndSlider = WinCreateWindow(
       hwndParent,     /* parent window  */
       WC_SLIDER,      /* slider class */
       (PSZ)0,         /* window text - none here */
       ulSliderStyle, /* slider styles */
       50,             /* x */
       50,             /* y */
       240,            /* cx */
       50,             /* cy */
       hwndOwner,      /* owner window */
       HWND_TOP,       /* Z-order */
       IDS_SLIDER,     /* slider ID */
       &structSliderData,     /* pointer to SLDCDATA structure */
       NULL ); /* presentation parameters */
```

When specifying a slider in a resource file, the following statements are necessary.

```
CONTROL          "", IDS_SLIDER, 50, 50, 240, 50, WC_SLIDER,
     SLS_SNAPTOINCREMENT | SLS_BUTTONSLEFT | WS_VISIBLE
                    CTLDATA 12, 0, 11, 0, 0, 0
```

The CTLDATA line represents the slider control data structure. The first two numbers represent the ULONG value that is the size of the structure. The next number is the number of divisions on scale one. The fourth number indicates auto-spacing if 0, or the number of pixels between increments if a nonzero value is used. The last two numbers represent the number of divisions on scale two and spacing on scale two, respectively.

**Gotcha!**

Auto-sizing for the increments between the tick marks is not ideal. The slider divides the number of increments into the size of the slider. This means that long-tick-mark text will fall off the edges. Unless short-tick-mark text is present, it is best for programmers to size the slider increments themselves.

## A Linear Slider Example Program

The following program is an example of a read-only slider that is used as a progress indicator. The slider is owner-drawn (SLS_OWNERDRAW) in order to change the ribbon strip color from ordinary gray to blue. The program is designed to create a backup copy of the source code. This program does not create a standard window as the parent of the dialog; instead, a dialog is created at program startup. HWND_DESKTOP is used as the parent and owner of the dialog. This is a perfectly legitimate way of designing small programs that don't need the extra functionality of a client window space or menu.

**SLIDER.C**

```
#define  INCL_DOSFILEMGR
#define  INCL_DOSMEMMGR
#define  INCL_WINDIALOGS
#define  INCL_WINMENUS
#define  INCL_WINSTDSLIDER
#define  INCL_WINSYS
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "slider.h"
#define  COPY_FILE      "SLIDER.C"
#define  BACKUP_FILE    "SLIDER.BAK"
BOOL CopyFile(HWND hwndSlider);

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinDlgBox(HWND_DESKTOP,
             HWND_DESKTOP,
             DlgProc,
             NULLHANDLE,
             IDD_FCOPYDLG,
             NULL);

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY DlgProc(HWND hwndDlg,ULONG ulMsg,MPARAM mpParm1,
                         MPARAM mpParm2)
{
   switch (ulMsg)
```

```
{
   case  WM_INITDLG :
      {
         CHAR              achFont[16];
         USHORT            usIndex;
         CHAR              achMessage[64];

         /*************************************************/
         /* Set the size of the tick marks and change font  */
         /* to smaller                                      */
         /*************************************************/

         WinSendDlgItemMsg(hwndDlg,
                           IDS_SLIDER,
                           SLM_SETTICKSIZE,
                           MPFROM2SHORT(SMA_SETALLTICKS,
                                        7),
                           0);

         strcpy(achFont,
                "8.Tms Rmn");
         WinSetPresParam(WinWindowFromID(hwndDlg,
                                         IDS_SLIDER),
                         PP_FONTNAMESIZE,
                         strlen(achFont)+1,
                         achFont);

         for (usIndex = 0; usIndex < 11; usIndex++)
         {
            sprintf(achMessage,
                    "%d%%",
                    usIndex          *10);

            WinSendDlgItemMsg(hwndDlg,
                              IDS_SLIDER,
                              SLM_SETSCALETEXT,
                              MPFROMSHORT(usIndex),
                              MPFROMP(achMessage));
         }                          /* endfor               */
      }
      break;

   case  WM_COMMAND :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  IDP_START :
            CopyFile(WinWindowFromID(hwndDlg,
                                     IDS_SLIDER));

            WinAlarm(HWND_DESKTOP,
                     WA_NOTE);
            WinMessageBox(HWND_DESKTOP,
                          hwndDlg,
                          "Backup is Complete",
                          "Status",
                          0,
                          MB_OK|MB_INFORMATION);
            break;
```

```
            case  IDP_CANCEL :
               WinDismissDlg(hwndDlg,
                             FALSE);
               break;

            default  :
               return  WinDefDlgProc(hwndDlg,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

         }                              /* endswitch            */
         break;

      case  WM_DRAWITEM :
         {
         POWNERITEM       poiItem;

         /************************************************/
         /* get the OWNERITEM structure from mpParm2 and    */
         /* user to draw the filled ribbon strip in blue    */
         /************************************************/

         poiItem = (POWNERITEM)PVOIDFROMMP(mpParm2);

         switch (poiItem->idItem)
         {
            case  SDA_RIBBONSTRIP :
               WinFillRect(poiItem->hps,
                           &poiItem->rclItem,
                           CLR_BLUE);
               return  MRFROMSHORT(TRUE);
            default  :
               return  WinDefDlgProc(hwndDlg,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);

         }                              /* endswitch            */
         }

      default  :
         return  WinDefDlgProc(hwndDlg,
                               ulMsg,
                               mpParm1,
                               mpParm2);

   }                                    /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

BOOL CopyFile(HWND hwndSlider)
{
   APIRET           arRc;
   FILESTATUS3      fsStatus;
   PBYTE            pbBuffer;
   HFILE            hfRead;
   ULONG            ulAction;
   HFILE            hfWrite;
   ULONG            ulSzBlock;
   USHORT           usIndex;
   ULONG            ulBytesRead;
   ULONG            ulBytesWritten;
```

```
arRc = DosQueryPathInfo(COPY_FILE,
                        FIL_STANDARD,
                        (PVOID)&fsStatus,
                        sizeof(fsStatus));

if (!arRc)
{
   ulSzBlock = fsStatus.cbFile/10+1;

   pbBuffer = (PBYTE)malloc(ulSzBlock);

   /*******************************************************/
   /* Open up the file for reading                        */
   /*******************************************************/

   arRc = DosOpen(COPY_FILE,
                  &hfRead,
                  &ulAction,
                  0,
                  FILE_NORMAL,
                  FILE_OPEN,
                  OPEN_ACCESS_READONLY|OPEN_SHARE_DENYWRITE,
                  0);

   /*******************************************************/
   /* Open up the backup file for write                   */
   /*******************************************************/

   arRc = DosOpen(BACKUP_FILE,
                  &hfWrite,
                  &ulAction,
                  0,
                  FILE_NORMAL,
                  FILE_CREATE,
                  OPEN_ACCESS_WRITEONLY|
                     OPEN_SHARE_DENYREADWRITE,
                  0);

   for (usIndex = 1; usIndex < 11; usIndex++)
   {

      /*****************************************************/
      /* Read a block                                      */
      /*****************************************************/

      DosRead(hfRead,
              pbBuffer,
              ulSzBlock,
              &ulBytesRead);

      /*****************************************************/
      /* Write a block                                     */
      /*****************************************************/

      DosWrite(hfWrite,
               pbBuffer,
               ulBytesRead,
               &ulBytesWritten);

      /*****************************************************/
      /* Tell the slider to move                           */
      /*****************************************************/
```

```
         WinSendMsg(hwndSlider,
                    SLM_SETSLIDERINFO,
                    MPFROM2SHORT(SMA_SLIDERARMPOSITION,
                                 SMA_INCREMENTVALUE),
                    MPFROMSHORT(usIndex));
      }                                 /* endfor                */

      /*********************************************************/
      /* Clean up                                              */
      /*********************************************************/

      DosClose(hfRead);
      DosClose(hfWrite);
      free(pbBuffer);
      return  TRUE;
   }

   else
   {
      return  FALSE;
   }                                    /* endif                 */
}
```

**SLIDER.RC**

```
#include <os2.h>
#include "slider.h"

DLGTEMPLATE IDD_FCOPYDLG LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG  "File Copy", IDD_FCOPYDLG, 67, 80, 324, 104,
      WS_VISIBLE,
      FCF_SYSMENU | FCF_TITLEBAR
   {
      LTEXT "Progress Indicator", -1, 115, 30, 80, 12
      CONTROL  "", IDS_SLIDER, 12, 43, 300, 46, WC_SLIDER,
         SLS_HORIZONTAL | SLS_OWNERDRAW | SLS_CENTER |
         SLS_SNAPTOINCREMENT | SLS_READONLY | SLS_RIBBONSTRIP |
         SLS_HOMELEFT | SLS_PRIMARYSCALE1 | WS_GROUP | WS_TABSTOP |
         WS_VISIBLE
         CTLDATA 12, 0, 11, 26, 0, 0
      PUSHBUTTON "~Start", IDP_START, 6, 4, 40, 14
      PUSHBUTTON "Cancel", IDP_CANCEL, 49, 4, 40, 14
   }
}
```

**SLIDER.H**

```
#define IDD_FCOPYDLG            256
#define IDS_SLIDER              512
#define IDP_START               513
#define IDP_CANCEL              514
```

**SLIDER.MAK**

```
SLIDER.EXE:                     SLIDER.OBJ \
                                SLIDER.RES
        LINK386 @<<
SLIDER
SLIDER
SLIDER
OS2386
SLIDER
<<
        RC SLIDER.RES SLIDER.EXE

SLIDER.RES:                     SLIDER.RC \
                                SLIDER.H
        RC -r SLIDER.RC SLIDER.RES

SLIDER.OBJ:                     SLIDER.C \
                                SLIDER.H
        ICC -C+ -Kb+ -Ss+ SLIDER.C
```

**SLIDER.DEF**

```
NAME SLIDER WINDOWAPI

DESCRIPTION 'Slider example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

# Initalizing the Slider

```
WinSendDlgItemMsg ( hwndDlg,
                    IDS_SLIDER,
                    SLM_SETTICKSIZE,
                    MPFROM2SHORT ( SMA_SETALLTICKS, 7 ) ,
                    0 ) ;
```

In the WM_INITDLG, a SLM_SETTICKSIZE message is sent to the slider window to set the height of the tick marks. This is different from the item in the CTRLDATA statement in the resource file that sets the width between the tick marks.

```
strcpy ( achFont, "8.Tms Rmn" ) ;
WinSetPresParam ( WinWindowFromID ( hwndDlg, IDS_SLIDER ),
                  PP_FONTNAMESIZE,
                  strlen ( achFont ) + 1,
                  achFont ) ;
```

*WinSetPresParam* is used to change the system font of the slider to something nicer and smaller, "8.Tms Rmn"; this can be useful when your slider text runs over the edges of the slider.

```
for ( usIndex = 0 ; usIndex < 11 ; usIndex ++ ) {
   sprintf ( achMessage, "%d%%", usIndex * 10 ) ;

   WinSendDlgItemMsg ( hwndDlg,
                       IDS_SLIDER,
                       SLM_SETSCALETEXT,
                       MPFROMSHORT ( usIndex ) ,
                       MPFROMP ( achMessage )) ;
}
```

Next, the tick marks are labeled with 11 percentage markers by sending the message SLM_SETSCALETEXT. The first parameter is the division number to set, and the second parameter is the string to use.

**Gotcha!**
One little note here: SLM_SETSCALETEXT does not recognize SMA_SETALLTICKS in *mpParm2*. (Not that anyone will want to set all the tick marks with the same text very often, but just in case it was desired. )

The WM_COMMAND processing is very simple: When the user pushes the START button, a WM_COMMAND message is sent to the dialog process. The function *CopyFile* is called to back up the file. If the CANCEL button is pressed, the dialog is dismissed, and the process exits.

## Using an Ownerdrawn Slider

Because the slider is of style SLS_OWNERDRAW, the dialog procedure also will receive the WM_DRAWITEM message. *mpParm2* contains a pointer to the owneritem structure. The structure is the same as the OWNERITEM structure covered in Chapter 15. For a slider the *idItem* can contain one of four different values: SDA_RIBBONSTRIP, SDA_SLIDERSHAFT, SDA_BACKGROUND, or SDA_SLIDERARM.

```
poiItem = (POWNERITEM) PVOIDFROMMP ( mpParm2 ) ;

switch ( poiItem->idItem ) {
case SDA_RIBBONSTRIP:
   WinFillRect ( poiItem->hps,
                 &poiItem->rclItem,
                 CLR_BLUE ) ;
   return MRFROMSHORT ( TRUE ) ;
```

In this case, the program checks to see if the item needing to be drawn, *poiItem->idItem*, is SDA_RIBBONSTRIP. If it isn't, we break out of the switch statement. If it is SDA_RIBBONSTRIP, *WinFillRect* is called to fill the RECTL structure, *poiItem->rclItem* with CLR_BLUE. After the area is filled we return TRUE, to indicate that we've already drawn the area, and there's no drawing left to do.

The last part of the program is the function *CopyFile*, which is used to copy the file, SLIDER.C, to the file SLIDER.BAK. This example copies the file in 10 equal increments, in order to demonstrate a progress indicator. Please note that there is an OS/2 function, *DosCopy*, that will do all this in one function call, but for this example we'll do our own copying. First, *DosQueryPathInfo* is used to make sure the file exists and to find the file size. A buffer, *pbBuffer*, is allocated to serve as the holding place for bytes read and then written. Next, *DosOpen* is called to open both files. The file functions are covered in more detail in Chapter 4; see this chapter for more information on the parameters used in *DosOpen, DosQueryPathInfo*, and *DosFindFirst.*

```
WinSendMsg ( hwndSlider,
             SLM_SETSLIDERINFO,
             MPFROM2SHORT ( SMA_SLIDERARMPOSITION,
                SMA_INCREMENTVALUE ) ,
             MPFROMSHORT ( usIndex )) ;
```

We will copy the file in 10 pieces. As each piece is copied, a message is sent to the slider to set the progress indicator to the next value. The message is SLM_SETSLIDERINFO. The first parameter is made up of two SHORTS. The first value is the type of information that is being set. Possible values are:

- SMA_SHAFTDIMENSIONS
- SMA_SHAFTPOSITION
- SMA_SLIDERARMDIMENSIONS
- SMA_SLIDERARMPOSITION.

We will use SMA_SLIDERARMPOSITION. The second value, SMA_INCREMENTVALUE, tells the slider to change the slider arm position using tick marks instead of pixels. The second parameter indicates the number of the tick mark at which to set the slider arm.

## Circular Sliders

A circular slider is used to provide a user interface similar to a volume control on a stereo. The user selects a new value by using the slider arm that radiates from the center of the circle, or by using the incremental and decremental buttons on either side of the slider. The circular slider is useful when there is not much screen space. Figure 25.2 illustrates a circular slider and Table 25.2 specifies circular slider styles.

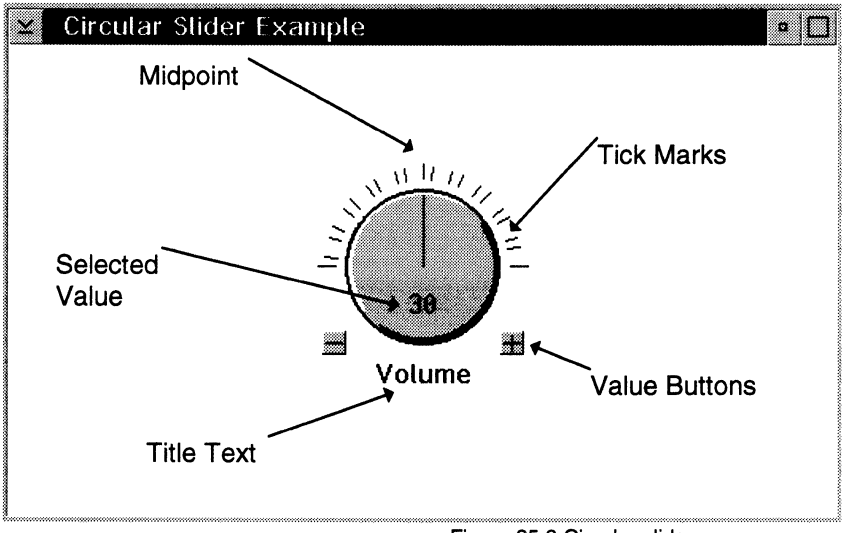



Figure 25.2 Circular slider.

## Circular Slider Styles

Table 25.2 Circular Slider Styles

| Style | Description |
|---|---|
| CSS_360 | The slider will have values extending a full 360 degrees (a full circle). The default is 180 degrees (a semicircle). See Figure 25.3 for an example of this style. |
| CSS_CIRCULARVALUE | A circular "thumb" is used, rather than a slider arm, to display the currently selected value. See Figure 25.4 for an example of this style. |

| Style | Description |
|---|---|
| CSS_MIDPOINT | The midpoint and end-point tick marks are made larger than the other tick marks. |
| CSS_NOBUTTON | No increment and decrement buttons are displayed. The default is to include the buttons. |
| CSS_NONUMBER | No numeric indicator of the dial's currently selected value is included. The default is to include the indicator. |
| CSS_NOTEXT | No title is displayed beneath the dial. The default is to include the title. |
| CSS_POINTSELECT | The user can use the mouse to select a value, and the slider arm instantly moves the the new value. The default method is for the slider arm to scroll through the slider tick marks sequentially. |
| CSS_PROPORTIONALTICKS | The tick mark length is calculated as a percentage of the radius of the dial. |

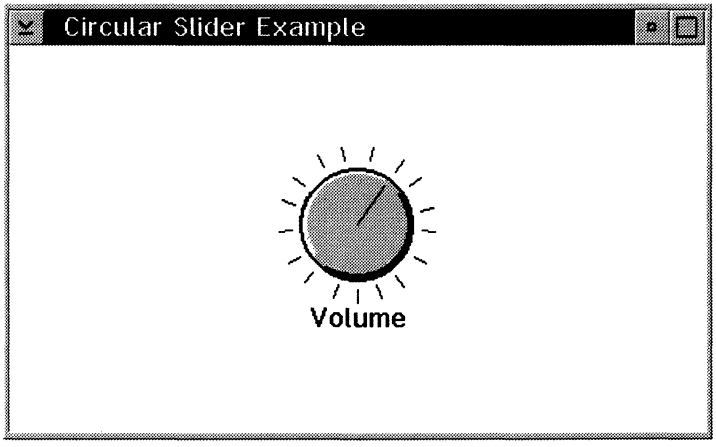



Figure 25.3 Circular slider with CSS_360 style.

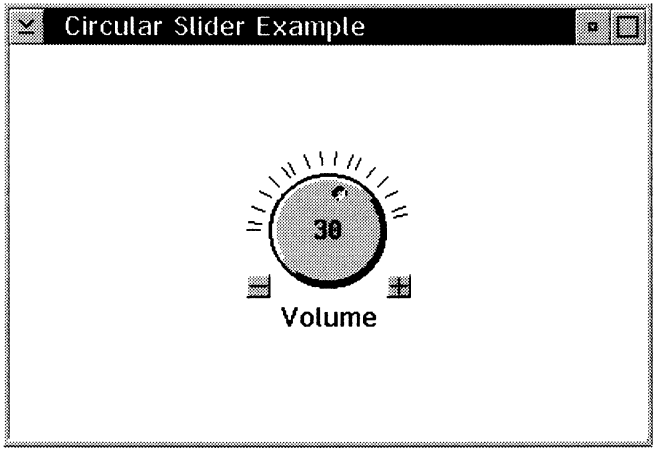



Figure 25.4 Circular slider with CSS_CIRCULARVALUE style.

## Creating a Circular Slider

A circular slider can be created using a resource file or the function *WinCreateWindow*. The following code shows a sample resource definition.

```
CONTROL "Volume",
        ID_VOLUME,
        10, 10, 100, 100,
        WC_CIRCULARSLIDER,
        WS_VISIBLE | CSS_360
```

or you can use the *WinCreateWindow* function to create a slider dynamically:

```
hwndCircle = WinCreateWindow( hwndClient,
                              WC_CIRCULARSLIDER,
                              "Volume",
                              WS_VISIBLE | CSS_360,
                              10, 10, 100, 100,
                              hwndClient,
                              ID_VOLUME,
                              NULL,
                              NULL ) ;
```

**Gotcha!**
The documentation for the Warp Toolkit indicates that *WinRegisterCircularSlider* must be called to register the circular slider class before a circular slider can be created. This is wrong. There is no *WinRegisterCircularSlider* defined. Earlier versions of the circular slider previously belonged to the MMPM/2 Toolkit and had to be registered before the class could be used. Obviously, someone forgot to update the manual.

## A Circular Slider Example Program

The following is a simple example program to create a circular slider.

**CIRCLE.C**

```
#define  INCL_WIN
#define  INCL_GPIBITMAPS
#include <os2.h>
#include "circle.h"
#include <stdlib.h>
#include <stdio.h>
#define  CLS_CLIENT    "ClientClass"

 /* Procedure Prototype                                          */

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mp1,MPARAM mp2);

INT main(VOID)
{

   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;
   LONG             lWidth,lHeight;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    0,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
      FCF_TASKLIST;

   /*************************************************************/
   /* create frame window                                       */
   /*************************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Circular Slider Example",
                                  0,
                                  NULLHANDLE,
                                  ID_FRAME,
                                  NULL);

   /*************************************************************/
   /* get screen height and width                               */
   /*************************************************************/

   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /*************************************************************/
   /* if failed, and set to default value                       */
   /*************************************************************/
```

```
  if (!lWidth || !lHeight)
   {
      lWidth = 640;
      lHeight = 480;
   }

   if (hwndFrame != NULLHANDLE)
   {

      /*******************************************************/
      /* set window position                                 */
      /*******************************************************/

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      10,
                      10,
                      lWidth/10*8,
                      lHeight/10*8,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile             */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mp1,MPARAM mp2)
{

   HWND             hwndCirc;

   switch (ulMsg)

   {

      case  WM_CREATE :
         {
            RGB2             rgb2 ;

            /*************************************************/
            /* Create circular slider control                */
            /*************************************************/
```

```
        hwndCirc = WinCreateWindow(hwndClient,
                                   WC_CIRCULARSLIDER,
                                   "Volume",
                                   CSS_MIDPOINT,
                                   0,
                                   0,
                                   0,
                                   0, /* Position & size   */
                                   hwndClient,
                                   HWND_TOP,
                                   ID_DIAL,
                                   NULL,
                                   NULL);

        /*************************************************/
        /* error checking                                */
        /*************************************************/

        if (!hwndCirc)
           return  MRFROMSHORT(FALSE);

        /*************************************************/
        /* change background color of slider to white    */
        /*************************************************/

        rgb2.bRed = 0xFF;
        rgb2.bGreen = 0xFF;
        rgb2.bBlue = 0xFF;
        rgb2.fcOptions = 0 ;

        WinSetPresParam(hwndCirc,
                        PP_BACKGROUNDCOLOR,
                        sizeof(RGB2),
                        &rgb2);

        /*************************************************/
        /* Specify range of values for circular slider   */
        /*************************************************/

        WinSendMsg(hwndCirc,
                   CSM_SETRANGE,
                   MPFROMLONG(0),
                   MPFROMLONG(60));

        /*************************************************/
        /* Specify scroll & tick mark increments         */
        /*************************************************/

        WinSendMsg(hwndCirc,
                   CSM_SETINCREMENT,
                   MPFROMLONG(10),
                   MPFROMLONG(0));

        /*************************************************/
        /* Set initial value                             */
        /*************************************************/

        WinSendMsg(hwndCirc,
                   CSM_SETVALUE,
                   MPFROMLONG(30),
                   NULL);

        return MRFROMSHORT(FALSE);
     }
  case  WM_SIZE :
     {

        SWP             swp;
```

```
            /*************************************************/
            /* resize circular slider as proportion of client  */
            /* window size                                     */
            /*************************************************/

            WinQueryWindowPos(hwndClient,
                              &swp);
            hwndCirc = WinWindowFromID(hwndClient,
                                       ID_DIAL);
            WinSetWindowPos(hwndCirc,
                            HWND_TOP,
                            swp.cx/4,
                            swp.cy/4,
                            swp.cx/2,
                            swp.cy/2,
                            SWP_MOVE|SWP_SHOW|SWP_SIZE);
            return MRFROMSHORT(FALSE);
         }
      case  WM_PAINT :
         {

            HPS              hps;

            /*************************************************/
            /* simple paint procedure to fill in client        */
            /* background                                      */
            /*************************************************/

            hps = WinBeginPaint(hwndClient,
                                0,
                                NULL);
            GpiErase(hps);
            WinEndPaint(hps);
            return MRFROMSHORT(FALSE);
         }
      default   :
         return (WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mp1,
                                  mp2));
   }
}
```

## CIRCLE.H

```
#define ID_FRAME        100
#define ID_DIAL         101
#define IDM_FILEEXIT    102
```

## CIRCLE.MAK

```
CIRCLE.EXE:                     CIRCLE.OBJ
        LINK386 @<<
CIRCLE
CIRCLE
CIRCLE
OS2386
CIRCLE
<<
```

```
CIRCLE.OBJ:                     CIRCLE.C \
                                CIRCLE.H
        ICC -C+ -Kb+ -Ss+ CIRCLE.C
```

**CIRCLE.DEF**

```
NAME CIRCLE WINDOWAPI

DESCRIPTION 'Circular Slider example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

## Initializing the Slider

```
WinSendMsg(hwndCirc,
           CSM_SETRANGE,
           MPFROMLONG(0),
           MPFROMLONG(50));

WinSendMsg(hwndCirc,
           CSM_SETINCREMENT,
           MPFROMLONG(10),
           MPFROMLONG(0));

WinSendMsg(hwndCirc,
           CSM_SETVALUE,
           MPFROMLONG(30),
           NULL);
```

Three items are initialized in the sample program. The range of the slider is set with the CSM_SETRANGE message. The first message parameter is the low value, 0. The second message parameter is the high value, 50. The CSM_SETINCREMENT message controls the amount of increments to move when the slider buttons are pressed. It also controls the number of tick marks to skip when drawing the slider tick marks. The first message parameter represents the increment movements of the slider buttons; the second message parameter sets the tick mark drawing at tick mark 0. The last message sent is CSM_SETVALUE. This message sets the currently selected value of the slider. In this example, the initial value is set at 30.

## Circular Slider Colors

```
rgb2.bRed = 0xFF;
rgb2.bGreen = 0xFF;
rgb2.bBlue = 0xFF;
rgb2.fcOptions = 0 ;

WinSetPresParam(hwndCirc,
                PP_BACKGROUNDCOLOR,
                sizeof(RGB2),
                &rgb2);
```

The circular slider responds only to two of the presentation parameters, PP_BACKGROUNDCOLOR and PP_BORDER. The background color is the area that sits outside the slider dial. In our example program, we will set the background color to white. Notice that this is PP_BACKGROUNDCOLOR and not PP_BACKGROUNDCOLORINDEX.

## Summary

The slider controls are a nice way to display a progress indicator or to provide the user with a large range of values to choose from. Sliders are simple controls to use in a program. Although they are not as customizable as might be desired, they still can be used in many instances. A volume control for a CD player program is an ideal use for a circular slider. A linear slider could be used as a thermostat.

# Chapter 26

# Font and File Dialogs

The font dialog and file dialog were introduced in OS/2 2.0 to provide two high-level functions that perform tasks that most programmers previously had written by hand at one time or another. The Font dialog is a dialog box with a listing of fonts and an example of each. The file dialog is a dialog box that contains a list of files on the end user's available drives. (See Figure 26.1.) Both functions can be reconfigured extensively by the programmer.

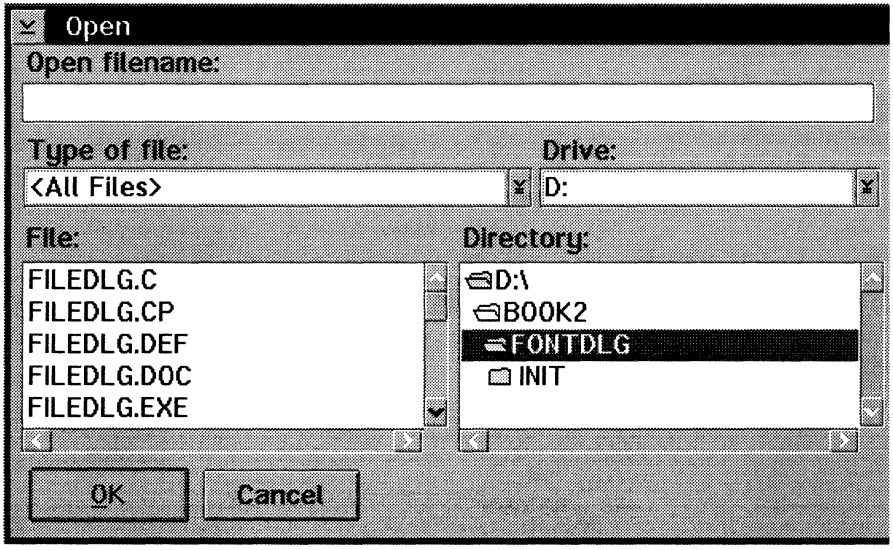



Figure 26.1 A file dialog box.

# The File Dialog

The file dialog can be created either as a "Save As..." or as an "Open" dialog. A list of all the controls in the file dialog is included in the Toolkit header file, PMSTDDLG.H, so that readers can add their own, or remove those they feel are unnecessary.

The meat of creating a file dialog structure is the FILEDLG structure. This structure, which follows, includes all the configurable options for the file dialog.

```
typedef struct _FILEDLG     /* filedlg */
{
   ULONG    cbSize;
   ULONG    fl;
   ULONG    ulUser;
   LONG     lReturn;
   LONG     lSRC;
   PSZ      pszTitle;
   PSZ      pszOKButton;
   PFNWP    pfnDlgProc;
   PSZ      pszIType;
   PAPSZ    papszITypeList;
   PSZ      pszIDrive;
   PAPSZ    papszIDriveList;
   HMODULE  hMod;
   CHAR     szFullFile[CCHMAXPATH];
   PAPSZ    papszFQFilename;
   ULONG    ulFQFCount;
   USHORT   usDlgId;
   SHORT    x;
   SHORT    y;
   SHORT    sEAType;
} FILEDLG;
   typedef FILEDLG *PFILEDLG;
```

The *cbSize* is the size of the FILEDLG structure. This field *must* be filled in. The *fl* field, which also *must* be filled in is the File Dialog flags used to describe the file dialog. Table 26.1 presents the possible values.

Table 26.1 File Dialog Flags

| Flag | Meaning |
|---|---|
| FDS_CENTER | The file dialog is centered within its owner. |
| FDS_CUSTOM | Use a custom-defined dialog box. |
| FDS_FILTERUNION | Use a union of extended attributes and file name filter. |
| FDS_HELPBUTTON | Include a HELP push button on the file dialog. |
| FDS_APPLYBUTTON | Include an APPLY push button on the file dialog. |
| FDS_PRELOAD_VOL_INFO | Load the volume information on the file dialog initialization. This can cause lengthy processing at startup. |
| FDS_MODELESS | The file dialog is modeless. |
| FDS_INCLUDE_EAS | Load the extended attribute information. |
| FDS_OPEN_DIALOG | File dialog is the "Open" dialog. |
| FDS_SAVEAS_DIALOG | File dialog is the "Save As..." dialog. |
| FDS_MULTIPLESEL | Multiple files can be selected from the list box. |
| FDS_ENABLEFILELB | If file dialog is the "Save As..." style, the list box of files is enabled, not disabled (the default). |

The *ulUser* field is 4 bytes of space that are available for the programmer to use. The *lReturn* field is the return code from the file dialog. This can be DID_OK, DID_CANCEL, or 0 if an error occurs. The *lSRC* field contains an FDS_ERR return code if an error occurs in the file dialog.

The *pszTitle* field is a pointer to a string that contains the title of the file dialog box window. If this is NULL, the title of the owner window is used. The *pszOKButton* field is a pointer to a string that contains the text for the OK push button. If this is NULL, "OK" is used. The *pfnDlgProc* field is a pointer to a user-defined dialog procedure. The function *WinDefFileDlgProc* can be used to call the default dialog procedure from the user-defined procedure.

The *pszIType* field is a pointer to a string containing a type of EA (extended attribute). Only files that contain this EA type will be shown in the list of available files. The field *papszITypeList* is an array of pointers that contain a list of EA types for filtering the available file list. This array must end with a NULL pointer. The *pszIDrive* field is a pointer to a string that contains the selected drive when the dialog is first made visible.

The field *papszIDriveList* is an array of pointers to strings that contain a list of drives to use as available drives. A NULL in this field will cause all available drives to be included in the list. This array must end with a NULL pointer. The *hMod* field is the handle of a .DLL that contains the dialog box resource to be used if a FDS_CUSTOM flag is specified. The character array *szFullFile* contains the filename of the initially selected file. On return, this field contains the fully qualified filename selected by the end user.

The field *papszFQFilename* is an array of pointers to fully qualified filenames. On return from the *WinFileDlg* function, this array contains all the files that the end user selected. The field *ulFQFCount* is the number of files selected by the user. The field *usDlgID* is the dialog resource ID of a file dialog to use as a replacement for the default file dialog. This field is used if FDS_CUSTOM is specified.

The fields *x* and *y* are the x,y coordinates to be used to place the file dialog. These fields are ignored when FDS_CENTER is used. The field *sEAType* is an index into the *papszITypeList* array that contains the EA type of the file that was selected.

## Special Considerations for Multiple File Selections

When a file dialog has the style FDS_MULTIPLESEL, multiple files can be selected from the file dialog. This causes a few events to happen:

- The number of files selected is returned in the field *ulFQFCount.*
- An array of pointers to the names of the selected files is returned in the *papszFQFilename* field.
- If the file dialog has allocated memory for these strings, the *lSRC* field will contain FDS_ERR_DEALLOCATE_MEMORY. This is a signal to the programmer that he or she needs to free the memory allocated for these strings with the *WinFreeFileDlgList* application.
- The first file selected will be contained in the *szFullFile* array.

The *WinFreeFileDlgList* function has only one parameter, the array of pointers to strings, *papszFQFilename.*

```
BOOL WinFreeFileDlgList(PAPSZ papszFQFilename);
```

# The FILEDLG Example Program

The example program FILEDLG creates a file dialog and prints the filename of the selected file on the client area.

**FILEDLG.C**

```
#define  INCL_WINSTDFILE
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "filedlg.h"
MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

BOOL FindFile(HWND hwndWnd,CHAR *pchFile);
VOID DisplayError(CHAR *pszText);

#define  CLS_CLIENT    "MyClass"
INT main(VOID)

{
   HMQ               hmqQueue;
   HAB               habAnchor;
   ULONG             ulFlags;
   HWND              hwndClient;
   BOOL              bLoop;
   QMSG              qmMsg;
   LONG              lWidth,lHeight;
   HWND              hwndFrame = NULLHANDLE;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW|CS_SYNCPAINT,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MENU|
      FCF_MINMAX;

   /*************************************************************/
   /* create frame and client window                            */
   /*************************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Font Dialog Example",
                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  &hwndClient);

   /*************************************************************/
   /* get screen height and width                               */
   /*************************************************************/
```

```
   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /************************************************************/
   /* set size and position of frame window                    */
   /************************************************************/

   if (hwndFrame)
   {
      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      lWidth/8,
                      lHeight/8,
                      lWidth/8*6,
                      lHeight/8*6,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCHAR            pchFile;

   switch (ulMsg)
   {

      case  WM_CREATE :
         pchFile = (PCHAR)calloc(1,
                                 CCHMAXPATH);
         if (!pchFile)
         {
            DisplayError("No memory could be allocated");
            return  MRFROMSHORT(TRUE);
         }                               /* endif                 */

         WinSetWindowPtr(hwndClient,
                         QWL_USER,
                         pchFile);
         break;

      case  WM_DESTROY :
         pchFile = WinQueryWindowPtr(hwndClient,
                                     0);
```

```
      if (pchFile)
      {
         free(pchFile);
      }                              /* endif                */
      break;

   case  WM_PAINT :
      {
         HPS               hpsPaint;
         RECTL             rclInvalid;
         CHAR              achText[CCHMAXPATH];

         hpsPaint = WinBeginPaint(hwndClient,
                                  NULLHANDLE,
                                  &rclInvalid);

         WinFillRect(hpsPaint,
                     &rclInvalid,
                     SYSCLR_WINDOW);
         pchFile = WinQueryWindowPtr(hwndClient,
                                     0);

         if (pchFile[0] != 0)
         {

            WinQueryWindowRect(hwndClient,
                               &rclInvalid);
            sprintf(achText,
                    "You have selected file %s",
                    pchFile);
            WinDrawText(hpsPaint,
                        -1,
                        achText,
                        &rclInvalid,
                        0,
                        0,
                        DT_CENTER|DT_VCENTER|DT_TEXTATTRS);
         }                              /* endif                */
         WinEndPaint(hpsPaint);
      }
      break;

   case  WM_COMMAND :

      switch (SHORT1FROMMP(mpParm1))
      {

         case  IDM_OPEN :
            pchFile = WinQueryWindowPtr(hwndClient,
                                        0);
            if (pchFile)
            {
               FindFile(hwndClient,
                        pchFile);
            }                           /* endif                */
            WinInvalidateRect(hwndClient,
                              NULL,
                              TRUE);
            break;

         case  IDM_EXIT :
            WinPostMsg(hwndClient,
                       WM_QUIT,
                       0,
                       0);
            break;
```

```
    default   :
                return  WinDefWindowProc(hwndClient,
                                         ulMsg,
                                         mpParm1,

                                         mpParm2);
            }                            /* endswitch            */
            break;

         default   :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,

                                     mpParm2);
      }                                  /* endswitch            */
      return  MRFROMSHORT(FALSE);
}

BOOL FindFile(HWND hwndClient,CHAR *pchFile)
{
   FILEDLG          fdFileDlg;

   memset(&fdFileDlg,
          0,
          sizeof(FILEDLG));

   fdFileDlg.cbSize = sizeof(FILEDLG);
   fdFileDlg.fl = FDS_CENTER|FDS_PRELOAD_VOLINFO|FDS_OPEN_DIALOG;

   if (WinFileDlg(HWND_DESKTOP,
                  hwndClient,
                  &fdFileDlg) != DID_OK)
   {
      DisplayError("WinFileDlg failed");
      return  FALSE;
   }                                    /* endif                */

   strcpy(pchFile,
          fdFileDlg.szFullFile);
   return  TRUE;
}

VOID DisplayError(CHAR *pszText)
{

   /*************************************************************/
   /* small function to display error string                    */
   /*************************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}
```

## FILEDLG.RC

```
#include <os2.h>
#include "filedlg.h"

MENU RES_CLIENT
{
   SUBMENU "~File", IDM_SUB1
   {
      MENUITEM "~New" , IDM_NEW
      MENUITEM "~Open File...", IDM_OPEN
      MENUITEM "~Close File", IDM_CLOSE
      MENUITEM "E~xit" , IDM_EXIT
   }
}
```

## FILEDLG.H

```
#define RES_CLIENT              256
#define IDM_SUB1                512
#define IDM_NEW                 513
#define IDM_OPEN                514
#define IDM_CLOSE               515
#define IDM_EXIT                516
```

## FILEDLG.MAK

```
FILEDLG.EXE:                    FILEDLG.OBJ \
                                FILEDLG.RES
        LINK386 @<<
FILEDLG
FILEDLG
FILEDLG
OS2386
FILEDLG
<<
        RC FILEDLG.RES FILEDLG.EXE

FILEDLG.RES:                    FILEDLG.RC \
                                FILEDLG.H
        RC -r FILEDLG.RC FILEDLG.RES

FILEDLG.OBJ:                    FILEDLG.C \
                                FILEDLG.H
        ICC -C+ -Kb+ -Ss+ FILEDLG.C
```

## FILEDLG.DEF

```
NAME FILEDLG WINDOWAPI
DESCRIPTION 'File dialog example
            Copyright (c) 1992-1995 by Kathleen Panov.
            All rights reserved.'
STACKSIZE 16384
```

## The Window Word

```
pchFile = (PCHAR)calloc(1,
                        CCHMAXPATH);
if (!pchFile)
{
   DisplayError("No memory could be allocated");
   return  MRFROMSHORT(TRUE);
}                              /* endif                */

WinSetWindowPtr(hwndClient,
                QWL_USER,
                pchFile);
```

In the FILEDLG example, a standard window is created with a menu. In the WM_CREATE processing, memory is allocated to hold the selected filename. The pointer to this memory is attached as a window word using *WinSetWindowPtr*. This memory is freed when the WM_DESTROY message is received.

When the user selects the "Open" selection from the menu, a WM_COMMAND message is sent. When the message is received, the user function *FindFile* is called. After this function returns, the client area is invalidated to force a repaint.

## Putting It All Together: *FindFile*

The *FindFile* function is a user-defined function where the FILEDLG structure is initialized and *WinFileDlg* is called. When the FILEDLG structure is declared, it is important to initialize the entire structure to 0.

**Gotcha!**
The FILEDLG structure is a very particular beast. Several fields in the structure are pointers or arrays of pointers. Very bad results ensue if unsued pointer fields are set to some arbitrary garbage, rather than NULL. This will occur if the FILEDLG structure is declared as an automatic structure variable and is left uninitialized. Also, note that most of these fields in the FILEDLG structure are *pointers*, not arrays. This means it is the programmer's responsibility to provide the memory. There is only one character array, *szFullFile*, of size CCHMAXFILEPATH. This is the only string field that data can be copied directly into!

## Initializing the FILEDLG Structure

```
fdFileDlg.cbSize = sizeof(FILEDLG);
fdFileDlg.fl = FDS_CENTER|FDS_PRELOAD_VOLINFO|FDS_OPEN_DIALOG;
```

The mandatory *cbSize* field is set to the size of the FILEDLG structure. The file dialog box in this example has the styles FDS_CENTER, FDS_PRELOAD_INFO, and FDS_OPEN_DIALOG. This centers the dialog, loads all the drive volume info on startup, and creates the "File Open..." dialog. These styles are OR'ed together in the *fl* field. This is also a mandatory field.

```
if (WinFileDlg(HWND_DESKTOP,
               hwndClient,
               &fdFileDlg ) != DID_OK)
```

*WinFileDlg* has three parameters. The first parameter is the parent window handle, in this case HWND_DESKTOP. The second parameter is the owner window handle, in this case *hwndClient*. The last parameter is a pointer to a FILEDLG structure.

Once the user has closed the file dialog, *szFullFile* is copied into the window word, *pchFileName*, and the function returns.

# The Font Dialog

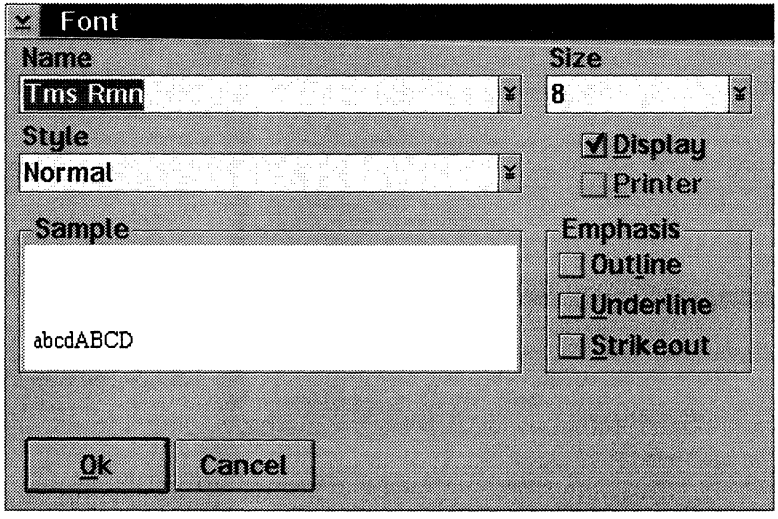



Figure 26.2 The font dialog.

The font dialog (see Figure 26.2) is created using *WinFontDlg*. This function is very similar to *WinFileDlg* in its setup. The structure FONTDLG is used to design the font dialog box layout. The structure is as follows.

```
typedef struct _FONTDLG
{
   ULONG    cbSize;
   HPS      hpsScreen;
   HPS      hpsPrinter;
   PSZ      pszTitle;
   PSZ      pszPreview;
   PSZ      pszPtSizeList;
   PFNWP    pfnDlgProc;
   PSZ      pszFamilyname;
   FIXED    fxPointSize;
   ULONG    fl;
   ULONG    flFlags;
   ULONG    flType;
   ULONG    flTypeMask;
   ULONG    flStyle;
   ULONG    flStyleMask;
```

```
    LONG    clrFore;
    LONG    clrBack;
    ULONG   ulUser;
    LONG    lReturn;
    LONG    lSRC;
    LONG    lEmHeight;
    LONG    lXHeight;
    LONG    lExternalLeading;
    HMODULE hMod;
    FATTRS  fAttrs;
    SHORT   sNominalPointSize;
    USHORT  usWeight;
    USHORT  usWidth;
    SHORT   x;
    SHORT   y;
    USHORT  usDlgId;
    USHORT  usFamilyBufLen;
    USHORT  usReserved;
} FONTDLG;
typedef FONTDLG *PFONTDLG;
```

The field *cbSize* is the size of the FONTDLG structure. The field *hpsScreen* is the presentation space for the screen. If this field is NULL, no screen fonts will be used as available fonts. The field *hpsPrinter* is the presentation space for the printer. If this field is NULL, no printer fonts will be used as available fonts. The field *pszTitle* is a pointer to a string that is the title of the file dialog box window. If this is NULL, the title of the owner window is used. The field *pszPreview* is a pointer to a string that is the text to be used in the preview window.

The field *pszPtSizeList* is a pointer to a string that is the list of font sizes that the font dialog will use as available fonts. The string is in the format "8 10 12", where each font size is separated by a space. The field *pfnDlgProc* is a pointer to a user-defined dialog procedure. The function *WinDefFontDlgProc* can be used to call the default dialog procedure from the user-defined dialog procedure. The field *pszFamilyname* is a pointer to a string that contains the font family name. On input, this field is used to determine the selected font when the font dialog is first started. When the user closes the dialog box, this field contains the family name of the font the user selected.

The field *fxPointSize* is the point size of the selected font. On input, this field is the point size of the default-selected font. When the user closes the dialog box, this field contains the point size of the font the user selected. The field *fl* is the font dialog styles flag. This field is a collection of styles OR'ed together. Table 26.2 presents the available styles.

Table 26.2 Font Dialog Styles

| Style | Description |
|---|---|
| FNTS_CENTER | The dialog is centered on the owner window. |
| FNTS_CUSTOM | Uses a custom-defined dialog template. |
| FNTS_OWNERDRAWPREVIEW | The preview box is owner-drawn. |
| FNTS_HELPBUTTON | A HELP button is included in the font dialog. |
| FNTS_APPLYBUTTON | An APPLY button is included in the font dialog. |
| FNTS_RESETBUTTON | A RESET button is included in the font dialog. |
| FNTS_MODELESS | The font dialog box is modeless. |
| FNTS_INITFROMFATTRS | The font dialog will choose the initially selected font by matching the values in the FATTRS structure. |
| FNTS_BITMAPONLY | Only bitmapped fonts will be used as available fonts. |

| | |
|---|---|
| FNTS_VECTORONLY | Only vector fonts will be used as available fonts. |
| FNTS_FIXEDWIDTHONLY | Only monospaced fonts will be used as available fonts. |
| FNTS_PROPORTIONALONLY | Only proportional fonts will be used as available fonts. |
| FNTS_NOSYNTHESIZEDFONTS | Fonts will not be synthesized. |

The field *flFlags* is a collection of font flags OR'ed together. The flags listed in Table 26.3 are available.

Table 26.3 Available Font Flags

| Flag | Description |
|---|---|
| FNTF_NOVIEWPRINTERFONTS | An input flag. If specified, and both *hpsScreen* and *hpsPrinter* are used, the printer fonts will not be included in the list of available fonts. |
| FNTF_NOVIEWSCREENFONTS | An input flag. If specified, and both *hpsScreen* and *hpsPrinter* are used, the screen fonts will not be included in the list of available fonts. |
| FNTF_PRINTERFONTSELECTED | An output flag. It indicates that the user selected a printer font. |
| FNTF_SCREENFONTSELECTED | An output flag. It indicates that the user selected a screen font. |

The field *flType* contains the additional characteristics of the font the user selected. Table 26.4 specifies the types available.

Table 26.4 Font Characteristics

| Type | Description |
|---|---|
| FTYPE_ITALIC | The font selected was italic. |
| FTYPE_ITALIC_DONT_CARE | The font selected was not italic. |
| FTYPE_OBLIQUE | The font selected was oblique. |
| FTYPE_OBLIQUE_DONT_CARE | The font selected was not oblique. |
| FTYPE_ROUNDED | The font selected was rounded. |
| FTYPE_ROUNDED_DONT_CARE | The font selected was not rounded. |

The field *flTypeMask* is a mask of which font types to use.

The field *flStyle* is the font styles the user selected. Table 26.5 lists the available styles.

Table 26.5 Font Style of Selected Font

| Style | Description |
|---|---|
| FATTR_SEL_ITALIC | The font selected was italic. |
| FATTR_SEL_UNDERSCORE | The font selected was underscore. |
| FATTR_SEL_BOLD | The font selected was bold. |
| FATTR_SEL_STRIKEOUT | The font selected was strikeout. |
| FATTR_SEL_OUTLINE | The font selected was outline. |

The field *flStyleMask* is a mask of which font styles to use. The field *clrFore* is the font foreground color index. The field *clrBack* is the font background color index.

The field *ulUser* is 4 bytes of user-defined storage space. The field *lReturn* is the ID of the push-button the user pushed to close the dialog; DID_OK, DID_CANCEL, or 0 if an error occurred.

The field *lSRC* is the system return code if the font dialog fails. Table 26.6 presents the possible values.

Table 26.6 Values of *lSRC*

| Value | Description |
|---|---|
| FNTS_SUCCESSFUL | Font dialog was successful |
| FNTS_ERR_INVALID_DIALOG | Invalid dialog error |
| FNTS_ERR_ALLOC_SHARED_MEM | Error allocating shared memory |
| FNTS_ERR_INVALID_PARM | Invalid parameter |
| FNTS_ERR_OUT_OF_MEMORY | Out-of-memory error |
| FNTS_ERR_INVALID_VERSION | Invalid version error |
| FNTS_ERR_DIALOG_LOAD_ERROR | Error loading dialog |

The field *lEmHeight* is the point size of the font converted into world coordinates. This field multiplied by 1.2 is often a good gauge for the vertical spacing between rows of text. The field *lXHeight* is the height in pixels of the character x. The field *lExternalLeading* is the recommended vertical spacing between rows of text. This value is the maximum spacing, not the actual spacing to use.

The field *hMod* is the module handle to use for loading a custom font dialog. This field is used only if FNTS_CUSTOM is set in the *fl* field. If FNTS_CUSTOM is set, and this field is NULL, the resource is drawn from the executable. The field *fAttrs* is a FATTRS structure for the selected font. The field *sNominalPointSize* is the font point size. This field is meaningful for bitmap fonts only.

The field *usWeight* is the weight of the font. Table 26.7 lists possible values.

Table 26.7 Values of *usWeight*

| Weight | Description |
|---|---|
| FWEIGHT_DONT_CARE | Any font weight is applicable. |
| FWEIGHT_ULTRA_LIGHT | The font is ultra-light. |
| FWEIGHT_EXTRA_LIGHT | The font is extra light. |
| FWEIGHT_LIGHT | The font is light. |
| FWEIGHT_SEMI_LIGHT | The font is semilight. |
| FWEIGHT_NORMAL | The font is normal weight. |
| FWEIGHT_SEMI_BOLD | The font is semibold. |
| FWEIGHT_BOLD | The font is bold. |
| FWEIGHT_EXTRA_BOLD | The font is extrabold. |
| FWEIGHT_ULTRA_BOLD | The font is ultra-bold. |

The field *usWidth* is the width class of the font the user selects. Table 26.8 lists possible values.

Table 26.8 Values of *usWidth*

| Width | Description |
|---|---|
| FWIDTH_DONT_CARE | Any font width is applicable. |
| FWIDTH_ULTRA_CONDENSED | The selected font has an aspect ratio 50 percent of the normal ratio. |
| FWIDTH_EXTRA_CONDENSED | The selected font has an aspect ratio 62.5 percent of the normal ratio. |
| FWIDTH_CONDENSED | The selected font has an aspect ratio 75 percent of the normal ratio. |

| Width | Description |
|---|---|
| FWIDTH_SEMI_CONDENSED | The selected font has an aspect ratio 87.5 percent of the normal ratio. |
| FWIDTH_NORMAL | The selected font has an aspect ratio 100 percent of the normal ratio. |
| FWIDTH_SEMI_EXPANDED | The selected font has an aspect ratio 112.25 percent of the normal ratio. |
| FWIDTH_EXPANDED | The selected font has an aspect ratio 125 percent of the normal ratio. |
| FWIDTH_EXTRA_EXPANDED | The selected font has an aspect ratio 150 percent of the normal ratio. |
| FWIDTH_ULTRA_EXPANDED | The selected font has an aspect ratio 200 percent of the normal ratio. |

The field *x* is the x coordinate for the font dialog box. This field is unused if the *fl* flag has FNTS_CENTER set. The field *y* is the y coordinate for the font dialog box. This field is unused if the *fl* flag has FNTS_CENTER set. The field *usDlgID* is the resource ID of the dialog box to be used if the FNTS_CUSTOM flag in the *fl* field is set. The field *usFamilyBufLen* is the length of the *pszFamilyname* buffer. This field is mandatory.

**Gotcha!**
Several fields are *mandatory* in the font dialog control: *cbSize*, *hpsScreen* or *hpsPrinter*, *pszFamilyname*, *usFamilyBufLen*, and *fl*. Also, all of the string fields in the FONTDLG structure are pointers. It is the programmer's responsibility to provide the space for these fields.

# An Example Program: FONTDLG

**FONTDLG.C**

```
#define  INCL_WIN
#define  INCL_STDDLG
#define  INCL_GPI
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fontdlg.h"

typedef struct
{
   FONTDLG          fdFontDlg;
   USHORT           bInit;
} MYFONTINFO,*PMYFONTINFO;

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);
```

```
VOID SetFont(HWND hwndClient,PMYFONTINFO pmfiFont);
BOOL InitFont(HWND hwndClient,PFONTDLG pfdFontDlg);
VOID DisplayError(CHAR *pszText);

#define  CLS_CLIENT    "MyClass"

INT main(VOID)

{
   HMQ             hmqQueue;
   HAB             habAnchor;
   ULONG           ulFlags;
   HWND            hwndClient;
   BOOL            bLoop;
   QMSG            qmMsg;
   LONG            lWidth,lHeight;
   HWND            hwndFrame = NULLHANDLE;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    CS_SIZEREDRAW|CS_SYNCPAINT,
                    sizeof(PVOID));

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MENU|
      FCF_MINMAX;

   /*************************************************************/
   /* create frame and client window                            */
   /*************************************************************/

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Font Dialog Example",
                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  &hwndClient);

   /*************************************************************/
   /* get screen height and width                               */
   /*************************************************************/

   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /*************************************************************/
   /* set size and position of frame window                     */
   /*************************************************************/

   if (hwndFrame)
   {
      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      10,
                      10,
                      lWidth/10*8,
                      lHeight/10*8,
```

```
                        SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile                */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                   */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PMYFONTINFO      pmfiFont;

   switch (ulMsg)
   {

      case  WM_CREATE :
         {

            /*************************************************/
            /* allocate the window word                      */
            /*************************************************/

            pmfiFont = calloc(1,
                              sizeof(MYFONTINFO));
            if (!pmfiFont)
            {
               DisplayError("No memory could be allocated");
               return  MRFROMSHORT(TRUE);
            }                            /* endif                   */

            WinSetWindowPtr(hwndClient,
                            QWL_USER,
                            pmfiFont);

            /*************************************************/
            /* set font to 8 pt Times Roman                  */
            /*************************************************/

            WinSetPresParam(hwndClient,
                            PP_FONTNAMESIZE,
                            10,
                            "8.Tms Rmn");

            /*************************************************/
            /* indicates first time dialog is brought up     */
            /*************************************************/

            pmfiFont->bInit = FALSE;
            break;
         }
      case  WM_DESTROY :
```

```
    {

       /*************************************************/
       /* clean up                                      */
       /*************************************************/

       pmfiFont = WinQueryWindowPtr(hwndClient,
                                    QWL_USER);
       if (pmfiFont)
       {
          free(pmfiFont);
       }                            /* endif             */
       break;
    }
 case  WM_PAINT :
    {
       HPS              hpsPaint;
       ULONG            ulReturn;
       RECTL            rclPaint;
       CHAR             achFontName[200],achMsg[256];

       hpsPaint = WinBeginPaint(hwndClient,
                                NULLHANDLE,
                                &rclPaint);

       /*************************************************/
       /* get the current font                          */
       /*************************************************/

       ulReturn = WinQueryPresParam(hwndClient,
                                    PP_FONTNAMESIZE,
                                    0,
                                    NULL,
                                    256,
                                    achFontName,
                                    0);
       if (ulReturn)
       {
          sprintf(achMsg,
                  "The font selected is \"%s\"",
                  achFontName);
       }
       else
       {
          strcpy(achMsg,
                 "No font selected");
       }                            /* endif             */

       /*************************************************/
       /* clear out the window                          */
       /*************************************************/

       WinFillRect(hpsPaint,
                   &rclPaint,
                   SYSCLR_WINDOW);
       WinQueryWindowRect(hwndClient,
                          &rclPaint);

       /*************************************************/
       /* write out current font on client area         */
       /*************************************************/

       WinDrawText(hpsPaint,
                   -1,
                   achMsg,
                   &rclPaint,
                   0,
```

```
                    0,
                    DT_VCENTER|DT_CENTER|DT_TEXTATTRS);

         WinEndPaint(hpsPaint);
      }
      break;

   case  WM_COMMAND :

      switch (SHORT1FROMMP(mpParm1))
      {

         case  IDM_FONT :
            {

               /*******************************************/
               /* get window word                         */
               /*******************************************/

               pmfiFont = WinQueryWindowPtr(hwndClient,
                                            QWL_USER);
               if (!pmfiFont)
               {
                  DisplayError(
                "Unable to retrieve font information structure")
                     ;
                  break;
               }

               /*******************************************/
               /* do font dialog                          */
               /*******************************************/

               SetFont(hwndClient,
                       pmfiFont);

               /*******************************************/
               /* repaint window with new font            */
               /*******************************************/

               WinInvalidateRect(hwndClient,
                                 NULL,
                                 TRUE);
               break;
            }
         case  IDM_EXIT :
            {
               WinPostMsg(hwndClient,
                          WM_CLOSE,
                          0,
                          0);
               break;
            }
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,

                                     mpParm2);
      }                             /* endswitch            */
      break;

   default  :
      return  WinDefWindowProc(hwndClient,
                               ulMsg,
                               mpParm1,

                               mpParm2);
}                                   /* endswitch            */
```

```
   return  MRFROMSHORT(FALSE);
}

VOID SetFont(HWND hwndClient,PMYFONTINFO pmfiFont)
{
   FATTRS           faAttrs;
   FIXED            fxSzFont;
   CHAR             achFamily[256];
   CHAR             achFont[256];

   /*********************************************************/
   /* save some values                                      */
   /*********************************************************/

   faAttrs = pmfiFont->fdFontDlg.fAttrs;
   fxSzFont = pmfiFont->fdFontDlg.fxPointSize;

   /*********************************************************/
   /* clear out the structures                              */
   /*********************************************************/

   memset(&pmfiFont->fdFontDlg,
          0,
          sizeof(FONTDLG));
   memset(achFont,
          0,
          256);

   if (pmfiFont->bInit)
   {

      /*********************************************************/
      /* fontdialog structure has already been initialized     */
      /*********************************************************/

      pmfiFont->fdFontDlg.fAttrs = faAttrs;
      pmfiFont->fdFontDlg.fxPointSize = fxSzFont;
   }                                    /* endif              */
   else
   {

      /*********************************************************/
      /* first time through                                    */
      /*********************************************************/

      InitFont(hwndClient,
               &(pmfiFont->fdFontDlg));
      pmfiFont->bInit = TRUE;
   }

   /*********************************************************/
   /* joint initialization                                  */
   /*********************************************************/

   pmfiFont->fdFontDlg.hpsScreen = WinGetPS(hwndClient);
   pmfiFont->fdFontDlg.cbSize = sizeof(FONTDLG);

   pmfiFont->fdFontDlg.pszFamilyname = achFamily;
   pmfiFont->fdFontDlg.usFamilyBufLen = sizeof(achFamily);
   pmfiFont->fdFontDlg.fl = FNTS_CENTER|FNTS_INITFROMFATTRS;
   pmfiFont->fdFontDlg.clrFore = CLR_NEUTRAL;
   pmfiFont->fdFontDlg.clrBack = SYSCLR_WINDOW;
```

```
   /*************************************************************/
   /* return from font dialog is TRUE or FALSE for non-modeless*/
   /* font dialog, window handle if FNTS_MODELESS is set ours  */
   /* is non-modeless, so we check for TRUE or FALSE           */
   /*************************************************************/

   if (!WinFontDlg(HWND_DESKTOP,
                   hwndClient,
                   &pmfiFont->fdFontDlg))
   {

      DisplayError("WinFontDlg failed");
      return ;
   }                                  /* endif                  */

   /*************************************************************/
   /* clean up                                                  */
   /*************************************************************/

   WinReleasePS(pmfiFont->fdFontDlg.hpsScreen);

   sprintf(achFont,
           "%d.%s",
           FIXEDINT(pmfiFont->fdFontDlg.fxPointSize),
           pmfiFont->fdFontDlg.fAttrs.szFacename);

   /*************************************************************/
   /* check for various attributes                              */
   /*************************************************************/

   if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_ITALIC)
   {
      strcat(achFont,
             ".Italic");
   }                                  /* endif                  */
   if (pmfiFont->fdFontDlg.fAttrs.fsSelection
      &FATTR_SEL_UNDERSCORE)
   {
      strcat(achFont,
             ".Underscore");
   }                                  /* endif                  */
   if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_STRIKEOUT
      )
   {
      strcat(achFont,
             ".Strikeout");
   }                                  /* endif                  */
   if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_BOLD)
   {
      strcat(achFont,
             ".Bold");
   }                                  /* endif                  */
   if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_OUTLINE)
   {
      strcat(achFont,
             ".Outline");
   }                                  /* endif                  */

   /*************************************************************/
   /* set the new font                                          */
   /*************************************************************/

   WinSetPresParam(hwndClient,
                   PP_FONTNAMESIZE,
                   strlen(achFont)+1,
                   achFont);

   return ;
}
```

```
/*************************************************************/
/*                                                           */
/* function that analyzes the current font and initializes   */
/* the font dialog to use the current font as the default font */
/*                                                           */
/*************************************************************/

BOOL InitFont(HWND hwndClient,PFONTDLG pfdFontDlg)
{
   FONTMETRICS      fm;
   HPS              hPS;
   HDC              hDC;
   SIZEF            sizef;
   LONG             lxFontResolution;

   /*********************************************************/
   /* Get the presentation space to query fonts from        */
   /*********************************************************/

   hPS = WinGetPS(hwndClient);

   if (GpiQueryFontMetrics(hPS,
                           (LONG)sizeof(FONTMETRICS),
                           &fm))
   {

      /******************************************************/
      /* Initialize the font dialog structure and the fattrs */
      /* fields                                             */
      /******************************************************/

      memset(&pfdFontDlg->fAttrs,
             0,
             sizeof(FATTRS));
      pfdFontDlg->fAttrs.usRecordLength = sizeof(FATTRS);

      /******************************************************/
      /* Initialize the font attributes                     */
      /******************************************************/

      if (fm.fsSelection&FM_SEL_ITALIC)
         pfdFontDlg->fAttrs.fsSelection |= FATTR_SEL_ITALIC;
      if (fm.fsSelection&FM_SEL_UNDERSCORE)
         pfdFontDlg->fAttrs.fsSelection |= FATTR_SEL_UNDERSCORE;
      if (fm.fsSelection&FM_SEL_OUTLINE)
         pfdFontDlg->fAttrs.fsSelection |= FATTR_SEL_OUTLINE;
      if (fm.fsSelection&FM_SEL_STRIKEOUT)
         pfdFontDlg->fAttrs.fsSelection |= FATTR_SEL_STRIKEOUT;
      if (fm.fsSelection&FM_SEL_BOLD)
         pfdFontDlg->fAttrs.fsSelection |= FATTR_SEL_BOLD;

      /******************************************************/
      /* Initialize the fattrs match-font and registry id   */
      /******************************************************/

      pfdFontDlg->fAttrs.lMatch = fm.lMatch;
      pfdFontDlg->fAttrs.idRegistry = fm.idRegistry;

      /******************************************************/
      /* Initialize the fattrs code page                    */
      /******************************************************/

      pfdFontDlg->fAttrs.usCodePage = GpiQueryCp(hPS);
```

```
/*********************************************************/
/* Initialize the fattrs max baseline ext and avg char   */
/* width                                                 */
/*********************************************************/

if (fm.fsDefn&FM_DEFN_OUTLINE)
{
   pfdFontDlg->fAttrs.lAveCharWidth = 0;
   pfdFontDlg->fAttrs.lMaxBaselineExt = 0;
}
else
{
   pfdFontDlg->fAttrs.lMaxBaselineExt = fm.lMaxBaselineExt;
   pfdFontDlg->fAttrs.lAveCharWidth = fm.lAveCharWidth;
}

/*********************************************************/
/* Initialize the fattrs type indicator field            */
/*********************************************************/

if (fm.fsType&FM_TYPE_KERNING)
   pfdFontDlg->fAttrs.fsType |= FATTR_TYPE_KERNING;
if (fm.fsType&FM_TYPE_DBCS)
   pfdFontDlg->fAttrs.fsType |= FATTR_TYPE_DBCS;
if (fm.fsType&FM_TYPE_MBCS)
   pfdFontDlg->fAttrs.fsType |= FATTR_TYPE_MBCS;

/*********************************************************/
/* Initialize the fattrs typeface name field             */
/*********************************************************/

strcpy(pfdFontDlg->fAttrs.szFacename,
       fm.szFacename);

/*********************************************************/
/* Initialize the fontdlg style flags                    */
/*********************************************************/

pfdFontDlg->fl = FNTS_CENTER|FNTS_HELPBUTTON|
   FNTS_INITFROMFATTRS;

/*********************************************************/
/* Initialize the fontdlg weight and width fields        */
/*********************************************************/

pfdFontDlg->usWeight = fm.usWeightClass;
pfdFontDlg->usWidth = fm.usWidthClass;

/*********************************************************/
/* Obtain character box size Note: This is critical for  */
/* outline font calculations.                            */
/*********************************************************/

GpiQueryCharBox(hPS,
                &sizef);

/*********************************************************/
/* Query the underlying device's capabilities for the    */
/* horizontal device resolution                          */
/*********************************************************/

hDC = GpiQueryDevice(hPS);
DevQueryCaps(hDC,
             CAPS_HORIZONTAL_FONT_RES,
             1L,
             &lxFontResolution);
```

```
      /*********************************************************/
      /* Initialize the fattrs font-use indicators, and the    */
      /* fontdlg point size field.                             */
      /*********************************************************/

      if (fm.fsDefn&FM_DEFN_OUTLINE)
      {
         pfdFontDlg->fAttrs.fsFontUse = FATTR_FONTUSE_OUTLINE;
         pfdFontDlg->fxPointSize = (FIXED)((sizef.cx *72)/
            lxFontResolution);

      }                                /* if outline font      */
      else
      {
         pfdFontDlg->fAttrs.fsFontUse = FATTR_FONTUSE_NOMIX;
         pfdFontDlg->fxPointSize = (FIXED)(fm.sNominalPointSize/
            100);

      }                                /* not outline font     */

      /*********************************************************/
      /* Release the PS from above and return                  */
      /*********************************************************/

      WinReleasePS(hPS);
      return (TRUE);
   }

   /*********************************************************/
   /* error in GpiQueryFontMetrics                          */
   /*********************************************************/

   DisplayError("GpiQueryFontMetrics failed");

   /*********************************************************/
   /* Release the PS from above and return                  */
   /*********************************************************/

   WinReleasePS(hPS);
   return (FALSE);
}

VOID DisplayError(CHAR *pszText)
{

   /*********************************************************/
   /* small function to display error string                */
   /*********************************************************/

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 HWND_DESKTOP,
                 pszText,
                 "Error!",
                 0,
                 MB_OK|MB_ERROR);

   return ;

}
```

**FONTDLG.RC**

```
#include <os2.h>
#include "fontdlg.h"

MENU RES_CLIENT
{
   SUBMENU "~Fonts", IDM_SUB1
   {
      MENUITEM "~Change font...", IDM_FONT
      MENUITEM "E~xit", IDM_EXIT
   }
}
```

**FONTDLG.H**

```
#define RES_CLIENT              256
#define IDM_SUB1                512
#define IDM_FONT                513
#define IDM_EXIT                514
```

**FONTDLG.MAK**

```
FONTDLG.EXE:                    FONTDLG.OBJ \
                                FONTDLG.RES
        LINK386 @<<
FONTDLG
FONTDLG
FONTDLG
OS2386
FONTDLG
<<
        RC FONTDLG.RES FONTDLG.EXE

FONTDLG.RES:                    FONTDLG.RC \
                                FONTDLG.H
        RC -r FONTDLG.RC FONTDLG.RES

FONTDLG.OBJ:                    FONTDLG.C \
                                FONTDLG.H
        ICC -C+ -Kb+ -Ss+ FONTDLG.C
```

**FONTDLG.DEF**

```
NAME FONTDLG WINDOWAPI
DESCRIPTION 'Font dialog example
            Copyright (c) 1992-1995 by Kathleen Panov.
            All rights reserved.'
STACKSIZE 32768
```

# Customizing the Font Dialog

The font dialog does not use the current font of a window as the default-selected font. There are two ways to make the default font the current font of a selected window:

- Query the current font characteristics, place these in the appropriate spots in the FONTDLG structure, and use the FNTS_INITFROMATTRS flag. This method must be used if the current font of a selected window will be used and this is the first time *WinFontDlg* has been called.

- Store the FONTDLG structure that was the output from *WinFontDlg*, and reuse it the next time *WinFontDlg* is called.

The first option is a real pain to implement but is used the first time the dialog is called. The function *InitFont* uses this method. After initialization, we use the second method.

We create a special structure, MYFONTINFO, to hold the FONTDLG structure in memory.

```
typedef struct {
   FONTDLG fdFontDlg ;
   USHORT  bInit ;
} MYFONTINFO, *PMYFONTINFO ;
```

This structure contains a FONTDLG structure and a flag to indicate whether the structure has been initialized or not.

In the WM_CREATE processing, space is allocated for the MYFONTINFO structure. This pointer is stored in a window word of the client window. This memory is freed in the WM_DESTROY message processing.

# Querying the Current Font

```
ulReturn = WinQueryPresParam(hwndClient,
                             PP_FONTNAMESIZE,
                             0,
                             NULL,
                             256,
                             achFontName,
                             0);
```

When a WM_PAINT message is received, *WinQueryPresParam* is used to determine the current font. The first parameter is the window to query. The next parameter is the attribute ID. PP_FONTNAMESIZE will retrieve the font name and point size. The third parameter is used to query a second type of presentation parameter. The next parameter is used to determine which presentation parameter, the first or second, was found first. The fifth parameter is the length of the results buffer. The buffer, *achFontName*, is the next parameter. The last parameter, the query options, is unused in this example. *WinQueryPresParam* returns the number of characters placed in the *achFontName* buffer. The font name is copied into a character array, and *WinDrawText* outputs the result onto the client window.

# Initializing the Font Dialog Structure with the Current Font

The *InitFont* function converts a FONTMETRICS structure, returned from *GpiQueryFontMetrics*, into a FATTRS structure that the font dialog can understand. The initial font attributes from the FONTMETRICS structure are OR'ed with the *fsSelection* field in the FATTRS structure. These attributes include italic, bold, outline, underscore, and strikeout.

*lMatch* is a unique identifier for a font. All fonts available to a presentation space are given a match ID. These vary from device to device and from system to system; however, within a single presentation space, they are consistent. The *idRegistry* is the IBM registered number for certain fonts. The current code page is also queried and set in the FATTRS structure.

```
GpiQueryCharBox(hPS,
                &sizef);
```

```
hDC = GpiQueryDevice(hPS);
DevQueryCaps(hDC,
             CAPS_HORIZONTAL_FONT_RES,
             1L,
             &lxFontResolution);

if (fm.fsDefn&FM_DEFN_OUTLINE)
{
   pfdFontDlg->fAttrs.fsFontUse = FATTR_FONTUSE_OUTLINE;
   pfdFontDlg->fxPointSize = (FIXED)((sizef.cx *72)/
      lxFontResolution);

}                                  /* if outline font      */
```

The setup for an outline font is a little more complicated than that of a nonoutline font. The *lMaxBaselineExt* is correct for the bitmap fonts, but for outline fonts this value is the actual distance from the highest pel to the lowest pel, with no leading indicator included. Instead of trying to determine this value, we find the exact point size and set *lMaxBaselineExt* and *lAveCharWidth* to 0. This is done by using the following conversion.

point size = cx pixels/inch * 72 points/inch / resolution pixels/inch

```
else
{
   pfdFontDlg->fAttrs.fsFontUse = FATTR_FONTUSE_NOMIX;
   pfdFontDlg->fxPointSize = (FIXED)(fm.sNominalPointSize/
      100);
}                                  /* not outline font     */
```

For a nonoutline font, the point size is simply the nominal point size divided by 100.

# Bringing Up the Font Dialog

*SetFont* is a user-defined function to initialize the font dialog, bring it up, and change the client window font to the newly selected font.

The MYFONTINFO structure that contains the FONTDLG structure is retrieved from the window word, and is passed to *SetFont.*

```
faAttrs = pmfiFont->fdFontDlg.fAttrs;
fxSzFont = pmfiFont->fdFontDlg.fxPointSize;

memset(&pmfiFont->fdFontDlg,
       0,
       sizeof(FONTDLG));
memset(achFont,
       0,
       256);
```

The first thing *SetFont* does is to save the FATTRS structure and also the *fxPointSize* values for use later. The FONTDLG structure and font name string are then cleared to 0. If this is the first time through this function, *InitFont* is called, and the initialization flag is set to TRUE. If this is not the first time *SetFont* has been called, we assume the FATTRS structure in memory is valid and set the font dialog structure FATTRS equal to the structure in memory.

```
pmfiFont->fdFontDlg.hpsScreen = WinGetPS(hwndClient);
pmfiFont->fdFontDlg.cbSize = sizeof(FONTDLG);
```

```
pmfiFont->fdFontDlg.pszFamilyname = achFamily;
pmfiFont->fdFontDlg.usFamilyBufLen = sizeof(achFamily);
pmfiFont->fdFontDlg.fl = FNTS_CENTER|FNTS_INITFROMFATTRS;
pmfiFont->fdFontDlg.clrFore = CLR_NEUTRAL;
pmfiFont->fdFontDlg.clrBack = SYSCLR_WINDOW;
```

Several elements of the FONTDLG structure are initialized. The screen presentation space is queried, and the size of the FONTDLG structure is set. The *pszFamilyname* member is set equal to the *achFamily* buffer. The size of this buffer is set in *usFamilyBufLen*. The flags used for this font dialog are FNTS_CENTER (center the dialog) and FNTS_INITFROMFATTRS (use the FATTRS structure to set the initial default font selection). The last elements initialized are the foreground and background colors for the sample preview box.

```
HWND WinFontDlg( HWND hwndParent, HWND hwndOwner, PFONTDLG
                 pFontDialog)
```

*WinFontDlg* has three parameters. The first is the parent window, HWND_DESKTOP. The next is the owner window, and the last is a *pointer* to the FONTDLG structure.

```
sprintf(achFont,
          "%d.%s",
          FIXEDINT(pmfiFont->fdFontDlg.fxPointSize),
          pmfiFont->fdFontDlg.fAttrs.szFacename);
```

The *fxPointSize* variable in the FONTDLG structure is a FIXED data type. This is a long integer used to represent a fractional integer. To obtain the actual point size, the macro FIXEDINT is used to extract the integer position of the fixed type. This value is the actual font point size.

The *szFacename* array in the *fAttrs* structure is where we get the font style from. This array contains a bit more descriptive font style than the *pszFamilyname* pointer. (We had mixed results using the *pszFamilyname* variable but got 100 percent accuracy using *szFacename.*)

```
if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_ITALIC)
{
   strcat(achFont,
          ".Italic");
}                                    /* endif                */
if (pmfiFont->fdFontDlg.fAttrs.fsSelection
   &FATTR_SEL_UNDERSCORE)
{
   strcat(achFont,
          ".Underscore");
}                                    /* endif                */
if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_STRIKEOUT
   )
{
strcat(achFont,
       ".Strikeout");
}                                    /* endif                */
if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_BOLD)
{
   strcat(achFont,
          ".Bold");
}                                    /* endif                */
if (pmfiFont->fdFontDlg.fAttrs.fsSelection&FATTR_SEL_OUTLINE)
{
   strcat(achFont,
          ".Outline");
}                                    /* endif
```

The *fsSelection* flag contains more information about the font type. A comparison is made, and if the result is TRUE, the string is concatenated with a ".Descriptor" string. The presentation parameter string can take multiple instances of these descriptors, for example, "10.Tms Rmn Bold.Italic.Underline".

```
WinSetPresParam(hwndClient,
                PP_FONTNAMESIZE,
                strlen(achFont)+1,
                achFont);
```

*WinSetPresParam* will change the font of the client window to the user-selected font. The first parameter is the window to apply the changes to, *hwndClient.* The next parameter is the presentation attribute, PP_FONTNAMESIZE, to change. The third parameter is the size of the presentation parameter. The last parameter is a pointer to the variable itself. A small note here: If the presentation parameter is a color, this value is the *address* of a LONG or an RGB structure.

# Chapter 27

# Subclassing Windows

Subclassing windows is the ability to intercept and process messages sent to the window procedure of an established window class. A message is normally sent to a window procedure where it is either processed and returned to the calling window, processed and returned to *WinDefWindowProc*, or passed directly to *WinDefWindowProc*. A subclassed procedure is placed in the calling chain directly above the window procedure. This also allows the subclassed procedure to sort through the messages and process only the ones it wishes to modify.

The flowchart shown in Figure 27.1 illustrates the normal calling chain for window messages.

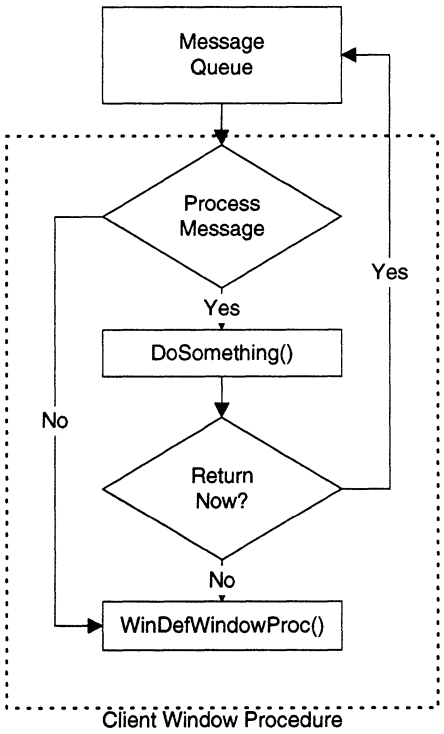



Figure 27.1 Diagram of normal window procedure.

The flowchart shown in Figure 27.2 illustrates what happens to the calling chain when a window is subclassed.

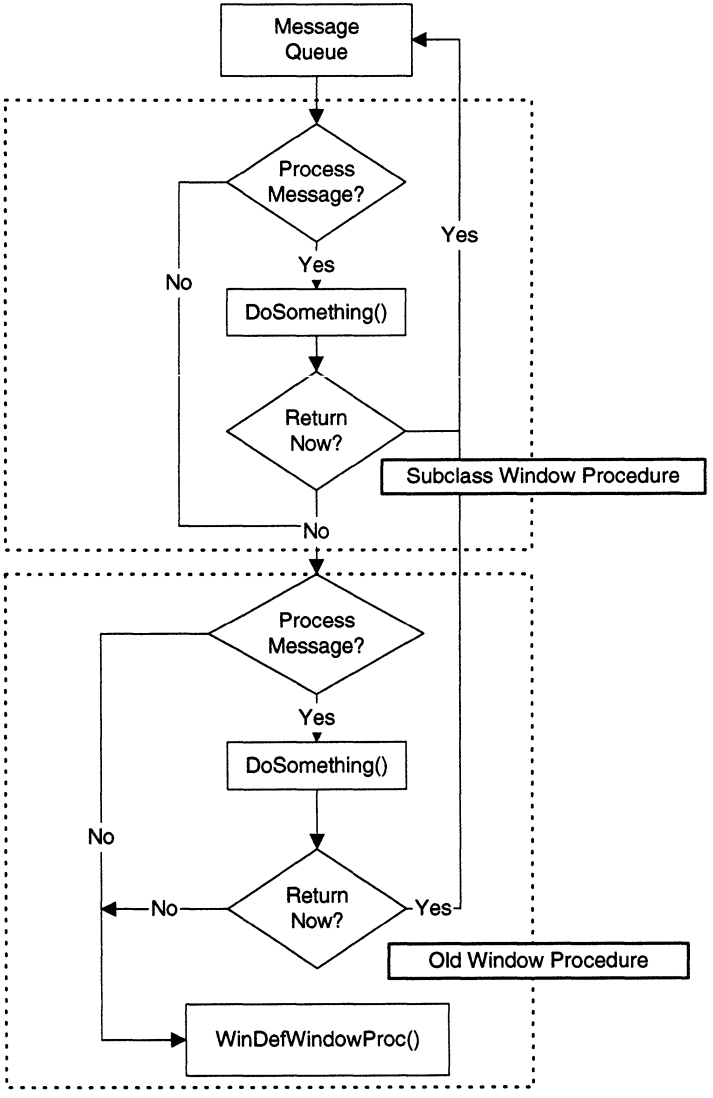



Figure 27.2 Subclassed window procedure calling chain.

Subclassing is a very easy way to modify the behavior of a window class. The subclassed procedure should be kept small to keep the window's behavior responsive to the user. A long and complex subclass procedure will cause a decrease in performance. (Three functions are being called for every message generated from the window.) The following code will define the subclassed procedure.

```
MRESULT EXPENTRY pfnwpOldProc;
pfnwpOldProc = WinSubclassWindow( hwndWindowToSubclass,
                                  pfnwpNewProc);
```

The function returns the previous window procedure as *pfnwpOldProc*. This function provides the subclassed procedure a way to call the previous window procedure.

Now let's put subclassing to use. Suppose we want an entry field that handles only numbers, say, for Zip codes. There's not an existing numerics-only entry field, so let's create one.

**SUBCLASS.C**

```
#define  INCL_WIN
#define  INCL_GPILCIDS

#include <os2.h>
#include <string.h>
#include <ctype.h>

#define  CLS_CLIENT     "MyClass"
#define  IDE_ENTRYFIELD 256
#define  STR_TEXT       "Zip code:"
#define  UM_CREATEDONE WM_USER+1

MRESULT EXPENTRY newEntryWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                                 mpParm1,MPARAM mpParm2);

MRESULT EXPENTRY ClientWndProc(HWND hwndWnd,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

INT main(VOID)
{

   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;
   LONG             lWidth,lHeight;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    ClientWndProc,
                    0,
                    0);

   ulFlags = FCF_TITLEBAR|FCF_SYSMENU|FCF_SIZEBORDER|FCF_MINMAX|
      FCF_TASKLIST;

   /************************************************************/
   /* create frame window                                      */
   /************************************************************/
```

```
   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  0,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Subclass Example",
                                  0,
                                  NULLHANDLE,
                                  0,
                                  NULL);

   /*********************************************************/
   /* get screen height and width                           */
   /*********************************************************/

   lWidth = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSCREEN);

   lHeight = WinQuerySysValue(HWND_DESKTOP,
                              SV_CYSCREEN);

   /*********************************************************/
   /* if failed, display error, and set to default value    */
   /*********************************************************/

   if (!lWidth || !lHeight)
   {
      lWidth = 640;
      lHeight = 480;
   }

   if (hwndFrame != NULLHANDLE)
   {

      /*******************************************************/
      /* set window position                                 */
      /*******************************************************/

      WinSetWindowPos(hwndFrame,
                      NULLHANDLE,
                      lWidth/8,
                      lHeight/8,
                      lWidth/8*6,
                      lHeight/8*6,
                      SWP_SIZE|SWP_MOVE|SWP_ACTIVATE|SWP_SHOW);

      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile          */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif             */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

```
MRESULT EXPENTRY ClientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {

      case  WM_CREATE :
         WinPostMsg(hwndClient,
                    UM_CREATEDONE,
                    MPVOID,
                    MPVOID);
         break;

      case  UM_CREATEDONE :

         {
            HPS               hpsChar;
            FONTMETRICS       fmMetrics;
            ENTRYFDATA        edEntry;
            HWND              hwndEntry;
            PFNWP             pfnOldEntryProc;
            RECTL             rclClient;
            LONG              lHeight,lWidth;

            hpsChar = WinGetPS(hwndClient);
            GpiQueryFontMetrics(hpsChar,
                                sizeof(fmMetrics),
                                &fmMetrics);
            WinReleasePS(hpsChar);

            /************************************************/
            /* set up entryfield data                       */
            /************************************************/

            edEntry.cb = sizeof(edEntry);
            edEntry.cchEditLimit = 9;
            edEntry.ichMinSel = 0;
            edEntry.ichMaxSel = 0;

            WinQueryWindowRect(hwndClient,
                               &rclClient);
            lHeight = rclClient.yTop-rclClient.yBottom;
            lWidth = rclClient.xRight-rclClient.xLeft;

            /************************************************/
            /* create entryfield window                     */
            /************************************************/

            hwndEntry = WinCreateWindow(hwndClient,
                                        WC_ENTRYFIELD,
                                        "",
                                        WS_VISIBLE|ES_MARGIN|
                                           ES_AUTOSIZE,
                                        lWidth/4,
                                        lHeight/4,
                                        fmMetrics.lAveCharWidth
                                        *15,
                                       fmMetrics.lMaxBaselineExt,
                                        hwndClient,
                                        HWND_TOP,
                                        IDE_ENTRYFIELD,
                                        &edEntry,
                                        NULL);
```

```
      pfnOldEntryProc = WinSubclassWindow(hwndEntry,
                                          newEntryWndProc);

            WinSetWindowPtr(hwndEntry,
                            QWL_USER,
                            (PVOID)pfnOldEntryProc);

            WinSetFocus(HWND_DESKTOP,
                        hwndEntry);
         }
         break;

      case  WM_DESTROY :
         WinDestroyWindow(WinWindowFromID(hwndClient,
                                          IDE_ENTRYFIELD));
         break;

      case  WM_PAINT :
         {
            HPS               hpsPaint;
            SWP               swpEntry;
            POINTL            ptlText;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     NULL);

            GpiErase(hpsPaint);
            WinQueryWindowPos(WinWindowFromID(hwndClient,
                                              IDE_ENTRYFIELD),
                              &swpEntry);

            /*************************************************/
            /* position label on top of entryfield           */
            /*************************************************/

            ptlText.x = swpEntry.x;
            ptlText.y = swpEntry.y+swpEntry.cy+10;

            GpiCharStringAt(hpsPaint,
                            &ptlText,
                            strlen(STR_TEXT),
                            STR_TEXT);

            WinEndPaint(hpsPaint);
         }
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,

                                  mpParm2);
   }                                   /* endswitch             */
   return  MRFROMSHORT(FALSE);
}

MRESULT EXPENTRY newEntryWndProc(HWND hwndEntry,ULONG ulMsg,
                                 MPARAM mpParm1,MPARAM mpParm2)
{
   PFNWP             pfnOldEntryProc;

   pfnOldEntryProc = (PFNWP)WinQueryWindowPtr(hwndEntry,
                                              QWL_USER);

   switch (ulMsg)
```

```
{

   /*******************************************************/
   /* check for keystrokes                                */
   /*******************************************************/

   case  WM_CHAR :
      if (CHARMSG(&ulMsg)->fs&KC_CHAR)
      {

         /*************************************************/
         /* test for what is allowed                      */
         /*************************************************/

         if (!isdigit(CHARMSG(&ulMsg)->chr) &&
            (CHARMSG(&ulMsg )->chr != '\b'))
         {

            WinMessageBox(HWND_DESKTOP,
                          HWND_DESKTOP,
            "Only numeric characters are allowed in this field",
                          "Numeric Field",
                          0,
                          MB_OK|MB_ERROR);

            return  MRFROMSHORT(TRUE);
         }                            /* endif               */
      }                               /* endif               */
      break;
   case  EM_PASTE :
      {

         /*************************************************/
         /* check for pasting from the clipboard          */
         /*************************************************/

         HAB               habAnchor;
         PCHAR             pchText;
         CHAR              achText[1024];
         USHORT            usIndex;

         habAnchor = WinQueryAnchorBlock(hwndEntry);
         WinOpenClipbrd(habAnchor);
         pchText = (PCHAR)WinQueryClipbrdData(habAnchor,
                                              CF_TEXT);

         if (pchText)
         {

            strcpy(achText,
                   pchText);
            WinCloseClipbrd(habAnchor);

            usIndex = 0;

            /**********************************************/
            /* loop through string checking for non-numeric */
            /**********************************************/

            while (achText[usIndex])
            {
               if (!isdigit(achText[usIndex++]))
               {
```

```
            WinMessageBox(HWND_DESKTOP,
                          HWND_DESKTOP,
                          "Only numeric characters are "
                             "allowed in this field",
                          "Numerical Field",
                          0,
                          MB_OK|MB_ERROR);

                  return  MRFROMSHORT(TRUE);
               }                        /* endif                */
            }                           /* endwhile             */
         }

         else
         {
            WinCloseClipbrd(habAnchor);
         }                              /* endif                */
      }
      break;
   default  :
      break;
}                                       /* endswitch            */
return (*pfnOldEntryProc)(hwndEntry,
                          ulMsg,
                          mpParm1,
                          mpParm2);
}
```

**SUBCLASS.MAK**

```
SUBCLASS.EXE:                   SUBCLASS.OBJ
        LINK386 @<<
SUBCLASS
SUBCLASS
SUBCLASS
OS2386
SUBCLASS
<<

SUBCLASS.OBJ:                   SUBCLASS.C
        ICC -C+ -Kb+ -Ss+ SUBCLASS.C
```

**SUBCLASS.DEF**

```
NAME SUBCLASS WINDOWAPI

DESCRIPTION 'Subclass example
Copyright (c) 1992-1995 by Kathleen Panov
All rights reserved.'

STACKSIZE 16384
```

The first part of the program should look fairly familiar by now; we're just creating a basic client window. In the WM_CREATE message processing, we post a UM_CREATEDONE message to indicate the client window has been created completely.

In the UM_CREATEDONE processing, we create an entry field using *WinCreateWindow*. After the window is created, *WinSubclassWindow* is called to subclass the default window procedure for an entry field.

```
pfnOldEntryProc = WinSubclassWindow(hwndEntry,
                                    newEntryWndProc);

WinSetWindowPtr(hwndEntry,
                QWL_USER,
                (PVOID)pfnOldEntryProc);
```

The first parameter is the window to subclass, *hwndEntryField*. The second parameter is a pointer to the procedure that messages to the window will be sent to. *WinSubclassWindow* returns the old window procedure, and this pointer is stored in the window word for the entry field.

*newEntryWndProc* is designed to handle only two messages, WM_CHAR and EM_PASTE. All the other messages will be passed to the normal window procedure for entry fields.

```
if (CHARMSG(&ulMsg)->fs&KC_CHAR)
         {
            if (!isdigit(CHARMSG(&ulMsg)->chr) &&
               (CHARMSG(&ulMsg )->chr != '\b'))
            {

               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
               "Only numeric characters are allowed in this field",
                             "Numeric Field",
                             0,
                             MB_OK|MB_ERROR);

               return  MRFROMSHORT(TRUE);
```

The WM_CHAR processing is fairly straightforward. We will look at all the KC_CHAR keys. The only character keys we want to allow are the digits 0 to 9 and the Backspace key. The other editing keys set the KC_VIRTUALKEY flag, not the KC_CHAR flag, so they will be allowed. If a nonnumeric character is entered, *WinMessageBox* is called to pop up an error message, telling the user that only numeric keys are allowed in this field. Next, we return TRUE in order to prevent the character from being processed by the next procedure called for the entry field.

The other message we want to intercept is EM_PASTE. This message is generated whenever text is pasted into the entry field from the clipboard. Remember, the keyboard is not the only method of entering text in an entry field. To determine if the data is valid, we have to take a peek at what is in the clipboard.

```
habAnchor = WinQueryAnchorBlock(hwndEntry);
WinOpenClipbrd(habAnchor);
pchText = (PCHAR)WinQueryClipbrdData(habAnchor,
                                     CF_TEXT);
```

The clipboard is opened by calling *WinOpenClipbrd*.

```
BOOL WinOpenClipbrd( HAB hab )
```

There is only one parameter for the function, the anchor block. This gives ownership of the clipboard to the application window. No other window can open the clipboard while it is open. This is potentially a very dangerous situation. If the clipboard is already open when *WinOpenClipbrd* is called, the function will not return until the clipboard can be opened. Presumably most programs out there are well behaved and will close the clipboard as soon as they are done, but programmers must beware: If programs don't close the clipboard, the message queue will be frozen unless the clipboard is opened in another thread. Once the clipboard is opened, *WinQueryClipbrdData* is called.

```
ULONG WinQueryClipbrdData(HAB hab,
                          ULONG fmt);
```

This function has two parameters, the anchor block and the clipboard data format that is to be retrieved. In our case, we are concerned only with text, so the format CF_TEXT is used. Table 27.1 presents the other possible values for formats.

Table 27.1 Clipboard Formats

| Value | Description |
|---|---|
| CF_TEXT | Text format |
| CF_DSPTEXT | Private text display format |
| CF_BITMAP | Bitmap |
| CF_DSPBITMAP | Private bitmat display format |
| CF_METAFILE | Metafile |
| CF_DSPMETAFILE | Private metafile display format |
| CF_PALETTE | Palette |

The function returns a string of the text contained in the clipboard. If no text is in the clipboard, the string will be NULL.

```
strcpy(achText,
       pchText);
WinCloseClipbrd(habAnchor);
usIndex = 0;
while (achText[usIndex])
{
   if (!isdigit(achText[usIndex++]))
   {
      WinMessageBox(HWND_DESKTOP,
                    HWND_DESKTOP,
                    "Only numeric characters are "
                    "allowed in this field",
                    "Numerical Field",
                    0,
                    MB_OK|MB_ERROR);

      return  MRFROMSHORT(TRUE);
    }
}
```

After we have the string, we check each digit to see if it is a numeric character. If not, the error message box is again displayed, and we return TRUE to avoid further processing.

```
return ( *pfnOldEntryProc ) ( hwndEntry,
                              ulMsg,
                              mpParm1,
                              mpParm2 ) ;
```

If the characters entered are valid, or if the message is something other than WM_CHAR or EM_PASTE, it will fall through the switch statement. At this point, we want to call the old procedure for the entry field.

## Superclassing

Suppose the developer wants to create a lot of these numeric-only entry fields. There is an easier way than to call *WinSubclassWindow* for every one that is created—a concept called superclassing. This creates a whole new window class, created using *WinRegisterClass*, that has the subclassed procedure as its default window procedure. In the last example program, we could call *WinRegisterClass* to create a class called WC_NUMERICENTRY. The window procedure used would be *newEntryWndProc*. However, instead of storing the old window procedure in the window word, we could call *WinQueryClassInfo* using the WC_ENTRYFIELD class and return that procedure for all the messages that we are not handling.

Subclassing is a very easy way to modify the existing controls in Presentation Manager to work the way we want them to. A lot of powerful things can be done using subclassing or superclassing.

# Chapter 28

# Presentation Manager Printing

One of the more profound limitations of DOS was that if an application needed to support many different screens and/or printers, display- and/or printer-specific code had to be written for each type of device. Even though the better programmers could make the job easier with a good design, the effort required to code and support the multitude of output devices was often disheartening enough to dissuade all but commercial developers from attempting the feat.

When the Presentation Manager was added to OS/2 1.1, the concept of output device independence finally became an attainable reality because of the layer of abstraction that a handle to the *presentation space* (HPS) provides; the HPS contains only the settings of the current logical attributes (color, fill type, line type, etc.) that were set by default or by the application. The binding of this (and thus the mapping of the logical attributes to their physical counterparts) to a specific device is done by associating the HPS to a *device context.* The device context (HDC) contains the actual attributes being used and other things, such as the size of the displayable area. This association between HPS and HDC is done either with the *GpiAssociate* call or when the HPS is created by using the GPIA_ASSOC flag in the *GpiCreatePS* call.

Knowing this, it probably is evident that by associating the HPS with an HDC that corresponds to the appropriate device, an application can create output on that device without any changes to the code. This is almost correct; Presentation Manager is more attuned to the display device than to the printer, since the display is used significantly more than the printer. Thus, when drawing to the screen, PM eliminates the need for a lot of the coding details that are necessary when drawing to a printer or plotter.

Still, this is much better than how DOS does it (or doesn't do it, depending on how it is looked at).

This chapter discusses the details of establishing a "connection" with a hardcopy device and the associated bells and whistles that can be created to ensure that an application has to do as little work as needed.

## A Printer's Overview

Before we begin, a bit of overview regarding printing system design is needed. As with the output device model, there is a layer of abstraction between the application and the printer. This is the *print queue*, which is associated with a *print port*, which can be a physical port or a networked logical port. The similarity stops here, however, since each print queue is also assigned a *printer driver.* Applications "print" to the print queue, which stores the output in a device-independent format and relies on the printer driver to convert the device-independent graphics commands to device-specific ones. (Actually, the output goes to a *queue processor*, which uses the printer driver to assist it in converting the commands to the printer-specific ones.) Figure 28.1 presents a view of the print subsystem.

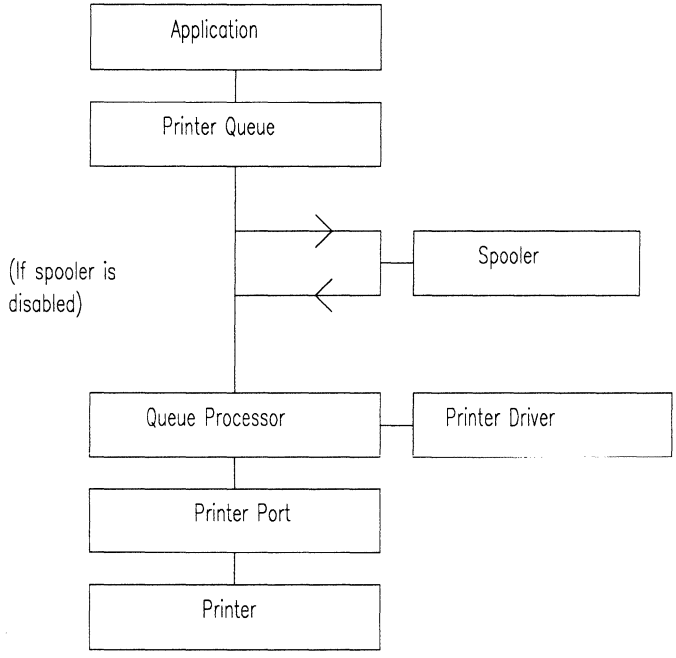



Figure 28.1 A view of the print subsystem.

Now we turn to the pseudocode on which we initially base the sample code. This describes the strategy used for creating hardcopy output. "Draw page" is an abstract term that is defined by the application.

```
Initialize a DEVOPENSTRUC for the desired printer/plotter
Open a device context (HDC)
Create a presentation space (HPS) associated with the printer HDC

Tell the printer that we are starting a print job
Draw page 1
Tell the printer to start a new page
Draw page 2
   :
Tell the printer to start a new page
Draw page n
Tell the printer that we are finished with the print job

Destroy the HPS
Close the printer HDC
```

Two things are worth noting: the reference to the data structure DEVOPENSTRUC and the phrase "tell the printer that..." The DEVOPENSTRUC is explained next; how to tell the printer anything at all is explained later in this chapter.

The DEVOPENSTRUC structure describes the hardcopy device to PM. It contains the following nine fields:

```
typedef struct _DEVOPENSTRUC {
   PSZ pszLogAddress;
   PSZ pszDriverName;
   PDRIVDATA pdriv;
   PSZ pszDataType;
   PSZ pszComment;
   PSZ pszQueueProcName;
   PSZ pszQueueProcParams;
   PSZ pszSpoolerParams;
   PSZ pszNetworkParams;
} DEVOPENSTRUC;
```

*pszLogAddress* points to the name of the printer queue to print to. *pszDriverName* points to the name of the printer driver to be used when converting the output to printer-specific commands. *pdriv* points to printer-specific data to be used when printing—whether to print in portrait or landscape mode. This will be discussed in more detail later in the chapter. *pszDataType* points to the type of output being sent. This can be either PM_Q_STD or PM_Q_RAW, the latter indicating that the application has already converted the output to the appropriate commands for the printer and that the output should pass directly to the printer port. Using this is discouraged, since it does not fit into the strategy of output device independence discussed at the beginning of this chapter. *pszComment* points to a string describing the output being printed. *pszQueueProcName* points to the name of the queue processor to be used. (OS/2 comes with two queue processors— "PMPRINT" and "PMPLOT"; see the function *printDoc* below for determining the default queue processor for a particular printer.) *pszQueueProcParams*, *pszSpoolerParams*, and *pszNetworkParams* point to a set of queue processor parameters, spooler parameters, and network parameters. We will not be using these fields.

The initialized DEVOPENSTRUC is passed to *DevOpenDC* as the fifth parameter, with the number of fields that are initialized as the fourth parameter. As a rule, all nine fields should always be initialized, even though all of them won't be used.

"Telling" the printer to do certain things is accomplished by sending it an "escape code." An escape code is a method of accessing the capabilities of an output device for which there is no API. Two examples of this are starting and ending a print job.

```
LONG DevEscape(HDC hdcDevice,
               LONG lEscCode,
               LONG lSzInData,
               PBYTE pbInData,
               PLONG plSzOutData,
               PBYTE pbOutData);
```

*hdcDevice* is the handle to the device context. *lEscCode* is the DEVESC_ code that you want to issue to the device. *lSzInData* is the size of the data being passed in. *pbInData* points to the data being passed in. *plSzOutData* points to the size of the buffer to receive the results (if any). On return, this variable is updated to reflect the number of bytes actually copied into *pbOutData*. *pbOutData* points to the receiving buffer for the results of the call (if any).

For escape codes that do not have any data, 0 should be specified for *lSzInData* and NULL for *pbInData*, *plSzOutData*, and *pbOutData*.

So, substituting real code where possible in our pseudocode, we now have the following code that reflects the initialization steps to establish the connection between the application and the printer.

```
BOOL printDoc(HAB habAnchor,PCHAR pchName)
{
   DEVOPENSTRUC dosPrinter;
   HDC hdcPrinter;
   SIZEL szlHps;
   HPS hpsPrinter;

   //-------------------------------------------------------------
   // Initialize a DEVOPENSTRUC for the desired printer/plotter
   //-------------------------------------------------------------
   dosPrinter.pszLogAddress="LPT1Q";
   dosPrinter.pszDriverName="PSCRIPT";
   dosPrinter.pdriv=NULL;
   dosPrinter.pszDataType="PM_Q_STD";
   dosPrinter.pszComment=pchName;
   dosPrinter.pszQueueProcName="PMPRINT";
   dosPrinter.pszQueueProcParams=NULL;
   dosPrinter.pszSpoolerParams=NULL;
   dosPrinter.pszNetworkParams=NULL;

   //-------------------------------------------------------------
   // Open a device context (HDC)
   //-------------------------------------------------------------
   hdcPrinter=DevOpenDC(habAnchor,
                        OD_QUEUED,
                        "*",
                        9L,
                        (PDEVOPENDATA)&dosPrinter,
                        NULLHANDLE);
   if (hdcPrinter==NULLHANDLE) {
      //----------------------------------------------------------
      // An error occurred
      //----------------------------------------------------------
      return;
   } /* endif */

   //-------------------------------------------------------------
   // Query the width and height of the printer page
   //-------------------------------------------------------------
   DevQueryCaps(hdcPrinter,CAPS_WIDTH,1L,&szlHps.cx);
   DevQueryCaps(hdcPrinter,CAPS_HEIGHT,1L,&szlHps.cy);

   //-------------------------------------------------------------
   // Create a presentation space (HPS) associated with
   //   the printer HDC
   //-------------------------------------------------------------
   hpsPrinter=GpiCreatePS(habAnchor,
                          hdcPrinter,
                          &szlHps,
                        PU_PELS|GPIT_MICRO|GPIF_DEFAULT|GPIA_ASSOC);
   if (hpsPrinter==NULLHANDLE) {
      //----------------------------------------------------------
      // An error occurred
      //----------------------------------------------------------
      DevCloseDC(hdcPrinter);
      return;
   } /* endif */
```

```
//-------------------------------------------------------------
// Tell the printer that we are starting a print job
//-------------------------------------------------------------
if (DevEscape(hdcPrinter,
              DEVESC_STARTDOC,
              (LONG)strlen(pchName),
              pchName,
              NULL,
              NULL)!=DEV_OK) {
   //----------------------------------------------------------
   // An error occurred
   //----------------------------------------------------------
   GpiDestroyPS(hpsPrinter);
   DevCloseDC(hdcPrinter);
   return;
} /* endif */

//-------------------------------------------------------------
// Draw page 1
//-------------------------------------------------
if (!drawPage(hpsPrinter,1)) {
   //-------------------------------------------
   // An error occurred so abort the print job
   //-------------------------------------------
   DevEscape(hdcPrinter,
             DEVESC_ABORTDOC,
             0L,
             NULL,
             NULL,
             NULL);
   GpiDestroyPS(hpsPrinter);
   DevCloseDC(hdcPrinter);
   return;
} /* endif */

//-------------------------------------------------
// Tell the printer to start a new page
//-------------------------------------------------
if (DevEscape(hdcPrinter,
              DEVESC_NEWFRAME,
              0,
              NULL,
              NULL,
              NULL)!=DEV_OK) {
   //-------------------------------------------
   // An error occurred so abort the print job
   //-------------------------------------------
   DevEscape(hdcPrinter,
             DEVESC_ABORTDOC,
             0L,
             NULL,
             NULL,
             NULL);
   GpiDestroyPS(hpsPrinter);
   DevCloseDC(hdcPrinter);
   return;
} /* endif */

//-------------------------------------------------
// Draw page 2
//-------------------------------------------------
if (!drawPage(hpsPrinter,2)) {
```

```
        //-------------------------------------------
        // An error occurred so abort the print job
        //-------------------------------------------
        DevEscape(hdcPrinter,
                  DEVESC_ABORTDOC,
                  0L,
                  NULL,
                  NULL,
                  NULL);
        GpiDestroyPS(hpsPrinter);
        DevCloseDC(hdcPrinter);
        return;
    } /* endif */

    //-------------------------------------------
    // Tell the printer that we are finished with the print job
    //-------------------------------------------
    if (DevEscape(hdcPrinter,
                  DEVESC_ENDDOC,
                  0L,
                  NULL,
                  NULL,
                  NULL)!=DEV_OK) {
        //-------------------------------------------
        // An error occurred so abort the print job
        //-------------------------------------------
        DevEscape(hdcPrinter,
                  DEVESC_ABORTDOC,
                  0L,
                  NULL,
                  NULL,
                  NULL);
        GpiDestroyPS(hpsPrinter);
        DevCloseDC(hdcPrinter);
        return;
    } /* endif */

    //-------------------------------------------
    // Destroy the HPS and close the printer HDC
    //-------------------------------------------
    GpiDestroyPS(hpsPrinter);
    DevCloseDC(hdcPrinter);
}
```

Looking at the hard-coded values for *pszLogAddress* and *pszDriverName*, it is hard to imagine this as being the device-independent code discussed earlier. Well, that's right. Actually there is a (huge) step before this to initialize the initialization—selecting the printer and any job-specific parameters.

# Where's My Thing?

What we need is a way to figure out what printers and queues are defined so that we do not have to rely on hard-coded values or prompt the user for this information. Instead, we should simply retrieve the needed data and present the user with a choice of printers to print to. Fortunately, this information is obtainable through the spooler (Spl) functions and in particular *SplEnumQueue*.

```
(SPLERR)SplEnumQueue(PSZ pszComputer,
                     ULONG ulLevel,
                     PVOID pvBuf,
                     ULONG ulSzBuf,
                     PULONG pulNumReturned,
                     PULONG pulNumTotal,
                     PULONG pulSzBufNeeded,
                     PVOID pvReserved);
```

*pszComputer* is the name of the computer containing the queues to enumerate. This is for networked printing and can be NULL to specify the local computer. *ulLevel* specifies the amount and type of information to return. *pvBuf* points to a buffer to contain the results. If NULL, the number of bytes needed is returned in *pulSzBufNeeded*. *ulSzBuf* specifies the size of the buffer pointed to by *pvBuf*. If *pvBuf* is NULL, this is ignored. *pulNumReturned* specifies the number of queues returned, while *pulNumTotal* specifies the total number of queues. *pvReserved* is reserved and must be NULL.

The data returned is dependent on the value of *ulLevel* and can be one of those specified in Table 28.1.

Table 28.1 Values of *ulLevel*

| *ulLev* | Data Returned in *pvBuf* |
|---|---|
| 3 | *pvBuf* points to an array of PPRQINFO3 structures. |
| 4 | *pvBuf* points to a list of PPRQINFO3 structures, with each element of the list followed by 0 or more PPRJINFO2 structures describing the jobs currently in the queue. |
| 5 | *pvBuf* points to a queue name. |
| 6 | *pvBuf* points to an array of PPRQINFO6 structures. |

We will be interested in information level 3, which returns all of the information that we will need to eliminate the hard-coded values shown in the preceding code. Let's look at the PRQINFO3 structure in detail.

```
typedef struct _PRQINFO3 {
   PSZ pszName;
   USHORT uPriority;
   USHORT uStartTime;
   USHORT uUntilTime;
   USHORT fsType;
   PSZ pszSepFile;
   PSZ pszPrProc;
   PSZ pszParms;
   PSZ pszComment;
   USHORT fsStatus;
   USHORT cJobs;
   PSZ pszPrinters;
   PSZ pszDriverName;
   PDRIVDATA pDriverData;
} PRQINFO3;
```

*pszName* is the queue name. *uPriority* is the default queue priority and is used to calculate the default job priority for the queue. *uStartTime* is the number of minutes past midnight when the queue becomes active. *uUntilTime* is the number of minutes past midnight when the queue becomes inactive. *fsType* specifies one or more flags describing any characteristics of the queue. *pszSepFile* points to the file name of the separator page. *pszPrProc* points to the name of the queue processor used. *pszParms* points to the default queue processor parameters to be used. *pszComment* points to the description string that is displayed in the Workplace Shell. *fsStatus* specifies one or more flags describing the status of the queue. *cJobs* specifies the number of jobs in the queue. *pszPrinters* specifies one or more printers, separated by commas, that use this queue (for printer pooling). *pszDriverName* specifies the printer driver and device (if the driver supports more than one device) separated by a period. *pDriverData* points to the default driver data to be used.

We will see later that the only information we really need for any printer is the corresponding DEVOPENSTRUC structure and the device name, if the printer driver supports more than one device. The function *createPrnList* enumerates the printers in the system and calls *extractPrnInfo* to initialize the DEVOPENSTRUC structure for the printer. Also included is *destroyPrnList,* which returns any consumed memory to the system.

The following is code for extracting the DEVOPENSTRUC information from a PRQINFO3 structure.

```
typedef struct {
   DEVOPENSTRUC dosPrinter;
   CHAR achDevice[256];
} PRNLISTINFO, *PPRNLISTINFO;

#define CPL_ERROR                (USHORT)0
#define CPL_NOPRINTERS           (USHORT)1
#define CPL_SUCCESS              (USHORT)2

VOID extractPrnInfo(PPRQINFO3 ppiQueue,DEVOPENSTRUC *pdosPrinter)
//--------------------------------------------------------------
// This function extracts the needed information from the specified
// PRQINFO3
// structure and places it into the specifies DEVOPENSTRUC
// structure.
//
// Input:  ppiQueue - points to the PRQINFO3 structure
// Output:  pdosPrinter - points to the initialized DEVOPENSTRUC
// structure
//--------------------------------------------------------------
{
   PCHAR pchPos;

   pdosPrinter->pszLogAddress=ppiQueue->pszName;

   pdosPrinter->pszDriverName=ppiQueue->pszDriverName;
   pchPos=strchr(pdosPrinter->pszDriverName,'.');
   if (pchPos!=NULL) {
      *pchPos=0;
   } /* endif */

   pdosPrinter->pdriv=ppiQueue->pDriverData;
   pdosPrinter->pszDataType="PM_Q_STD";
   pdosPrinter->pszComment=ppiQueue->pszComment;

   if (strlen(ppiQueue->pszPrProc)>0) {
      pdosPrinter->pszQueueProcName=ppiQueue->pszPrProc;
   } else {
      pdosPrinter->pszQueueProcName=NULL;
   } /* endif */

   if (strlen(ppiQueue->pszParms)>0) {
      pdosPrinter->pszQueueProcParams=ppiQueue->pszParms;
   } else {
      pdosPrinter->pszQueueProcParams=NULL;
   } /* endif */

   pdosPrinter->pszSpoolerParams=NULL;
   pdosPrinter->pszNetworkParams=NULL;
}

USHORT createPrnList(HWND hwndListbox)
//--------------------------------------------------------------
// This function enumerates the printers available and inserts them
// into the specified listbox.
//
```

```
// Input:  hwndListbox - handle to the listbox to contain the list
// Returns:  an CPL_* constant
//--------------------------------------------------------------------
{
   SPLERR seError;
   ULONG ulSzBuf;
   ULONG ulNumQueues;
   ULONG ulNumReturned;
   ULONG ulSzNeeded;
   ULONG ulIndex;
   PPRQINFO3 ppiQueue;
   PCHAR pchPos;
   PPRNLISTINFO ppliInfo;
   SHORT sInsert;



   //-----------------------------------------------------------------
   // Get the size of the buffer needed
   //-----------------------------------------------------------------
   seError=SplEnumQueue(NULL,
                        3,
                        NULL,
                        0L,
                        &ulNumReturned,
                        &ulNumQueues,
                        &ulSzNeeded,
                        NULL);
   if (seError!=ERROR_MORE_DATA) {
      return CPL_ERROR;
   } else
   if (ulNumQueues==0) {
      return CPL_NOPRINTERS;
   } /* endif */

   ppiQueue=malloc(ulSzNeeded);
   if (ppiQueue==NULL) {
      return CPL_ERROR;
   } /* endif */

   ulSzBuf=ulSzNeeded;

   //-----------------------------------------------------------------
   // Get the information
   //-----------------------------------------------------------------
   SplEnumQueue(NULL,
                3,
                ppiQueue,
                ulSzBuf,
                &ulNumReturned,
                &ulNumQueues,
                &ulSzNeeded,
                NULL);

   //-----------------------------------------------------------------
   // ulNumReturned has the count of the number of PRQINFO3
   // structures.
   //-----------------------------------------------------------------
   for (ulIndex=0; ulIndex<ulNumReturned; ulIndex++) {
      //--------------------------------------------------------------
      // Since the "comment" can have newlines in it, replace them
      // with spaces
      //--------------------------------------------------------------
      pchPos=strchr(ppiQueue[ulIndex].pszComment,'\n');
      while (pchPos!=NULL) {
         *pchPos=' ';
         pchPos=strchr(ppiQueue[ulIndex].pszComment,'\n');
      } /* endwhile */
```

```
      ppliInfo=malloc(sizeof(PRNLISTINFO));
if (ppliInfo==NULL) {
         continue;
      } /* endif */

      //-------------------------------------------------------------
      // Extract the device name before initializing the
      // DEVOPENSTRUC structure
      //-------------------------------------------------------------
      pchPos=strchr(ppiQueue[ulIndex].pszDriverName,'.');
      if (pchPos!=NULL) {
         *pchPos=0;
         strcpy(ppliInfo->achDevice,pchPos+1);
      } /* endif */

      extractPrnInfo(&ppiQueue[ulIndex],&ppliInfo->dosPrinter);

      sInsert=(SHORT)WinInsertLboxItem(hwndListbox,
                                       0,
                                    ppiQueue[ulIndex].pszComment);

      WinSendMsg(hwndListbox,
                 LM_SETITEMHANDLE,
                 MPFROMSHORT(sInsert),
                 MPFROMP(ppliInfo));

      if (( ppiQueue[ulIndex].fsType &
               PRQ3_TYPE_APPDEFAULT) != 0 ) {
         WinSendMsg( hwndListbox,
                     LM_SELECTITEM,
                     MPFROMSHORT( sInsert ),
                     MPFROMSHORT( TRUE ));
      } /* endif */
   } /* endfor */

   free(ppiQueue);
   return CPL_SUCCESS;
}

VOID destroyPrnList(HWND hwndListbox)
//-------------------------------------------------------------------
// This function destroys the printer list and returns the memory
// to the system.
//
// Input:  hwndListbox - handle of the listbox containing the
// printer list
//-------------------------------------------------------------------
{
   USHORT usNumItems;
   USHORT usIndex;
   PPRNLISTINFO ppliInfo;

   usNumItems=WinQueryLboxCount(hwndListbox);

   for (usIndex=0; usIndex<usNumItems; usIndex++) {
      ppliInfo=(PPRNLISTINFO)PVOIDFROMMR(WinSendMsg(hwndListbox,
                                          LM_QUERYITEMHANDLE,
                                           MPFROMSHORT(usIndex),
                                           0L));
      if (ppliInfo!=NULL) {
         free(ppliInfo);
      } /* endif */
   } /* endfor */

   WinSendMsg(hwndListbox,LM_DELETEALL,0L,0L);
}
```

# I Want That with Mustard, Hold the Mayo, No Onions, Extra Ketchup

Okay, so now we have the printer selection tools needed (you're going to have to write the dialog procedure!), but what if the user wants the printer output to go to a file, for example? In a restaurant, when we want to order an entree, we usually can see what it comes with ("a vegetable and a choice of salad or a dessert"). With printers, the same concept applies; it is referred to as the *job properties* (or as the *printer driver data*). These are usually specific to the printer type and can specify portrait or landscape mode and so on. These job properties are stored in the *pdriv* field of the DEVOPENSTRUC structure and are queried and changed via the *DevPostDeviceModes* function.

```
(LONG)DevPostDeviceModes(HAB habAnchor,
                                    PDRIVDATA pddData,
                                    PSZ pszDriver,
                                    PSZ pszDevice,
                                    PSZ pszPrinter,
                                    ULONG ulOptions);
```

*habAnchor* is the anchor block of the thread calling the function. *pddData* is used to store the results. If NULL, this function returns the number of bytes needed to store the data. *pszDriver* is the printer driver name and corresponds to the *pszDriverName* field of the DEVOPENSTRUC structure. *pszDevice* is the device name and corresponds to the *achDevice* field of our PRNLISTINFO structure. *pszPrinter* is the key name passed to *PrfQueryProfileData* and is passed to the *queryPrinter* routine (and is stored in the *achPrinter* field). Finally, *ulOptions* can be one of three DPDM_ constants, as specified in Table 28.2.

Table 28.2 Values of *ulOptions*

| Constant | Description |
|---|---|
| DPDM_QUERYJOBPROP | Returns the default data in *pddData*. |
| DPDM_POSTJOBPROP | Displays the printer-specific dialog box containing the job properties and the forms list. If *pszPrinter* is NULL, the initial values displayed on the dialog box are taken from the *pddData* field. |
| DPDM_CHANGEPROP | Displays first the DPDM_POSTJOBPROP dialog box and then displays the "printer-properties" dialog box, allowing the user to change any permanent settings regarding the printer. |

This information now allows us to provide a "Properties" button on a printer selection dialog box. Note that normally the DPDM_QUERYJOBPROP option isn't needed since the *SplEnumQueue* returns this information. We have all of the tools needed to query the printers defined for the system, the data specific to each, and the job properties.

# Where Were We?

Looking back, we now know that somewhere before the initialization of the DEVOPENSTRUC structure, we need to display a dialog box allowing the user to select which printer to print the document on and any job properties he or she wishes to use. From the values returned, we can properly initialize the DEVOPENSTRUC structure with non–hard-coded values, thereby increasing our device independence. To firm up our knowledge, the following is a simple example program that prints a box.

**PRINT.C**

```
#define  INCL_DEV
#define  INCL_DOSERRORS
#define  INCL_SPL
#define  INCL_SPLDOSPRINT
#define  INCL_WINERRORS
#define  INCL_WININPUT
#define  INCL_WINLISTBOXES
#define  INCL_WINMENUS
#define  INCL_WINSHELLDATA
#define  INCL_WINSYS
#define  INCL_WINWINDOWMGR
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>
#include "print.h"
#define  CLS_CLIENT     "SampleClass"
#define  CPL_ERROR      (USHORT) 0
#define  CPL_NOPRINTERS (USHORT) 1
#define  CPL_SUCCESS    (USHORT) 2
typedef VOID(*_Optlink PRNTHREAD)(PVOID);

typedef          struct
{
   DEVOPENSTRUC     dosPrinter;
   CHAR             achDevice[256];
} PRNLISTINFO,*PPRNLISTINFO;

typedef          struct
{
   ULONG            ulSizeStruct;
   HWND             hwndOwner;
   DEVOPENSTRUC     dosPrinter;
} PRNTHREADINFO,*PPRNTHREADINFO;

typedef struct   _CLIENTINFO
{
   HWND             hwndListbox;
} CLIENTINFO,*PCLIENTINFO;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

VOID _Optlink printThread(PPRNTHREADINFO pptiInfo);
BOOL drawPage(HPS hpsDraw,USHORT usPage);
USHORT createPrnList(HWND hwndListbox);
VOID destroyPrnList(HWND hwndListbox);
VOID extractPrnInfo(PPRQINFO3 ppiQueue,DEVOPENSTRUC *pdosPrinter)
;

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);
```

```
   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    sizeof(PVOID));

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU|FCF_MENU|FCF_ACCELTABLE;

   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                                  "Printing Sample Application",
                                  0L,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   if (hwndFrame != NULLHANDLE)
   {
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
      while (bLoop)
      {
         WinDispatchMsg(habAnchor,
                        &qmMsg);
         bLoop = WinGetMsg(habAnchor,
                           &qmMsg,
                           NULLHANDLE,
                           0,
                           0);
      }                                  /* endwhile              */
      WinDestroyWindow(hwndFrame);
   }                                     /* endif                 */
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   PCLIENTINFO      pciInfo;

   pciInfo = WinQueryWindowPtr(hwndClient,
                               0);

   switch (ulMsg)
   {
      case  WM_CREATE :
         {
         //-------------------------------------------------------
         // Allocate and initialize the CLIENTINFO structure
         //-------------------------------------------------------
            pciInfo = malloc(sizeof(CLIENTINFO));
            if (pciInfo == NULL)
            {
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                        "An error occurred in initialization.",
                             "Error",
                             0,
                             MB_OK|MB_INFORMATION);
```

```
            return  MRFROMSHORT(TRUE);
         }                          /* endif                */

         WinSetWindowPtr(hwndClient,
                         0,
                         pciInfo);

         pciInfo->hwndListbox = WinCreateWindow(hwndClient,
                                                WC_LISTBOX,
                                                "",
                                                LS_HORZSCROLL|
                                               LS_NOADJUSTPOS,
                                                0,
                                                0,
                                                0,
                                                0,
                                                hwndClient,
                                                HWND_TOP,
                                                WND_LISTBOX,
                                                NULL,
                                                NULL);
         if (pciInfo->hwndListbox == NULLHANDLE)
         {
            WinMessageBox(HWND_DESKTOP,
                          HWND_DESKTOP,
                        "An error occurred in initialization.",
                          "Error",
                          0,
                          MB_OK|MB_INFORMATION);
            free(pciInfo);
            return  MRFROMSHORT(TRUE);
         }                          /* endif                */

         WinSendMsg(hwndClient,
                    WM_COMMAND,
                    MPFROMSHORT(MI_REFRESH),
                    0L);
      }
      break;
   case  WM_DESTROY :
   //-------------------------------------------------------
   // Return the resources to the system
   //-------------------------------------------------------
      destroyPrnList(pciInfo->hwndListbox);
      WinDestroyWindow(pciInfo->hwndListbox);
      break;
   case  WM_SIZE :
      WinSetWindowPos(pciInfo->hwndListbox,
                      NULLHANDLE,
                      0,
                      0,
                      SHORT1FROMMP(mpParm2),
                      SHORT2FROMMP(mpParm2),
                      SWP_MOVE|SWP_SIZE|SWP_SHOW);
      break;
   case  WM_INITMENU :
      switch (SHORT1FROMMP(mpParm1))
      {
         case  M_SAMPLE :
            {
               SHORT            sSelect;
               PPRNLISTINFO     ppliInfo;

               sSelect = WinQueryLboxSelectedItem
                  (pciInfo->hwndListbox);
```

```
            ppliInfo = (PPRNLISTINFO)PVOIDFROMMR(WinSendMsg
               (pciInfo->hwndListbox,
                LM_QUERYITEMHANDLE,
                MPFROMSHORT(sSelect),
                0L));
         //-------------------------------------------------
         // If no printer is selected, disable the print
         // menuitem
         //-------------------------------------------------

            if (sSelect != LIT_NONE)
            {
               WinEnableMenuItem(HWNDFROMMP(mpParm2),
                                 MI_PRINT,
                                 TRUE);
            }
            else
            {
               WinEnableMenuItem(HWNDFROMMP(mpParm2),
                                 MI_PRINT,
                                 FALSE);
            }                    /* endif                */
         //-------------------------------------------------
         // If no printer is selected or there is no driver
         // data, disable the setup menuitem
         //-------------------------------------------------
            if ((sSelect != LIT_NONE) && (ppliInfo != NULL)
               && (ppliInfo->dosPrinter.pdriv != NULL))
            {
               WinEnableMenuItem(HWNDFROMMP(mpParm2),
                                 MI_SETUP,
                                 TRUE);
            }
            else
            {
               WinEnableMenuItem(HWNDFROMMP(mpParm2),
                                 MI_SETUP,
                                 FALSE);
            }                    /* endif                */
         }
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                             /* endswitch            */
   break;

case  WM_COMMAND :
   switch (SHORT1FROMMP(mpParm1))
   {
      case  MI_PRINT :
         {
            SHORT            sIndex;
            PPRNLISTINFO     ppliInfo;
            PPRNTHREADINFO   pptiInfo;
         //-------------------------------------------------
         // Query the selected printer
         //-------------------------------------------------

            sIndex = WinQueryLboxSelectedItem
               (pciInfo->hwndListbox);
```

```
            ppliInfo = (PPRNLISTINFO)PVOIDFROMMR(WinSendMsg
               (pciInfo->hwndListbox,
                LM_QUERYITEMHANDLE,
                MPFROMSHORT(sIndex),
                0L));
         //-----------------------------------------------------
         // Allocate and initialize the PRNTHREADINFO
         // structure to pass to the thread
         //-----------------------------------------------------

            pptiInfo = malloc(sizeof(PRNTHREADINFO));
            if (pptiInfo == NULL)
            {
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                "An error occurred while trying to print",

                             "Error",
                             0,
                             MB_OK|MB_INFORMATION);
               return  MRFROMSHORT(TRUE);
            }                    /* endif                */

            pptiInfo->ulSizeStruct = sizeof(PRNTHREADINFO);
            pptiInfo->hwndOwner = hwndClient;
            pptiInfo->dosPrinter = ppliInfo->dosPrinter;
         //-----------------------------------------------------
         // Create the thread to do the printing
         //-----------------------------------------------------

            if (_beginthread((PRNTHREAD)printThread,
                             NULL,
                             0x8000,
                             (PVOID)pptiInfo) == -1)
            {
               WinMessageBox(HWND_DESKTOP,
                             HWND_DESKTOP,
                 "An error occurred while trying to print",

                             "Error",
                             0,
                             MB_OK|MB_INFORMATION);
            }                    /* endif                */
         }
         break;
      case  MI_SETUP :
         {
            USHORT           usSelect;
            PPRNLISTINFO     ppliInfo;
         //-----------------------------------------------------
         // Query the selected printer
         //-----------------------------------------------------

            usSelect = WinQueryLboxSelectedItem
               (pciInfo->hwndListbox);
            ppliInfo = (PPRNLISTINFO)PVOIDFROMMR(WinSendMsg
               (pciInfo->hwndListbox,
                LM_QUERYITEMHANDLE,
                MPFROMSHORT(usSelect),
                0L));
```

```
            DevPostDeviceModes(WinQueryAnchorBlock
               (hwndClient),
                                 ppliInfo->dosPrinter.pdriv,
            ppliInfo->dosPrinter.pszDriverName,
                                  ppliInfo->achDevice,
                                  NULL,
                                  DPDM_POSTJOBPROP);
         }
         break;
      case  MI_REFRESH :
         {
            USHORT           usResult;

            destroyPrnList(pciInfo->hwndListbox);

            usResult = createPrnList(pciInfo->hwndListbox);

            switch (usResult)
            {
               case  CPL_ERROR :
                  WinMessageBox(HWND_DESKTOP,
                                HWND_DESKTOP,
                   "An error occurred while refreshing the list",

                                "Error",
                                0,
                                MB_OK|MB_INFORMATION);
                  break;
               case  CPL_NOPRINTERS :
                  WinMessageBox(HWND_DESKTOP,
                                HWND_DESKTOP,
                  "There are no printers defined.",

                                "Warning",
                                0,
                                MB_OK|MB_INFORMATION);
                  break;
               default  :
                  WinSendMsg(WinWindowFromID(hwndClient,
                                             WND_LISTBOX),
                             LM_SELECTITEM,
                             MPFROMSHORT(0),
                             MPFROMSHORT(TRUE));
                  break;
            }                    /* endswitch            */
         }
         break;
      case  MI_EXIT :
         WinPostMsg(hwndClient,
                    WM_CLOSE,
                    0L,
                    0L);
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                             /* endswitch            */
   break;

case  WM_PAINT :
   {
      HPS              hpsPaint;
      RECTL            rclPaint;
```

```
            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     &rclPaint);
            WinFillRect(hpsPaint,
                        &rclPaint,
                        SYSCLR_WINDOW);
            WinEndPaint(hpsPaint);
         }
         break;
      default  :
         return  WinDefWindowProc(hwndClient,
                                  ulMsg,
                                  mpParm1,
                                  mpParm2);

   }                                    /* endswitch            */
   return  MRFROMSHORT(FALSE);
}

VOID _Optlink printThread(PPRNTHREADINFO pptiInfo)
//--------------------------------------------------------------
// This function is the secondary thread which prints the output.
//
// Input:  pptiInfo - points to the PRNTHREADINFO structure
//                    containing needed information
//--------------------------------------------------------------
{
   HAB               habAnchor;
   HMQ               hmqQueue;
   HDC               hdcPrinter;
   SIZEL             szlHps;
   HPS               hpsPrinter;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                                0);
   WinCancelShutdown(hmqQueue,
                     TRUE);
   //-----------------------------------------------------------
   // Open a device context (HDC)
   //-----------------------------------------------------------

   hdcPrinter = DevOpenDC(habAnchor,
                          OD_QUEUED,
                          "*",
                          9L,
                          (PDEVOPENDATA)&pptiInfo->dosPrinter,
                          NULLHANDLE);

   if (hdcPrinter == NULLHANDLE)
   {
      //--------------------------------------------------------
      // An error occurred
      //--------------------------------------------------------
      WinAlarm(HWND_DESKTOP,
               WA_ERROR);
      WinMessageBox(HWND_DESKTOP,
                    pptiInfo->hwndOwner,
                    "Error creating the device context.",
                    "Error",
                    0,
                    MB_INFORMATION|MB_OK);

      WinDestroyMsgQueue(hmqQueue);
      WinTerminate(habAnchor);
      free(pptiInfo);
```

```
   _endthread();
}                                       /* endif                */
//-------------------------------------------------------------
// Query the width and height of the printer page
//-------------------------------------------------------------
DevQueryCaps(hdcPrinter,
             CAPS_WIDTH,
             1L,
             &szlHps.cx);

DevQueryCaps(hdcPrinter,
             CAPS_HEIGHT,
             1L,
             &szlHps.cy);
//-------------------------------------------------------------
// Create a presentation space (HPS) associated with the
// printer HDC
//-------------------------------------------------------------
hpsPrinter = GpiCreatePS(habAnchor,
                         hdcPrinter,
                         &szlHps,
                         PU_LOENGLISH|GPIT_MICRO|GPIA_ASSOC);
if (hpsPrinter == NULLHANDLE)
{
   //----------------------------------------------------------
   // An error occurred
   //----------------------------------------------------------
   DevCloseDC(hdcPrinter);

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
   WinMessageBox(HWND_DESKTOP,
                 pptiInfo->hwndOwner,
                 "Error creating the presentation space.",
                 "Error",
                 0,
                 MB_INFORMATION|MB_OK);

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   free(pptiInfo);
   _endthread();
}                                       /* endif                */
//-------------------------------------------------------------
// Tell the printer that we are starting a print job
//-------------------------------------------------------------
if (DevEscape(hdcPrinter,
              DEVESC_STARTDOC,
              strlen(pptiInfo->dosPrinter.pszComment),
              pptiInfo->dosPrinter.pszComment,
              NULL,
              NULL) != DEV_OK)
{
   //----------------------------------------------------------
   // An error occurred
   //----------------------------------------------------------
   GpiDestroyPS(hpsPrinter);
   DevCloseDC(hdcPrinter);

   WinAlarm(HWND_DESKTOP,
            WA_ERROR);
```

```
    WinMessageBox(HWND_DESKTOP,
                  pptiInfo->hwndOwner,
                  "Error starting the print job.",
                  "Error",
                  0,
                  MB_INFORMATION|MB_OK);

    WinDestroyMsgQueue(hmqQueue);
    WinTerminate(habAnchor);
    free(pptiInfo);
    _endthread();
}                                       /* endif                */
//-------------------------------------------------------------
// Draw sample output
//-------------------------------------------------------------
if (!drawPage(hpsPrinter,
              1))
{
    //---------------------------------------------------------
    // An error occurred so abort the print job
    //---------------------------------------------------------
    DevEscape(hdcPrinter,
              DEVESC_ABORTDOC,
              0L,
              NULL,
              NULL,
              NULL);
    GpiDestroyPS(hpsPrinter);
    DevCloseDC(hdcPrinter);

    WinAlarm(HWND_DESKTOP,
             WA_ERROR);
    WinMessageBox(HWND_DESKTOP,
                  pptiInfo->hwndOwner,
                  "Error drawing the printer page.",
                  "Error",
                  0,
                  MB_INFORMATION|MB_OK);

    WinDestroyMsgQueue(hmqQueue);
    WinTerminate(habAnchor);
    free(pptiInfo);
    _endthread();
}                                       /* endif                */
//-------------------------------------------------------------
// Tell the printer that we are finished with the print job
//-------------------------------------------------------------
if (DevEscape(hdcPrinter,
              DEVESC_ENDDOC,
              0L,
              NULL,
              NULL,
              NULL) != DEV_OK)
{
    //---------------------------------------------------------
    // An error occurred so abort the print job
    //---------------------------------------------------------
    DevEscape(hdcPrinter,
              DEVESC_ABORTDOC,
              0L,
              NULL,
              NULL,
              NULL);
    GpiDestroyPS(hpsPrinter);
    DevCloseDC(hdcPrinter);
```

```
      WinAlarm(HWND_DESKTOP,
               WA_ERROR);
      WinMessageBox(HWND_DESKTOP,
                    pptiInfo->hwndOwner,
                    "Error ending the print job.",
                    "Error",
                    0,
                    MB_INFORMATION|MB_OK);

      WinDestroyMsgQueue(hmqQueue);
      WinTerminate(habAnchor);
      free(pptiInfo);
      _endthread();
   }                                    /* endif                */
   //-------------------------------------------------------------
   // Destroy the HPS and close the printer HDC
   //-------------------------------------------------------------
   GpiDestroyPS(hpsPrinter);
   DevCloseDC(hdcPrinter);
   WinAlarm(HWND_DESKTOP,
            WA_NOTE);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   free(pptiInfo);
   _endthread();
}

BOOL drawPage(HPS hpsDraw,USHORT usPage)
//-------------------------------------------------------------------
// This function draws a box for page 1.  Any other page number
// returns an error.
//
// Input:  usPage - specifies the page number to draw.
// Returns:  TRUE if successful, FALSE otherwise
//-------------------------------------------------------------------
{
   switch (usPage)
   {
      case 1 :
         {
            POINTL           ptlMove;

            ptlMove.x = 10;
            ptlMove.y = 10;

            GpiMove(hpsDraw,
                    &ptlMove);

            ptlMove.x = 100;
            ptlMove.y = 100;

            GpiBox(hpsDraw,
                   DRO_OUTLINE,
                   &ptlMove,
                   0,
                   0);
         }
         break;
      default  :
         return  FALSE;
   }                                    /* endswitch            */
   return  TRUE;
}
```

```
USHORT createPrnList(HWND hwndListbox)
//------------------------------------------------------------
// This function enumerates the printers available and inserts
// them into the specified listbox.
//
// Input:  hwndListbox - handle to the listbox to contain the
//                       list
// Returns:  an CPL_* constant
//------------------------------------------------------------
{
   SPLERR           seError;
   ULONG            ulSzBuf;
   ULONG            ulNumQueues;
   ULONG            ulNumReturned;
   ULONG            ulSzNeeded;
   ULONG            ulIndex;
   PPRQINFO3        ppiQueue;
   PCHAR            pchPos;
   PPRNLISTINFO     ppliInfo;
   SHORT            sInsert;
   //---------------------------------------------------------
   // Get the size of the buffer needed
   //---------------------------------------------------------

   seError = SplEnumQueue(NULL,
                          3,
                          NULL,
                          0L,
                          &ulNumReturned,
                          &ulNumQueues,
                          &ulSzNeeded,
                          NULL);
   if (seError != ERROR_MORE_DATA)
   {
      return  CPL_ERROR;
   }
   else

      if (ulNumQueues == 0)
      {
         return  CPL_NOPRINTERS;
      }                                  /* endif                */

   ppiQueue = malloc(ulSzNeeded);
   if (ppiQueue == NULL)
   {
      return  CPL_ERROR;
   }                                     /* endif                */

   ulSzBuf = ulSzNeeded;
   //---------------------------------------------------------
   // Get the information
   //---------------------------------------------------------

   SplEnumQueue(NULL,
                3,
                ppiQueue,
                ulSzBuf,
                &ulNumReturned,
                &ulNumQueues,
                &ulSzNeeded,
                NULL);
   //---------------------------------------------------------
   // ulNumReturned has the count of the number of PRQINFO3
   // structures.
   //---------------------------------------------------------
```

```
   for (ulIndex = 0; ulIndex < ulNumReturned; ulIndex++)
   {
      //-------------------------------------------------------
      // Since the "comment" can have newlines in it, replace
      // them with spaces
      //-------------------------------------------------------
      pchPos = strchr(ppiQueue[ulIndex].pszComment,
                      '\n');
      while (pchPos != NULL)
      {
         *pchPos = ' ';
         pchPos = strchr(ppiQueue[ulIndex].pszComment,
                         '\n');
      }                                  /* endwhile              */
      ppliInfo = malloc(sizeof(PRNLISTINFO));
      if (ppliInfo == NULL)
      {
         continue;
      }                                  /* endif                 */
      //-------------------------------------------------------
      // Extract the device name before initializing the
      // DEVOPENSTRUC structure
      //-------------------------------------------------------
      pchPos = strchr(ppiQueue[ulIndex].pszDriverName,
                      '.');
      if (pchPos != NULL)
      {
         *pchPos = 0;
         strcpy(ppliInfo->achDevice,
                pchPos+1);
      }                                  /* endif                 */
      extractPrnInfo(&ppiQueue[ulIndex],
                     &ppliInfo->dosPrinter);

      sInsert = (SHORT)WinInsertLboxItem(hwndListbox,
                                         0,
                                         ppiQueue[ulIndex].
                                            pszComment);

      WinSendMsg(hwndListbox,
                 LM_SETITEMHANDLE,
                 MPFROMSHORT(sInsert),
                 MPFROMP(ppliInfo));

      if ((ppiQueue[ulIndex].fsType&PRQ3_TYPE_APPDEFAULT) != 0)
      {
         WinSendMsg(hwndListbox,
                    LM_SELECTITEM,
                    MPFROMSHORT(sInsert),
                    MPFROMSHORT(TRUE));
      }                                  /* endif                 */
   }                                     /* endfor                */
   free(ppiQueue);
   return  CPL_SUCCESS;
}

VOID destroyPrnList(HWND hwndListbox)
//-------------------------------------------------------------
// This function destroys the printer list and returns the memory
// to the system.
//
// Input:  hwndListbox - handle of the listbox containing the
//                       printer list
//-------------------------------------------------------------
{
   USHORT           usNumItems;
   USHORT           usIndex;
```

```
   PPRNLISTINFO     ppliInfo;

   usNumItems = WinQueryLboxCount(hwndListbox);

   for (usIndex = 0; usIndex < usNumItems; usIndex++)
   {
      ppliInfo = (PPRNLISTINFO)PVOIDFROMMR(WinSendMsg(hwndListbox,
                                        LM_QUERYITEMHANDLE,
                                        MPFROMSHORT(usIndex),
                                                    0L));
      if (ppliInfo != NULL)
      {
         //-------------------------------------------------------
         // The following 3 lines help fix a bug from the 1st
         // edition
         //-------------------------------------------------------
         if (ppliInfo->dosPrinter.pdriv != NULL)
         {
            free(ppliInfo->dosPrinter.pdriv);
         }                                /* endif               */
         free(ppliInfo);
      }                                   /* endif               */
   }                                      /* endfor              */
   WinSendMsg(hwndListbox,
              LM_DELETEALL,
              0L,
              0L);
}

VOID extractPrnInfo(PPRQINFO3 ppiQueue,DEVOPENSTRUC *pdosPrinter)
//---------------------------------------------------------------
// This function extracts the needed information from the
// specified PRQINFO3 structure and places it into the specifies
// DEVOPENSTRUC structure.
//
// Input:  ppiQueue - points to the PRQINFO3 structure
// Output:  pdosPrinter - points to the initialized DEVOPENSTRUC
//                        structure
//---------------------------------------------------------------
{
   PCHAR            pchPos;

   pdosPrinter->pszLogAddress = ppiQueue->pszName;

   pdosPrinter->pszDriverName = ppiQueue->pszDriverName;
   pchPos = strchr(pdosPrinter->pszDriverName,
                   '.');
   if (pchPos != NULL)
   {
      *pchPos = 0;
   }                                   /* endif               */
   //-----------------------------------------------------------
   // The following if statement helps fix a bug from the 1st
   // edition.  The sample originally copied the pointer into
   // the DEVOPENSTRUC structure, but the memory into which
   // the memory was pointing was freed by createPrnList().
   //
   // The fix, obviously, is to allocate our own area of memory
   // and to copy the printer data to the new buffer.  Note that
   // destroyPrnList() must free the memory.  destroyPrnList()
   // doesn't get called until the application exits.
   //-----------------------------------------------------------
   if (ppiQueue->pDriverData != NULL)
   {
      pdosPrinter->pdriv = malloc(ppiQueue->pDriverData->cb);
      if (pdosPrinter->pdriv == NULL)
      {
```

```
            return ;
         }                                  /* endif                */
         memcpy(pdosPrinter->pdriv,
                ppiQueue->pDriverData,
                ppiQueue->pDriverData->cb);
      }
      else
      {
         pdosPrinter->pdriv = NULL;
      }                                     /* endif                */
      pdosPrinter->pszDataType = "PM_Q_STD";
      pdosPrinter->pszComment = ppiQueue->pszComment;

      if (strlen(ppiQueue->pszPrProc) > 0)
      {
         pdosPrinter->pszQueueProcName = ppiQueue->pszPrProc;
      }
      else
      {
         pdosPrinter->pszQueueProcName = NULL;
      }                                      /* endif                */
      if (strlen(ppiQueue->pszParms) > 0)
      {
         pdosPrinter->pszQueueProcParams = ppiQueue->pszParms;
      }
      else
      {
         pdosPrinter->pszQueueProcParams = NULL;
      }                                      /* endif                */
      pdosPrinter->pszSpoolerParams = NULL;
      pdosPrinter->pszNetworkParams = NULL;
}
```

## PRINT.RC

```
#include <os2.h>
#include "print.h"

ACCELTABLE RES_CLIENT
{
   "^p", MI_PRINT, CHAR
   "^P", MI_PRINT, CHAR
   "^s", MI_SETUP, CHAR
   "^S", MI_SETUP, CHAR
   "^r", MI_REFRESH, CHAR
   "^R", MI_REFRESH, CHAR
   VK_F3, MI_EXIT, VIRTUALKEY
}

MENU RES_CLIENT
{
   SUBMENU "~Print", M_SAMPLE
   {
      MENUITEM "~Print sample\tCtrl P", MI_PRINT
      MENUITEM "Printer ~setup...\tCtrl S", MI_SETUP
      MENUITEM SEPARATOR
      MENUITEM "~Refresh list\tCtrl R", MI_REFRESH
   }
   SUBMENU "E~xit", M_EXIT
   {
      MENUITEM "E~xit sample\tF3", MI_EXIT
      MENUITEM "~Resume", MI_RESUME
   }
}
```

**PRINT.H**

```
#define RES_CLIENT                256
#define WND_LISTBOX               257
#define M_SAMPLE                  256
#define MI_PRINT                  257
#define MI_SETUP                  258
#define MI_REFRESH                259
#define M_EXIT                    260
#define MI_EXIT                   261
#define MI_RESUME                 262
```

**PRINT.MAK**

```
ICCOPTS=-C+ -Gm+ -Kb+ -Ss+
LINKOPTS=/MAP /A:16

PRINT.EXE:                        PRINT.OBJ \
                                   PRINT.RES
        LINK386 $(LINKOPTS) @<<
PRINT,
PRINT,
PRINT,
OS2386
PRINT
<<
        RC PRINT.RES PRINT.EXE

PRINT.RES:                        PRINT.RC \
                                   PRINT.H
        RC -r PRINT.RC PRINT.RES

PRINT.OBJ:                        PRINT.C \
                                   PRINT.H
        ICC $(ICCOPTS) PRINT.C
```

**PRINT.DEF**

```
NAME PRINT WINDOWAPI

DESCRIPTION 'Printing example
Copyright (c) 1992-1995 by Larry Salomon
All rights reserved.'

STACKSIZE 16384
```

This program illustrates the use of multithreading within a PM application. For more information, see Chapter 30.

Note that because many PM functions require the existence of a message queue, likely *WinInitialize* and *WinCreateMsgQueue* will have to be called. Also, it is usually good to call *WinCancelShutdown* so that PM will not send the thread a WM_QUIT message if the user should shut down the system while processing is still in progress.

The *extractPrnInfo*, *createPrnList*, and *destroyPrnList* functions were used in previous examples. *createPrnList* creates a DEVOPENSTRUC structure for each printer present and calls *extractPrnInfo* to initialize it. It also saves the device name for calls to *DevPostDeviceModes*.

*drawPage* is present only to separate the drawing from the print-job initialization (in *printThread*). Since it really does nothing, it could instead be placed directly in *printThread.* If an application does any complex drawing, it might be beneficial to keep the drawing separate so that (1) it allows reuse if the code between printing and repainting and (2) it does not clutter up the print-job handling. Mileage may vary.

*printThread* handles the creation of a queued device context and associated presentation space and the print-job creation. It calls *drawPage* to actually draw the output. Except for a few changes, it is the same code that was used earlier.

The client's window procedure (*clientWndProc*) provides the meat on the bones, so to speak. It utilitizes the window words to store a pointer to a structure containing any needed instance data. The instance data here contains the handle of a list box, to avoid using global variables instead. This list box, created in the WM_CREATE processing, contains the list of the printers defined for the system. It is resized in the WM_SIZE processing to match the size of the client, for maximum utilitization of "screen real estate."

Here we also see our first use of the WM_INITMENU message. This message is sent whenever the action bar or a pull-down menu is selected. This allows the application to disable menu items according to the state of the application at the time the menu was selected instead of trying to doing this on a per-action basis (i.e., the user selected item A on the menu, so immediately disable item B and enable item C). Taking a snapshot of the application often is much easier to do than figuring out state tables and all sorts of third-order differential equations just to see if the "Save" menu item should be selectable.

The WM_INITMENU has two parameters as well: SHORT1FROMMP(*mpParm1*) contains the resource ID of the menu that was selected, and HWNDFROMMP(*mpParm2*) contains the handle of the menu that was selected. The client checks to see if a printer is selected and if it contains any driver data and enables or disables the menu items as appropriate.

Of particular interest should be the processing of the menu items. MI_PRINT indicates that something should be printed, and this should take place asynchronously, so a PRNTHREADINFO structure is allocated and initialized with the handle of the client window and a pointer to the PRNLISTINFO structure for the selected window. A second thread finally is created using *_beginthread* and is passed the pointer to the PRNTHREADINFO structure. (This second thread is responsible for freeing the structure.)

MI_SETUP has practically nothing to do since everything was done already by *createPrnList.* It simply queries the PRNLISTINFO structure and calls *DevPostDeviceModes.*

MI_REFRESH simply calls *destroyPrnList* followed by *createPrnList.* This is needed in case the user adds a new printer after starting the application. Unfortunately, yet understandably, there is no way to be notified whenever the system configuration changes, so we have to force the user to select this menu item to update the list.

# Chapter 29

# Help Manager

Beginning with OS/2 1.2, IBM introduced an addition to the Presentation Manager interface (touted as the "Help Manager") that allowed an application to add both general help and field help online. (With 1.3, IBM published the previously undocumented method for creating online books, which are viewed using the system-supplied utility VIEW.EXE.) It should be noted, however, that while this capability is very appealing, it is by no means added to an application quickly; in fact, well-written online help can take on the average of one day per 3,000 lines of code to complete for the text alone. (This figure is based on personal experience.) The upside is that, for most Presentation Manager applications, programmers do not have to think about this designing the programs; online help can be added at any time, providing that the source code to the application is available.

## Application Components

There are at least three parts to the help component of any application: the source code, the HELPTABLEs, and the definitions of the help panels. The source code is obviously part of the application source, and includes the corresponding Win calls and HM_ messages sent to and received from the Help Manager. The HELPTABLEs (and HELPSUBTABLEs) are part of the resource file, and they define the relationships between the various windows and the corresponding help panels. Finally, the help panel definitions describe the look as well as the text of the help panels and are written using a *general markup language* (GML)–like language. (SCRIPT and Bookmaster users will recognize the help panel definition language as a subset of the Bookmaster macros they are familiar with.) Let us take a closer look at each of these three parts in more detail.

## The Application Source

The source code is usually the smallest component of the three, because it typically consists of an initialization section and the processing of a few messages. The initialization section normally goes in the *main* routine after the main window is created and follows the next which is the typical initialization code used in a Presentation Manager application to create a help instance.

```
#define HELP_CLIENT 256

HELPINIT hiInit;
CHAR achHelpTitle[256];
HAB habAnchor;
HWND hwndHelp;
HWND hwndFrame;

     : // WinInitialize, etc. goes here

// We need to initialize the HELPINIT structure before calling
// WinCreateHelpInstance.  See the online technical reference
```

```
// for an explanation of the individual fields.

hiInit.cb=sizeof(HELPINIT);
hiInit.ulReturnCode=0L;
hiInit.pszTutorialName=NULL;

// By specifying 0xFFFF in the high word of phtHelpTable, we are
// indicating that the help table is in the resource tables with
// the id specified in the low word.

hiInit.phtHelpTable=(PHELPTABLE)MAKEULONG(HELP_CLIENT,0xFFFF);

hiInit.hmodHelpTableModule=NULLHANDLE;
hiInit.hmodAccelActionBarModule=NULLHANDLE;
hiInit.idAccelTable=0;
hiInit.idActionBar=0;
hiInit.pszHelpWindowTitle=achHelpTitle;
hiInit.fShowPanelId=CMIC_HIDE_PANEL_ID;
hiInit.pszHelpLibraryName="MYAPPL.HLP";

hwndHelp=WinCreateHelpInstance(habAnchor,&hiInit);
if ((hwndHelp!=NULLHANDLE) && (hiInit.ulReturnCode!=0)) {
   WinDestroyHelpInstance(hwndHelp);
   hwndHelp=NULLHANDLE;
} /* endif */

     :
     :  // Message loop goes here
     :

if (hwndHelp!=(HWND)NULL) {
   WinDestroyHelpInstance(hwndHelp);
   hwndHelp=NULLHANDLE;
} /* endif */
```

As with the relationship between window classes and window instances, there exists a help manager class of which an instance can be created by calling *WinCreateHelpInstance*. This function can have one of three outcomes.

1. The call can complete successfully, and the return value is the handle of the help instance.
2. The function can complete partially, returning a help instance handle and specifying an error code in the *ulReturnCode* field.
3. The function can fail, returning NULL. Because of the subtle difference between 1 and 2, it is not sufficient simply to check the return value.

If the help instance is created successfully, it becomes the recipient of any messages that are sent and the originator of any messages that are sent to the active window.

Since a help instance is associated with a "root" window and all of its descendants, the programmer must indicate what the root window is. This is done using the *WinAssociateHelpInstance* function.

```
(BOOL)WinAssociateHelpInstance(HWND hwndHelp,
                               HWND hwndWindow);
```

Specifing a non-NULL value for *hwndHelp* indicates that this is the active window that should be used when determining which help panel to display. Specifying NULL for this parameter removes the current association between the help instance and the window specified. We will see how this is used shortly.

**Gotcha!**
Note that *WinAssociateHelpInstance* will not work if it is called within the WM_CREATE message of the window with which it is associated. *WinAssociateHelpInstance* needs a valid window handle, and when the WM_CREATE message is received, the window handle is not yet valid.

## Messages

The next piece of source code that will be used in most applications deals with the "Help" pull-down menu and "Help" push buttons. According to IBM's guidelines on developing a application user interface, there should exist on all menus a pull-down titled "Help" that contains the following four items:

- "Using help..."
- "General help..."
- "Keys help..."
- "Help index..."

There also can be an optional fifth item, labeled "Product information..." which displays an "About" box when selected. Fortunately, four messages can be sent to the Help Manager to process these four menu items. Each of them takes no parameters. They are listed in Table 29.1.

Table 29.1 Help Manager Display Messages

| Message | Description |
|---|---|
| HM_DISPLAY_HELP | Displays help on using online help. |
| HM_EXT_HELP | Displays the "extended" help for the current window. |
| HM_KEYS_HELP | Displays the keys help for the current window. |
| HM_HELP_INDEX | Displays the help index. |

Except for HM_KEYS_HELP, all that needs to be done is send the appropriate message to the help instance. Sending HM_KEYS_HELP results in the help instance sending the window a HM_QUERY_KEYS_HELP message back to determine which "keys help" panel to display. The panel resource ID should be returned by the programmer in response to this message.

The behavior of a "Help" push button is left somewhat up to the programmer. The official IBM line is that it should display field help—a panel that describes what the purpose is of the control containing the cursor. We follow this strategy in our applications; it results in the displaying of the extended help for the frame or dialog. To display this help for the frame, the programmer should define the push button with the BS_NOPOINTERFOCUS style to avoid receiving the input focus and should send the help instance a HM_DISPLAY_HELP message (this time with either the panel resource ID or the panel name in *mpParm1* and either HM_RESOURCEID or HM_PANELNAME in *mpParm2*) to display the help panel for the current control with the focus. To display this help for the dialog, the programmer simply needs to send an HM_EXT_HELP message to the help instance.

# The Help Tables

The help tables define the relationship between the control windows and the help panels to be displayed when the user requests help. Visualizing the help tables as a two-dimensional array of help panel Ids may make understanding what they are easier. The first index into this array is the ID of the window that has been associated with a help instance via *WinAssociateHelpInstance*; the second index is either a menu item ID or an ID of a child window that can receive the input focus. To understand how the help tables are used, we need to understand the sequence of events beginning with the user pressing F1 and the displaying of the help panel.

1. The user presses F1.
2. The help instance determines the ID of the window that it is currently associated with.
3. The HELPITEM for the given window ID is referenced and the appropriate HELPSUBTABLE is determined.
4. The menu item ID (or the ID of the window with the focus) is used to look up in the HELPSUBTABLE the ID of the help panel to display.
5. The help panel definition is retrieved from the compiled help file.
6. The help panel is displayed.

There are obviously many places where errors can occur; the most frequent one is when the menu item ID/child window ID is not in the HELPSUBTABLE. When this occurs, the owner window chain is searched (steps 3 to 6). If it is still not found, the parent window chain also is searched. If the ID has not been found after both searches, the current window is sent a HM_HELPSUBITEM_NOT_FOUND message, giving it the opportunity to remedy the situation (via a HM_DISPLAY_HELP message). The default action is to display the extended help for the current window.

When the ID is found in a HELPSUBTABLE but the panel definition does not exist, or when any other error occurs (with the exception of HELPSUBITEM not found, just described above and when the extended help panel cannot be determined), the application is sent an HM_ERROR message. This message contains an error code in the first parameter that describes the condition causing the error. The typical response to receiving this is to display a message and then disable the help manager by calling *WinDestroyHelpInstance*.

Given this logical view of the help tables, let us look at a sample definition in a resource file.

# Sample HELPTABLE

The following sample HELPTABLEs describe the online help panels that correspond to the child windows and menuitems in the application and its associated dialogs.

```
HELPTABLE HELP_CLIENT
{
        HELPITEM HELP_CLIENT, SUBHELP_CLIENT, EXTHELP_CLIENT
        HELPITEM DLG_OPEN, SUBHELP_OPEN, EXTHELP_OPEN
        HELPITEM DLG_PRODUCTINFO,
                SUBHELP_PRODUCTINFO, EXTHELP_PRODUCTINFO
}

HELPSUBTABLE SUBHELP_CLIENT
{
        HELPSUBITEM M_FILE, HELP_M_FILE
        HELPSUBITEM MI_NEW, HELP_MI_NEW
```

```
        HELPSUBITEM MI_OPEN, HELP_MI_OPEN
        HELPSUBITEM MI_SAVE, HELP_MI_SAVE
        HELPSUBITEM MI_CLOSE, HELP_MI_CLOSE
        HELPSUBITEM MI_EXIT, HELP_MI_EXIT
        HELPSUBITEM M_HELP, HELP_M_HELP
        HELPSUBITEM MI_USINGHELP, HELP_MI_USINGHELP
        HELPSUBITEM MI_GENERALHELP, HELP_MI_GENERALHELP
        HELPSUBITEM MI_KEYSHELP, HELP_MI_KEYSHELP
        HELPSUBITEM MI_HELPINDEX, HELP_MI_HELPINDEX
        HELPSUBITEM MI_PRODINFO, HELP_MI_PRODINFO
}

HELPSUBTABLE SUBHELP_SETOPTIONS
{
        HELPSUBITEM DOPEN_EF_FILENAME, HELP_DOPEN_EF_FILENAME
        HELPSUBITEM DLG_PB_OK, HELP_DLG_PB_OK
        HELPSUBITEM DLG_PB_CANCEL, HELP_DLG_PB_CANCEL
        HELPSUBITEM DLG_PB_HELP, HELP_DLG_PB_HELP
}

HELPSUBTABLE SUBHELP_PRODINFO
{
        HELPSUBITEM DLG_PB_CANCEL,  HELP_DLG_PB_CANCEL
        HELPSUBITEM DLG_PB_HELP, HELP_DLG_PB_HELP
}
```

As is clear from the sample, our application has two dialogs with online help. Their resource identifiers are DLG_OPEN and DLG_PRODUCTINFO. There are 12 child windows or menu items that belong to the client window. In each of the HELPSUBITEMS, the window ID is on the left and the corresponding help panel resource ID is on the right.

**Gotcha!**
If the resource ID specified in the *WinCreateStdWindow* call is different from that used as the resource ID of the HELPTABLE, the first parameter to the HELPITEM that refers to the main window should be the same as the HELPTABLE resource ID and not the ID for the frame resources.

## Message Boxes

When an application needs to give the user some information, one way it can do so is by using the *WinMessageBox* function. This displays a window containing application-specified title and text as well as an optional icon to the left and one or more predefined push buttons (e.g., "OK", "Yes", "Abort", etc.). It returns a constant that specifies the push button selected on the message box (e.g., MBID_OK, MBID_NO, MBID_RETRY, etc.).

As might be imagined, only so much can be said in a small dialog box. Often, what fits is enough for most users to figure out what the programmer is trying to say. However, it would be nice to provide another level of detail for those who would like more information (i.e., online help). The constant MB_HELP specifies that a "Help" push button is requested; this is the only button that does not cause the function to return. Unfortunately, since a message box doesn't have to have an application window as the owner (HWND_DESKTOP will work fine for *hwndOwner;* this could be used in, for example, a program that simply calls *WinMessageBox* with the command line for the message for CMD files), it cannot simply send

the owner a message saying that the help button was pressed. The system, therefore, provides two ways to display help for message boxes: using a help hook and using HELPTABLEs. We will look at the latter method later in the chapter.

## Fishing, Anyone?

A "hook" is a function that Presentation Manager calls whenever a certain event occurs. In a perverted way, we could look at it as subclassing the entire system, but instead of intercepting messages before the intended recipient receives them, the application intercepts "events." These events range from the "code page changed" event to the "DLL has been loaded with *WinLoadLibrary*" event and cover 16 different items. There is, of course, a "help requested" event as well, and it is this event that we are interested in.

Hooks are installed with *WinSetHook* and are released with *WinReleaseHook*. Both take the same parameters:

```
(BOOL)WinSetHook(HAB habAnchor,
                 HMQ hmqQueue,
                 USHORT usHookType,
                 PFN pfnHookProc,
                 HMODULE hmodProc);
```

*habAnchor* is the handle to the anchor block for the calling thread. *hmqQueue* is the handle of the queue for which events are to be monitored. If this is NULLHANDLE, events for the entire system are monitored; however, the hook function—since it will be called by different processes—must reside in a DLL so that PM can load the function when needed. *usHookType* is one of the HK_ constants specifying the event to be monitored. *pfnHookProc* is a pointer to the event monitoring function (the "hook"). *hmodProc* is a handle to the DLL containing the hook function or NULLHANDLE if *hmqQueue* is not NULLHANDLE and the hook function resides in the executable.

Each of the procedures for the different hook types takes different parameters and returns different values. Since we're interested in the HK_HELP hook, here is the prototype of the hook function:

```
(BOOL)pfnHookProc(HAB habAnchor,
                  SHORT sMode,
                  USHORT usTopic,
                  USHORT usSubTopic,
                  PRECTL prclPosition);
```

*habAnchor* is the handle to the anchor block of the thread for which the event occurred. *sMode* indicates the context in which help was requested and is a HLPM_ constant. *usTopic* and *usSubTopic* are dependent on the value of *sMode*.

Table 29.2 Hook Variables

| *sMode Is* | *usTopic Is* | *usSubTopic Is* |
|---|---|---|
| HLPM_FRAME | Identifier of the active frame window | Identifier of the window with the focus |
| HLPM_MENU | Identifier of the pull-down menu or FID_MENU if the action bar is selected | Identifier of the menu item or submenu item for which help was requested |
| HLPM_WINDOW | Identifier of the message box | Not used |

The help hook returns TRUE if the next hook in the help hook chain should not be called and FALSE if the next hook should be called. The typical function of the help hook when used in this context is to send the help instance a HM_DISPLAY_HELP message to display the specified help panel.

**Gotcha!**
Note that the documentation states that the help hook should be installed before the help instance is created. However, since *WinSetHook* installed the hook at the head of the hook chain, this information is backward. For this procedure to work properly, the call to *WinSetHook* should be placed after the call to *WinAssociateHelpInstance*.

Given the information in the *Gotcha*, the following question comes up: Since *WinAssociateHelpInstance* is called only after the frame window has been created successfully, how does an application provide message box help for the WM_CREATE message? The answer is to call *WinSetHook* after creating the help instance, calling *WinCreateStdWindow* to create the frame window, and then releasing the hook, associating the help instance, and resetting the hook with *WinReleaseHook*, *WinAssociateHelpInstance*, and *WinSetHook*, respectively.

**Gotcha!**
The header files in the Toolkit indicate that the parameters for the help hook are a SHORT and two USHORTs for 16-bit applications and a LONG and two ULONGs for 32-bit application. This is incorrect. The parameters are always a SHORT and two USHORTs.

## The Help Panels

Now that we've seen how easy the code and resource definitions are, it is time to tackle the most difficult (to do well) and time-consuming aspect of this development phase—writing the help panels. While the definition of the language is large, it is fairly easy to digest. We will look at only the rudiments of the language; the full language definition can be gleaned from the online document entitled "IPF Reference" that is included with the OS/2 Warp Programmer's Toolkit.

The help file (whose file extension is usually ".IPF") is compiled by the "Information Presentation Facility Compiler" (a.k.a. IPFC) to produce a ".HLP" file that is read by the Help Manager when *WinCreateHelpInstance* is called. The source file contains a collection of "tags," which begin with a colon (:), followed by the tag name, an optional set of attributes, and finally a period. Some tags also require a matching "end tag" (e.g., a "begin list" and "end list" tag), which have no attributes and whose name usually matches the beginning tag name preceded by an e (e.g., ":sl." and ":esl."). Table 29.3 presents common tags and their meanings.

Table 29.3 Common IPF Tags

| Tag | Meaning |
|---|---|
| :h1. through :h6. | Heading tag. Headings 1 to 3 also have an entry in the table of contents. |
| :p. | New paragraph.x |
| :fn. :efn. | Footnote and ending tag. |

| | |
|---|---|
| :hp1. through :hp9. | Emphasis tag. This requires the matching ending tag (:ehp1. through :ehp9.). |
| :link. | Hypertext link. |
| :sl. :esl. | Simple list and ending tag. |
| :ul. :eul. | Unordered list and ending tag. |
| :ol. :eol. | Ordered list and ending tag. |
| :li. | List item. Used between the list tags to describe the items in the list. |
| :dl. :edl. | Definition list and ending tag. Whereas the other lists consist of a single element, definition lists consist of a "data term" and "data definition" (:dt. and :dd., respectively). |
| :dt. :dd. | Data term and data definition tags. |
| :dthd. :ddhd. | Data term heading and data definition heading tags. Also, a few special tags are used only once in a help file. |
| :userdoc. :euserdoc. | Beginning and ending of the document. |
| :title. | The text to be placed in the title bar of the help panels. |

While most of these tags have attributes, the ones used most are the resource and ID attributes. The resource attribute allows programmers to assign a numerical value to a heading tag (e.g., ":h1 res=2048.Help panel"), and this is what the HELPSUBITEMs reference. The ID attribute allows programmers to assign an alphanumeric name for use in hypertext links (e.g., ":h2 id='MYPANEL'.Help panel"). The ID attribute can be used on both heading and footnote tags, while the resource attribute can be used only on heading tags. Heading IDs are referenced using the "refid" attribute of a hypertext link; a footnote is referenced also using the "refid" attribute of a ":fnref" (footnote reference) tag.

In addition to the tags, certain symbols that are translated into different values in other languages, not easily enterable using the keyboard, or used by IPF are defined. These are referenced by symbol name substitution, beginning with an ampersand (&), including the symbol name, and ending with a period. Table 29.4 lists some commonly used symbols.

Table 29.4 Commonly Used Symbols

| Symbol | Meaning |
|---|---|
| &amp. | Ampersand |
| &cdq. | Close double quote |
| &colon. | Colon |
| &csq. | Close single quote |
| &lbrk. | Left bracket |
| &odq. | Open double quote |
| &osq. | Open single quote |
| &rbrk. | Right bracket |
| &vbar. | Vertical bar |

A help panel begins with a heading 1, 2, or 3 tag and ends with either the next heading 1, 2, or 3 tag or the end of the document. Everything that is in between is shown when the panel is displayed. The next section presents a sample panel showing how some of the tags are used.

## Sample Help Panel

```
:h1 id='MYPANEL' res=1000.My help panel
:p.When adding online help to your PM application, the following steps must be
taken
:fnref refid='MYFN'.&colon.
:ul compact.
:li.Initialization code must be written
:li.Messages must be processed
:li.Resources must be defined
:li.Help panels must be compiled
:eul.
:fn id='MYFN'.
Footnotes start on an implied paragraph.
:efn.
```

Note that at least five tags are required in a valid IPF file—:userdoc., :title., :h1., :p., and :euserdoc., in that order.

## Putting It All Together

We now have enough information to write a simple application to illustrate the points made in this chapter. The application, HELP1, contains a few menu items and dialog boxes but otherwise does nothing. Its sole purpose is to allow the user to display the online help.

**HELP1.C**

```
#define  INCL_WINHELP
#define  INCL_WINHOOKS
#define  INCL_WINSYS
#include <os2.h>
#include "help1.h"
#define  CLS_CLIENT    "SampleClass"
HWND hwndHelp;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

BOOL EXPENTRY helpHook(HAB habAnchor,SHORT sMode,USHORT usTopic,
                       USHORT usSubTopic,PRECTL prclPos);

INT main(VOID)
{
   HAB              habAnchor;
   HMQ              hmqQueue;
   ULONG            ulFlags;
   HWND             hwndFrame;
   HELPINIT         hiInit;
   BOOL             bLoop;
   QMSG             qmMsg;

   habAnchor = WinInitialize(0);
   hmqQueue = WinCreateMsgQueue(habAnchor,
                               0);

   WinRegisterClass(habAnchor,
                    CLS_CLIENT,
                    clientWndProc,
                    0,
                    0);

   ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
      FCF_SHELLPOSITION|FCF_SYSMENU|FCF_MENU;
```

```
   hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &ulFlags,
                                  CLS_CLIENT,
                              "Help Manager Sample Application 1",

                                  0,
                                  NULLHANDLE,
                                  RES_CLIENT,
                                  NULL);

   hiInit.cb = sizeof(HELPINIT);
   hiInit.ulReturnCode = 0;
   hiInit.pszTutorialName = NULL;
   hiInit.phtHelpTable = (PHELPTABLE)MAKEULONG(HELP_CLIENT,
                                               0xFFFF);
   hiInit.hmodHelpTableModule = NULLHANDLE;
   hiInit.hmodAccelActionBarModule = NULLHANDLE;
   hiInit.idAccelTable = 0;
   hiInit.idActionBar = 0;
   hiInit.pszHelpWindowTitle = "Help Manager Sample Help File";
   hiInit.fShowPanelId = CMIC_HIDE_PANEL_ID;
   hiInit.pszHelpLibraryName = "HELP1.HLP";

   hwndHelp = WinCreateHelpInstance(habAnchor,
                                    &hiInit);

   if ((hwndHelp != NULLHANDLE) && (hiInit.ulReturnCode != 0))
   {
      WinDestroyHelpInstance(hwndHelp);
      hwndHelp = NULLHANDLE;
   }
   else
      if ((hwndHelp != NULLHANDLE))
      {
         WinAssociateHelpInstance(hwndHelp,
                                  hwndFrame);
      }                                /* endif                */
   WinSetHook(habAnchor,
              hmqQueue,
              HK_HELP,
              (PFN)helpHook,
              NULLHANDLE);

   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
                     0,
                     0);
   while (bLoop)
   {
      WinDispatchMsg(habAnchor,
                     &qmMsg);
      bLoop = WinGetMsg(habAnchor,
                        &qmMsg,
                        NULLHANDLE,
                        0,
                        0);
   }                                /* endwhile             */
   WinReleaseHook(habAnchor,
                  hmqQueue,
                  HK_HELP,
                  (PFN)helpHook,
                  NULLHANDLE);

   if (hwndHelp != NULLHANDLE)
   {
```

```
      WinAssociateHelpInstance(NULLHANDLE,
                               hwndFrame);
      WinDestroyHelpInstance(hwndHelp);
      hwndHelp = NULLHANDLE;
   }                                    /* endif                */
   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return ;
}

BOOL EXPENTRY helpHook(HAB habAnchor,SHORT sMode,USHORT usTopic,
                       USHORT usSubTopic,PRECTL prclPos)
{
   if ((sMode == HLPM_WINDOW) && (hwndHelp != NULLHANDLE))
   {
      WinSendMsg(hwndHelp,
                 HM_DISPLAY_HELP,
                 MPFROMLONG(MAKELONG(usTopic,
                                     0)),
                 MPFROMSHORT(HM_RESOURCEID));
      return  TRUE;
   }

   else
   {
      return  FALSE;
   }                                    /* endif                */
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {

      case  WM_PAINT :
         {
            HPS              hpsPaint;
            RECTL            rclPaint;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     &rclPaint);
            WinFillRect(hpsPaint,
                        &rclPaint,
                        SYSCLR_WINDOW);
            WinEndPaint(hpsPaint);
         }
         break;
      case  WM_COMMAND :
         switch (SHORT1FROMMP(mpParm1))
         {
            case  MI_HELPINDEX :
               WinSendMsg(hwndHelp,
                          HM_HELP_INDEX,
                          0,
                          0);
               break;
            case  MI_GENERALHELP :
               WinSendMsg(hwndHelp,
                          HM_EXT_HELP,
                          0,
                          0);
               break;
            case  MI_HELPFORHELP :
```

```
                WinSendMsg(hwndHelp,
                           HM_DISPLAY_HELP,
                           0,
                           0);
                break;
             case  MI_KEYSHELP :
                WinSendMsg(hwndHelp,
                           HM_KEYS_HELP,
                           0,
                           0);
                break;
             case  MI_PRODINFO :
                WinMessageBox(HWND_DESKTOP,
                              hwndClient,
                           "Copyright 1995 by Larry Salomon,Jr.",

                              "Help Sample",
                              HLP_MESSAGEBOX,
                              MB_OK|MB_HELP|MB_INFORMATION);
                break;
             default  :
                return  WinDefWindowProc(hwndClient,
                                         ulMsg,
                                         mpParm1,
                                         mpParm2);
          }                              /* endswitch            */
          break;

       case  HM_QUERY_KEYS_HELP :
          return  MRFROMSHORT(KEYSHELP_CLIENT);
       default  :
          return  WinDefWindowProc(hwndClient,
                                   ulMsg,
                                   mpParm1,
                                   mpParm2);
    }                                   /* endswitch            */
    return  MRFROMSHORT(FALSE);
}
```

**HELP1.RC**

```
#include <os2.h>
#include "help1.h"

MENU RES_CLIENT
{
   SUBMENU "~Help", M_HELP
   {
      MENUITEM "Help ~index...", MI_HELPINDEX
      MENUITEM "~General help...", MI_GENERALHELP
      MENUITEM "~Using help...", MI_HELPFORHELP
      MENUITEM "~Keys help...", MI_KEYSHELP
      MENUITEM SEPARATOR
      MENUITEM "~Product information...", MI_PRODINFO
   }
}

HELPTABLE HELP_CLIENT
{
   HELPITEM RES_CLIENT, SUBHELP_CLIENT, EXTHELP_CLIENT
}
```

```
HELPSUBTABLE SUBHELP_CLIENT
{
   HELPSUBITEM MI_HELPINDEX, MI_HELPINDEX
   HELPSUBITEM MI_GENERALHELP, MI_GENERALHELP
   HELPSUBITEM MI_HELPFORHELP, MI_HELPFORHELP
   HELPSUBITEM MI_KEYSHELP, MI_KEYSHELP
   HELPSUBITEM MI_PRODINFO, MI_PRODINFO
}
```

## HELP1.H

```
#define RES_CLIENT              256
#define HELP_CLIENT             257
#define SUBHELP_CLIENT          258
#define EXTHELP_CLIENT          259
#define KEYSHELP_CLIENT         260
#define M_HELP                  320
#define MI_HELPINDEX            321
#define MI_GENERALHELP          322
#define MI_HELPFORHELP          323
#define MI_KEYSHELP             324
#define MI_PRODINFO             325
#define HLP_MESSAGEBOX          326
```

## HELP1.MAK

```
ICCOPTS=-C+ -Gm- Kb+ -Ss+
LINKOPTS=/MAP /A:16

HELP1.EXE:                      HELP1.OBJ \
                                HELP1.RES \
                                HELP1.HLP
        LINK386 $(LINKOPTS) @<<
HELP1
HELP1
HELP1
OS2386
HELP1
<<
        RC HELP1.RES HELP1.EXE

HELP1.OBJ:                      HELP1.C \
                                HELP1.H
        ICC $(ICCOPTS) HELP1.C

HELP1.RES:                      HELP1.RC \
                                HELP1.H
        RC -r HELP1.RC HELP1.RES

HELP1.HLP:                      HELP1.IPF
        IPFC HELP1.IPF
```

**HELP1.DEF**

```
;-----------------------------------------
; HELP1.DEF
;-----------------------------------------
NAME HELP1 WINDOWAPI

DESCRIPTION 'Help Example 1
Copyright 1995 by Larry Salomon.
All rights reserved.'

STACKSIZE 32768
```

**HELP1.IPF**

```
:userdoc.
:title.Help Manager Sample Help File
:h1 res=259.Extended help
:p.Normally, you would write a longer panel here describing an overview of the
function of the active window.  Diagrams, etc. are certainly welcome, since
this is usually called when the user has no idea of what is going on at the
moment.
:h1 res=260.Keys help
:p.A list of the accelerator keys in use is appropriate here.  Do not forget
the &odq.hidden&cdq. accelerators in dialog boxes and elsewhere such as
:hp2.enter:ehp2., :hp2.escape:ehp2., and :hp2.F1:ehp2..
:h1 res=321.Help for menuitem &odq.Help index...&cdq.
:p.Selecting this menuitem will display a help index.  Note that the system
has its own help index, while we can add our own entries using the &colon.i1.
and &colon.i2. tags.
:h1 res=322.Help for menuitem &odq.General help...&cdq.
:p.Selecting this menuitem will display the general help panel.
:h1 res=323.Help for menuitem &odq.Using help...&cdq.
:p.Selecting this menuitem will display the system help panel on how to use
the Help Manager.
:h1 res=324.Help for menuitem &odq.Keys help...&cdq.
:p.Selecting this menuitem will display the active key list.
:h1 res=325.Help for menuitem &odq.Product information...&cdq.
:p.Selecting this menuitem will display a message box with a help button in
it, to demonstrate the use of a help hook to provide message box help.
:h1 res=326.Message box help
:p.This application demonstrates the use of the Help Manager to provide online
help for an application.  Help for both menuitems and message boxes is shown.
:euserdoc.
```

# Restrictions

While the HELPSUBITEMs use manifest constants (those that have been #define'd), the help panels must hard-code their resource IDs. This is a nasty problem that can be solved only by a preprocessor that can accept C include files; a few good public domain ones are available for only the cost of connecting to CompuServe or the Internet.

# Using HELPTABLEs for Message Box Help

As we stated earlier, there are two methods for providing help for message boxes. The second method uses HELPTABLEs defined in your .RC file to specify the appropriate help panels to display. This method has advantages and disadvantages:

Advantages

- Hooks are "performance-eaters," so help-related activity will execute faster using HELPTABLEs

- Using HELPTABLEs is less confusing which makes them easier to code

Disadvantages

- Each message box requires a HELPITEM in the HELPTABLE. For large applications, your HELPTABLE can quickly get unwieldy using HELPTABLEs.

The method is simple to implement and can be broken down into the following steps, which must be done *for every message box that your application displays*:

1) Add a HELPITEM in the HELPTABLE with the first parameter having the value of the identifier specified as the fifth parameter to *WinMessageBox*. The second parameter of the HELPITEM specifies a HELPSUBTABLE for the message box and the third specifies the resource id of the "General help" panel for the message box.
2) Add a HELPSUBTABLE whose identifier is the same as that of the second parameter of the HELPITEM created in step 1.
3) Add one HELPSUBITEM in the HELPSUBTABLE created in step 2 for each button displayed in the message box (specified using the MB_ constants). The first parameter of the HELPSUBITEM is the MBID_ constant corresponding to the message box button, and the second parameter is—as you would guess— the resource id of the help panel corresponding to the message box button.

That's all you need to do. Let me reiterate, however, that this must be done *for every message box your application displays*.

**Gotcha!**
If a HELPSUBTABLE corresponding to a message box is empty, nothing will happen when the user presses F1 on that message box. No error message can be sent because, as we stated earlier, the message box does not necessarily know to whom the message should be sent.

**HELP2.C**

```
#define  INCL_WINHELP
#define  INCL_WINHOOKS
#define  INCL_WINSYS
#include <os2.h>
#include "help2.h"
#define  CLS_CLIENT     "SampleClass"
HWND hwndHelp;

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2);

BOOL EXPENTRY helpHook(HAB habAnchor,SHORT sMode,USHORT usTopic,
                       USHORT usSubTopic,PRECTL prclPos);

INT main(VOID)
{
   HAB               habAnchor;
   HMQ               hmqQueue;
   ULONG             ulFlags;
   HWND              hwndFrame;
   HELPINIT          hiInit;
   BOOL              bLoop;
   QMSG              qmMsg;
```

```
habAnchor = WinInitialize(0);
hmqQueue = WinCreateMsgQueue(habAnchor,
                             0);

WinRegisterClass(habAnchor,
                 CLS_CLIENT,
                 clientWndProc,
                 0,
                 0);

ulFlags = FCF_SIZEBORDER|FCF_TITLEBAR|FCF_TASKLIST|
   FCF_SHELLPOSITION|FCF_SYSMENU|FCF_MENU;

hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
                               WS_VISIBLE,
                               &ulFlags,
                               CLS_CLIENT,
                     "Help Manager Sample Application 2",

                               0,
                               NULLHANDLE,
                               RES_CLIENT,
                               NULL);

hiInit.cb = sizeof(HELPINIT);
hiInit.ulReturnCode = 0;
hiInit.pszTutorialName = NULL;
hiInit.phtHelpTable = (PHELPTABLE)MAKEULONG(HELP_CLIENT,
                                            0xFFFF);
hiInit.hmodHelpTableModule = NULLHANDLE;
hiInit.hmodAccelActionBarModule = NULLHANDLE;
hiInit.idAccelTable = 0;
hiInit.idActionBar = 0;
hiInit.pszHelpWindowTitle = "Help Manager Sample Help File";
hiInit.fShowPanelId = CMIC_HIDE_PANEL_ID;
hiInit.pszHelpLibraryName = "HELP2.HLP";

hwndHelp = WinCreateHelpInstance(habAnchor,
                                 &hiInit);

if ((hwndHelp != NULLHANDLE) && (hiInit.ulReturnCode != 0))
{
   WinDestroyHelpInstance(hwndHelp);
   hwndHelp = NULLHANDLE;
}
else
   if ((hwndHelp != NULLHANDLE))
   {
      WinAssociateHelpInstance(hwndHelp,
                               hwndFrame);
   }                                 /* endif                */
bLoop = WinGetMsg(habAnchor,
                  &qmMsg,
                  NULLHANDLE,
                  0,
                  0);
while (bLoop)
{
   WinDispatchMsg(habAnchor,
                  &qmMsg);
   bLoop = WinGetMsg(habAnchor,
                     &qmMsg,
                     NULLHANDLE,
                     0,
                     0);
}                                    /* endwhile             */
```

```
   if (hwndHelp != NULLHANDLE)
   {
      WinAssociateHelpInstance(NULLHANDLE,
                               hwndFrame);
      WinDestroyHelpInstance(hwndHelp);
      hwndHelp = NULLHANDLE;
   }                                   /* endif                */
   WinDestroyWindow(hwndFrame);
   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return ;
}

MRESULT EXPENTRY clientWndProc(HWND hwndClient,ULONG ulMsg,MPARAM
                               mpParm1,MPARAM mpParm2)
{
   switch (ulMsg)
   {
      case  WM_PAINT :
         {
            HPS              hpsPaint;
            RECTL            rclPaint;

            hpsPaint = WinBeginPaint(hwndClient,
                                     NULLHANDLE,
                                     &rclPaint);
            WinFillRect(hpsPaint,
                        &rclPaint,
                        SYSCLR_WINDOW);
            WinEndPaint(hpsPaint);
         }
         break;
      case  WM_COMMAND :
         switch (SHORT1FROMMP(mpParm1))
         {
            case  MI_HELPINDEX :
               WinSendMsg(hwndHelp,
                          HM_HELP_INDEX,
                          0,
                          0);
               break;
            case  MI_GENERALHELP :
               WinSendMsg(hwndHelp,
                          HM_EXT_HELP,
                          0,
                          0);
               break;
            case  MI_HELPFORHELP :
               WinSendMsg(hwndHelp,
                          HM_DISPLAY_HELP,
                          0,
                          0);
               break;
            case  MI_KEYSHELP :
               WinSendMsg(hwndHelp,
                          HM_KEYS_HELP,
                          0,
                          0);
               break;
            case  MI_PRODINFO :
               WinMessageBox(HWND_DESKTOP,
                             hwndClient,
                      "Copyright 1995 by Larry Salomon,Jr.",

                             "Help Sample",
                             HLP_MESSAGEBOX,
                             MB_OK|MB_HELP|MB_INFORMATION);
```

```
            break;
         default  :
            return  WinDefWindowProc(hwndClient,
                                     ulMsg,
                                     mpParm1,
                                     mpParm2);
      }                              /* endswitch            */
      break;

   case  HM_QUERY_KEYS_HELP :
      return  MRFROMSHORT(KEYSHELP_CLIENT);
   default  :
      return  WinDefWindowProc(hwndClient,
                               ulMsg,
                               mpParm1,
                               mpParm2);
}                                   /* endswitch            */
return  MRFROMSHORT(FALSE);
}
```

**HELP2.RC**

```
#include <os2.h>
#include "help2.h"

MENU RES_CLIENT
{
   SUBMENU "~Help", M_HELP
   {
      MENUITEM "Help ~index...", MI_HELPINDEX
      MENUITEM "~General help...", MI_GENERALHELP
      MENUITEM "~Using help...", MI_HELPFORHELP
      MENUITEM "~Keys help...", MI_KEYSHELP
      MENUITEM SEPARATOR
      MENUITEM "~Product information...", MI_PRODINFO
   }
}

HELPTABLE HELP_CLIENT
{
   HELPITEM RES_CLIENT, SUBHELP_CLIENT, EXTHELP_CLIENT
   HELPITEM HLP_MESSAGEBOX, SUBHELP_MESSAGEBOX, EXTHELP_MESSAGEBOX
}

HELPSUBTABLE SUBHELP_CLIENT
{
   HELPSUBITEM MI_HELPINDEX, MI_HELPINDEX
   HELPSUBITEM MI_GENERALHELP, MI_GENERALHELP
   HELPSUBITEM MI_HELPFORHELP, MI_HELPFORHELP
   HELPSUBITEM MI_KEYSHELP, MI_KEYSHELP
   HELPSUBITEM MI_PRODINFO, MI_PRODINFO
}

HELPSUBTABLE SUBHELP_MESSAGEBOX
{
;   HELPSUBITEM MBID_OK, HLPPNL_OK
;   HELPSUBITEM MBID_HELP, HLPPNL_HELP
}
```

**HELP2.H**

```
#define RES_CLIENT              256
#define HELP_CLIENT             257
#define SUBHELP_CLIENT          258
```

```
#define EXTHELP_CLIENT          259
#define KEYSHELP_CLIENT         260
#define SUBHELP_MESSAGEBOX      261
#define EXTHELP_MESSAGEBOX      262
#define M_HELP                  320
#define MI_HELPINDEX            321
#define MI_GENERALHELP          322
#define MI_HELPFORHELP          323
#define MI_KEYSHELP             324
#define MI_PRODINFO             325
#define HLP_MESSAGEBOX          326
#define HLPPNL_OK               327
#define HLPPNL_HELP             328
```

**HELP2.MAK**

```
ICCOPTS=-C+ -Gm- Kb+ -Ss+
LINKOPTS=/MAP /A:16

HELP2.EXE:                      HELP2.OBJ \
                                HELP2.RES \
                                HELP2.HLP
        LINK386 $(LINKOPTS) @<<
HELP2
HELP2
HELP2
OS2386
HELP2
<<
        RC HELP2.RES HELP2.EXE

HELP2.OBJ:                      HELP2.C \
                                HELP2.H
        ICC $(ICCOPTS) HELP2.C

HELP2.RES:                      HELP2.RC \
                                HELP2.H
        RC -r HELP2.RC HELP2.RES

HELP2.HLP:                      HELP2.IPF
        IPFC HELP2.IPF
```

**HELP2.DEF**

```
;-------------------------------------------
; HELP2.DEF
;-------------------------------------------
NAME HELP2 WINDOWAPI

DESCRIPTION 'Help Example 2
Copyright 1995 by Larry Salomon.
All rights reserved.'

STACKSIZE 32768
```

**HELP2.IPF**

```
:userdoc.
:title.Help Manager Sample Help File
:h1 res=259.Extended help
:p.Normally, you would write a longer panel here describing an overview of the
function of the active window.  Diagrams, etc. are certainly welcome, since
this is usually called when the user has no idea of what is going on at the
```

```
moment.
:h1 res=260.Keys help
:p.A list of the accelerator keys in use is appropriate here.  Do not forget
the &odq.hidden&cdq. accelerators in dialog boxes and elsewhere such as
:hp2.enter:ehp2., :hp2.escape:ehp2., and :hp2.F1:ehp2..
:h1 res=262.Help for message box
:p.This application demonstrates the use of the Help Manager to provide online
help for an application.  Help for both menuitems and message boxes is shown.
:h1 res=321.Help for menuitem &odq.Help index...&cdq.
:p.Selecting this menuitem will display a help index.  Note that the system
has its own help index, while we can add our own entries using the &colon.i1.
and &colon.i2. tags.
:h1 res=322.Help for menuitem &odq.General help...&cdq.
:p.Selecting this menuitem will display the general help panel.
:h1 res=323.Help for menuitem &odq.Using help...&cdq.
:p.Selecting this menuitem will display the system help panel on how to use
the Help Manager.
:h1 res=324.Help for menuitem &odq.Keys help...&cdq.
:p.Selecting this menuitem will display the active key list.
:h1 res=325.Help for menuitem &odq.Product information...&cdq.
:p.Selecting this menuitem will display a message box with a help button in
it, to demonstrate the use of a help hook to provide message box help.
:h1 res=327.Help for message box &odq.Ok&cdq. button
:p.Selecting this dismisses the message box.
:h1 res=328.Help for message box &odq.Help&cdq. button
:p.Selecting this displays the appropriate help.
:euserdoc.
```

# Chapter 30

# Multithreading in Presentation Manager Applications

## Introduction

Because of what is often perceived as a design flaw in Presentation Manager, tasks that require more time than is suggested by IBM's "well-behaved" application guideline should be performed in a thread separate from that which contains the message dispatch loop (denoted by the calls to *WinGetMsg* and *WinDispatchMsg*). However, the issue of communication between the user interface and additional threads created by the user interface arises because there is no recommended design to follow. This chapter attempts to design an architecture that is easy to implement yet expandable and requires no global variables (always a good thing).

Before we can begin to explore this topic, we need to know exactly when should it be used—what exactly is a "well-behaved" application? When Presentation Manager was introduced in OS/2 1.1, IBM defined this to be an application that does not take longer than one-tenth of a second to process each message and return to the message loop. Multithreading lets us avoid this is by creating separate threads for the various tasks that will take (significantly) longer to complete.

Throughout the years, every conceivable technique has been tried to accomplish mulithreading in a smooth fashion. The solution presented herein seems to be good for most actions requiring the user to initiate a task that requires the additional thread. It should be stressed, however, that mileage may vary and that this may not work as well for programmers and their design "methodologies." This chapter should be used as a starting point and not as the final result.

For the curious, the reason for this one-tenth of a second rule involves changing the input focus from one window to another. Developers at IBM decided that, for backward compatibility, *type-ahead* should be included as a feature in Presentation Manager. Because of the resulting design, all input from the keyboard and mouse first goes into a *system input queue*; it gets moved to the queue of the window with the input focus whenever *WinGetMsg* is called.

Whenever *WinDispatchMsg* routes a message to a window procedure, the function does not return until the window procedure finishes processing the message; this means that the *WinGetMsg* function is not called, which ultimately results in the input not being rerouted from the system queue to the application queue. To a user, if PM appears "hung"—if he or she tries to change the input focus by clicking with the mouse on another window, nothing will happen because *WinGetMsg* is not being called regularly.

Fortunately, PM has a "watchdog" thread that monitors the rate at which input messages are removed from the system queue. If none is removed before a certain time has elapsed, the infamous "the application is not responding to system messages" window is displayed, allowing the user to terminate the offending application. OS/2 Warp has a new option in the *System* notebook of the *System Setup* folder to disable type-ahead; while this may be a workaround (its effectiveness has yet to be fully tested), this chapter still is relevant because this setting may or may not be in effect.

# Types of Threads

With the brouhaha about client/server programming everywhere we look, it could appear that this is the only multithreading application. However, a quick reality check reveals that many common user-initiated operations can be performed in a separate thread. Examples of this include file loading and saving, printing, and even window initialization (if it takes awhile to finish). What makes these tasks different from others is that, once the specifics have been collected from the user (if necessary), the processing can be performed without further user intervention. Threads that perform the tasks are dubbed *one-shot threads* because they are created as needed and are destroyed once they are no longer needed. We will concentrate on these, since they are one of the more common uses of multithreading.

Consider the following list of events.

1. The user selects "Open..." from the menu.
2. The application is notified of this selection.
3. The application prompts the user for a filename.
4. The application reads the selected file.
5. The user is then allowed to perform operations on the file's data.

As can be seen, these one-shot threads have a specific purpose and usually are accompanied by user input (e.g., filename to load); thus, communication quickly becomes an issue to be considered. The easy way out is to use global variables to hold data, but this is inadequate because of synchronization issues and more important, because the number of threads that perform a specific task must be limited to the number of global variables defined to hold the data resulting from the operations. Thus only two choices are left: local (automatic) variables and dynamic allocation. Because we cannot exceed the one-tenth second in our window procedure we will quickly discard the option of using local variables.

Assuming dynamic allocation is the solution to use, how is data communicated to the thread, and how does the thread return the results to the user interface?

# Designing the Architecture

Since the quality of the solution to any nontrivial problem is dependent on the quality of the design, we will take a look at this first. There are three defined areas of interest: data communications, entry and exitpoints, and user feedback.

*Data communications* involves passing parameters to the thread and receiving results from the thread. *Entry* and *exit points* provide a consistent interface to the programmer, to ease the coding necessary to start a thread (including data communications) and to allow the easy addition of new one-shot thread types. *User feedback* is less an issue of the threading but more an issue of communicating to the user that processing is being performed in the background.

# Data Communications

Although *data communications* is more likely to be associated with *interprocess communication,* the latter is unnecessarily complex, because the two ends of the communcations line are not always in the same process. Because threads always belong to the same process, we can simplify things considerably by (carefully) using pointers instead of shared memory, queues, or pipes to communicate our intentions. Even though most compilers provide a runtime function to start a thread and set up the run-time environment so that the new thread also can call the C runtime library, they are all constrained by the *DosCreateThread* function to passing a single argument to the new thread; this limits us to one pointer for all data, which immediately forces us to use structures to pass things back and forth.

Experience shows that most threads require a common set of information, encapsulated in a *THREADINFO* structure.

```
typedef struct _THREADINFO {
   ULONG ulSzStruct;
   HWND hwndOwner;
   BOOL bKillThread;
   HAB habThread;
   BOOL bThreadDead;
   BOOL bResult;
} THREADINFO, *PTHREADINFO;
```

*ulSzStruct* specifies the size of the structure. *hwndOwner* specifies the handle of the window that created the thread. *bKillThread* is set to TRUE by the owner when the request is to be aborted. *habThread* specifies the anchor block handle of the thread. (Readers should keep reading to see why this is necessary.) *bThreadDead* is set to TRUE by the thread when it is dead. (Again, readers should keep reading to see why this is necessary.) *bResult* is a blanket indicator of the success or failure of the task.

Since we said that this information is common to most threads and not specific to a particular task, it can be deduced that the task-specific data is encapsulated in another structure, with a *THREADINFO* structure as one of the fields. In fact, the *THREADINFO* structure should always be the first field, so that any task-independent code can safely typecast any task-specific structure pointer to access the common fields.

```
typedef struct _OPENTHREADINFO {
   THREADINFO tiCommon;
   CHAR achFilename[CCHMAXPATH];
   PFILEDATA pfdData;
} OPENTHREADINFO, *POPENTHREADINFO;
```

# Entry and Exit Points

As explained earlier, one-shot threads are created as the result of a user action, usually from a menu item. Because the context of an action (*Open file*, for example) is dependent on the window that was active when the action was requested, it makes sense to say that the one-shot thread belongs to the active window. Since one window class might support many different thread types, a common entry and exit point for all asynchronous tasks can save a lot of typing. Windows primarily communicate using messages, so we will introduce two user messages to be used as these entry and exit points.

**MYM_STARTTHREAD** This message is sent by a window to create a thread to perform a user-initiated request.

| | | |
|---|---|---|
| **Parameter 1:** | *ULONG ulType* | ID of thread type to be created |
| **Parameter 2:** | *PVOID pvData* | Pointer to task-specific data |
| **Reply:** | *BOOL bSuccess* | Successful? TRUE: FALSE |

**MYM_ENDTHREAD** This message is sent by a thread to indicate that processing has completed.

| | | |
|---|---|---|
| **Parameter 1:** | *ULONG ulType* | ID of thread type sending the message |
| **Parameter 2:** | *PVOID pvData* | Pointer to task-specific data |
| **Reply:** | *ULONG ulReserved* | Reserved, 0 |

*pvData* in both messages points to the task-specific data discussed in the last section. Because there is more to the data than the common information, we need to specify the type of the thread being created; the type identifiers have a one-to-one correspondence to the task-specific structures that also are created. *ulType* allows us to *switch* on this value to access the task-specific portion of each thread type. We will see later that each thread type identifier should occupy a unique bit in the 32 available.

MYM_STARTTHREAD first initializes the common portion of the structure, allocates enough memory from the heap (based on the value of *ulType*) to hold a copy of the structure, and copies *pvData* to this new memory block. After this, the thread is created and passed the pointer to the new memory block as the parameter. Any task-specific fields should be initialized prior to sending this message.

Not all of the fields of the THREADINFO structure can be initialized by the MYM_STARTTHREAD message. In particular, the *habThread*, *bThreadDead*, and *bResult* fields can be initialized only by the thread.

```
#define MYM_STARTTHREAD     (WM_USER)
#define MYM_ENDTHREAD       (WM_USER+1)

#define ASYNC_OPEN          0x00000001L

typedef VOID (_Optlink PFNREQ)(PVOID);

  :
  :

case MYM_STARTTHREAD:
   {
      ULONG ulBit;
      PTHREADINFO ptiInput;
      PFNREQ pfnThread;
      PVOID pvParm;

      ulBit=LONGFROMMR(mpParm1);
      ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);

      ptiInput->hwndOwner=hwndWnd;
      ptiInput->bKillThread=FALSE;

      switch (ulBit) {
      case ASYNC_OPEN:
         {
```

```
            POPENTHREADINFO potiInfo;

            ptiInput->ulSzStruct=sizeof(OPENTHREADINFO);

            potiInfo=(POPENTHREADINFO)malloc(
                sizeof(OPENTHREADINFO));

            if (potiInfo==NULL) {
               WinMessageBox(HWND_DESKTOP,
                             hwndWnd,
                             "There is not enough memory.",
                             "Error",
                             0,
                             MB_OK|MB_ICONEXCLAMATION|
                                MB_MOVEABLE);
               return MRFROMSHORT(FALSE);
            } /* endif */

            memcpy(potiInfo,
                   ptiInput,
                   sizeof(OPENTHREADINFO));

            pfnThread=(PFNREQ)openThread;
            pvParm=(PVOID)potiInfo;
         }
         break;
      default:
         WinMessageBox(HWND_DESKTOP,
                       hwndWnd,
                       "There is an internal error.",
                       "Error",
                       0,
                       MB_OK|MB_ICONEXCLAMATION|
                          MB_MOVEABLE);
         return MRFROMSHORT(FALSE);
      } /* endswitch */

      if (_beginthread(pfnThread,NULL,0x4000,pvParm)==-1) {
         free(pvParm);
         WinMessageBox(HWND_DESKTOP,
                       hwndWnd,
                       "The thread could not be created.",
                       "Error",
                       0,
                       MB_OK|MB_ICONEXCLAMATION|
                          MB_MOVEABLE);
         return MRFROMSHORT(FALSE);
      } /* endif */
   }
   break;
```

Note the need for the PFNREQ type. If we do not use this, then we will not be able to use the *pfnThread* variable; more important, we will receive compiler warnings on the call to *_beginthread.*

MYM_ENDTHREAD waits for the thread to die, using the *bThreadDead* field of the THREADINFO structure as its cue. Afterward, it uses the value of *ulType* to check the return information (or it could use the *bResult* field of the THREADINFO structure for a quick-check). Finally, it performs any processing necessary to allow the application to continue and then frees the memory allocated for *pvData* in MYM_STARTTHREAD.

```
#define MYM_STARTTHREAD     (WM_USER)
#define MYM_ENDTHREAD       (WM_USER+1)
```

```
#define ASYNC_OPEN          0x00000001L

  :
  :

case MYM_ENDTHREAD:
   {
      ULONG ulBit;
      PTHREADINFO ptiInput;

      ulBit=LONGFROMMR(mpParm1);
      ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);

      while (!ptiInput->bThreadDead) {
         DosSleep(1);
      } /* endwhile */

      switch (ulBit) {
      case ASYNC_OPEN:
         {
            POPENTHREADINFO potiInfo;

            potiInfo=(POPENTHREADINFO)ptiInput;
            free(potiInfo);
         }
         break;
      default:
         return MRFROMSHORT(FALSE);
      } /* endswitch */
   }
   break;
```

Programmers who think about it for a second will undoubtedly question the use of *DosSleep* in the preceding code. Isn't multithreading used in PM programs so that the message loop is always returned to in one-tenth of a second? Yes, it is; however, as we will see in the thread termination processing, this message is not sent until just before the thread dies, so the *while* loop will be executed a few times at most. Thus, the *DosSleep* call and the entire loop is rather harmless in this situation.

# What Have We So Far?

Let's take a look at an example that illustrates the concepts described up to this point.

**THRD1.C**

```
#define INCL_DOSPROCESS
#define INCL_WININPUT
#define INCL_WINMENUS
#define INCL_WINPOINTERS
#define INCL_WINSYS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include "thrd1rc.h"

#define CLS_MAIN                "Thread1Class"

#define ASYNC_TEST              0x00000001L

typedef VOID (* _Optlink PFNREQ)(PVOID);

#define MYM_BASE                (WM_USER)
#define MYM_STARTTHREAD         (MYM_BASE)
#define MYM_ENDTHREAD           (MYM_BASE+1)
```

```
typedef struct _THREADINFO {
   //-------------------------------------------------
   // Initialized by the main thread
   //-------------------------------------------------
   ULONG ulSzStruct;
   HWND hwndOwner;
   BOOL bKillThread;
   //-------------------------------------------------
   // Initialized by the secondary thread
   //-------------------------------------------------
   HAB habThread;
   BOOL bThreadDead;
   BOOL bResult;
} THREADINFO, *PTHREADINFO;

typedef struct _TESTTHREADINFO {
   THREADINFO tiInfo;
} TESTTHREADINFO, *PTESTTHREADINFO;

typedef struct _INSTDATA {
   ULONG ulSzStruct;
   HWND hwndMenu;
} INSTDATA, *PINSTDATA;

VOID _Optlink testThread(PTESTTHREADINFO pttiInfo)
//-------------------------------------------------
// This is a do nothing thread that beeps, sleeps,
// and beeps again
//-------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   pttiInfo->tiInfo.habThread=habAnchor;
   pttiInfo->tiInfo.bThreadDead=FALSE;
   pttiInfo->tiInfo.bResult=FALSE;

   WinAlarm(HWND_DESKTOP,WA_NOTE);
   DosSleep(2000);
   WinAlarm(HWND_DESKTOP,WA_NOTE);

   WinPostMsg(pttiInfo->tiInfo.hwndOwner,
              MYM_ENDTHREAD,
              MPFROMLONG(ASYNC_TEST),
              MPFROMP(pttiInfo));

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);

   DosEnterCritSec();
   pttiInfo->tiInfo.bThreadDead=TRUE;
   return;
}

MRESULT EXPENTRY wndProc(HWND hwndWnd,
                         ULONG ulMsg,
                         MPARAM mpParm1,
                         MPARAM mpParm2)
{
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);
```

```
   switch (ulMsg) {
   case WM_CREATE:
      pidData=(PINSTDATA)malloc(sizeof(INSTDATA));
      if (pidData==NULL) {
         WinAlarm(HWND_DESKTOP,WA_ERROR);
         return MRFROMSHORT(TRUE);
      } /* endif */

      WinSetWindowPtr(hwndWnd,0,(PVOID)pidData);

      pidData->ulSzStruct=sizeof(INSTDATA);
      pidData->hwndMenu=WinLoadMenu(HWND_OBJECT,
                                    NULLHANDLE,
                                    RES_CLIENT);
      break;
   case WM_DESTROY:
      WinDestroyWindow(pidData->hwndMenu);
      free(pidData);
      break;
   case WM_CONTEXTMENU:
      {
         POINTL ptlMouse;

         WinQueryPointerPos(HWND_DESKTOP,&ptlMouse);
         WinPopupMenu(HWND_DESKTOP,
                      hwndWnd,
                      pidData->hwndMenu,
                      ptlMouse.x,
                      ptlMouse.y,
                      0,
                      PU_HCONSTRAIN | PU_VCONSTRAIN |
                         PU_NONE | PU_KEYBOARD |
                         PU_MOUSEBUTTON1 |
                         PU_MOUSEBUTTON2);
      }
      break;
   case WM_COMMAND:
      switch (SHORT1FROMMP(mpParm1)) {
      case MI_THREAD:
         {
            TESTTHREADINFO ttiTest;

            //---------------------------------------
            // Request a thread
            //---------------------------------------
            WinSendMsg(hwndWnd,
                       MYM_STARTTHREAD,
                       MPFROMLONG(ASYNC_TEST),
                       MPFROMP(&ttiTest));
         }
         break;
      case MI_EXIT:
         WinPostMsg(hwndWnd,WM_CLOSE,0,0);
         break;
      default:
         return WinDefWindowProc(hwndWnd,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2);
      } /* endswitch */
      break;
   case WM_PAINT:
      {
         HPS hpsWnd;
         RECTL rclPaint;
```

```
      hpsWnd=WinBeginPaint(hwndWnd,
                           NULLHANDLE,
                           &rclPaint);
      WinFillRect(hpsWnd,&rclPaint,SYSCLR_WINDOW);
      WinEndPaint(hpsWnd);
   }
   break;
case MYM_STARTTHREAD:
   {
      ULONG ulBit;
      PTHREADINFO ptiInput;
      PFNREQ pfnThread;
      PVOID pvParm;

      ulBit=LONGFROMMR(mpParm1);
      ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);

      ptiInput->hwndOwner=hwndWnd;
      ptiInput->bKillThread=FALSE;

      switch (ulBit) {
      case ASYNC_TEST:
         {
            PTESTTHREADINFO pttiInfo;

            ptiInput->ulSzStruct=
               sizeof(TESTTHREADINFO);

            pttiInfo=
               (PTESTTHREADINFO)malloc(
                  sizeof(TESTTHREADINFO));
            if (pttiInfo==NULL) {
               WinMessageBox(HWND_DESKTOP,
                             hwndWnd,
                             "There is not "
                                "enough memory.",
                             "Error",
                             0,
                             MB_OK |
                                MB_ICONEXCLAMATION|
                                MB_MOVEABLE);
               return MRFROMSHORT(FALSE);
            } /* endif */

            memcpy(pttiInfo,
                   ptiInput,
                   sizeof(TESTTHREADINFO));
            pfnThread=(PFNREQ)testThread;
            pvParm=(PVOID)pttiInfo;
         }
         break;
      default:
         WinMessageBox(HWND_DESKTOP,
                       hwndWnd,
                       "There is an internal "
                          "error.",
                       "Error",
                       0,
                       MB_OK |
                          MB_ICONEXCLAMATION |
                          MB_MOVEABLE);
         return MRFROMSHORT(FALSE);
      } /* endswitch */
```

```
         if (_beginthread(pfnThread,
                          NULL,
                          0x4000,
                          pvParm)==-1) {
            free(pvParm);
            WinMessageBox(HWND_DESKTOP,
                          hwndWnd,
                          "The thread could not be "
                             "created.",
                          "Error",
                          0,
                          MB_OK |
                             MB_ICONEXCLAMATION |
                             MB_MOVEABLE);
            return MRFROMSHORT(FALSE);
         } /* endif */
      }
      break;
   case MYM_ENDTHREAD:
      {
         ULONG ulBit;
         PTHREADINFO ptiInput;

         ulBit=LONGFROMMR(mpParm1);
         ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);

         //-----------------------------------------
         // Wait for the thread to finish dying.
         // There is a bug in DosSleep() such that if
         // 0 is the argument, nothing happens.  Call
         // it with 1 instead to achieve the same
         // result.
         //-----------------------------------------
         while (!ptiInput->bThreadDead) {
            DosSleep(1);
         } /* endwhile */

         switch (ulBit) {
         case ASYNC_TEST:
            {
               PTESTTHREADINFO pttiInfo;

               pttiInfo=(PTESTTHREADINFO)ptiInput;
               free(pttiInfo);
            }
            break;
         default:
            return MRFROMSHORT(FALSE);
         } /* endswitch */
      }
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(USHORT usArgs,PCHAR apchArgs[])
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
```

```
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_MAIN,
                    wndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
            FCF_SIZEBORDER | FCF_TASKLIST |
            FCF_SHELLPOSITION;

   if (WinQuerySysValue(HWND_DESKTOP,SV_ANIMATION)) {
      ulCreate|=WS_ANIMATE;
   } /* endif */

   hwndFrame=WinCreateStdWindow(HWND,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_MAIN,
                                "Asynchronous Call",
                                0L,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,
                         &qmMsg,
                         NULLHANDLE,
                         0,
                         0);
      } /* endwhile */

      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

## THRD1.RC

```
#include <os2.h>
#include "thrd1rc.h"

MENU RES_CLIENT
{
   MENUITEM "~Start thread", MI_THREAD
   MENUITEM SEPARATOR
   MENUITEM "E~xit", MI_EXIT
}
```

**THRD1RC.H**

```
#define RES_CLIENT                256
#define MI_THREAD                 257
#define MI_EXIT                   258
```

**THRD1.MAK**

```
APP=THRD1

$(APP).EXE:                       $(APP).OBJ \
                                  $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                       $(APP).RC \
                                  $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                       $(APP).C \
                                  $(APP)RC.H
        ICC -C+ -Gm+ -Kb+ -Ss+ $(APP).C
```

**THRD1.DEF**

```
NAME THRD1 WINDOWAPI

DESCRIPTION 'PM Threads Example 1
Copyright (c) 1993 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

Readers should recognize and understand much of the program. Of particular interest is the processing for the MI_THREAD menu item.

```
case MI_THREAD:
   {
      TESTTHREADINFO ttiTest;

      //-------------------------------------
      // Request a thread
      //-------------------------------------
      WinSendMsg(hwndWnd,
                 MYM_STARTTHREAD,
                 MPFROMLONG(ASYNC_TEST),
                 MPFROMP(&ttiTest));
   }
   break;
```

That is all there is to it. Of course, this sample is simplified somewhat. If, as will likely be the case, there is task-specific data (there is none in the THRD1 sample), you should be initialized prior to sending the MYM_STARTTHREAD message.

The thread procedure contains some elements that will likely show up in thread procedures. First is the thread initialization, including initializing the remainder of the THREADINFO structure. Also is the thread termination, including signaling the owner thread that it is finished.

```
habAnchor=WinInitialize(0);
hmqQueue=WinCreateMsgQueue(habAnchor,0);

pttiInfo->tiInfo.habThread=habAnchor;
```

```
pttiInfo->tiInfo.bThreadDead=FALSE;
pttiInfo->tiInfo.bResult=FALSE;
   :
   :
   :
WinPostMsg(pttiInfo->tiInfo.hwndOwner,
           MYM_ENDTHREAD,
           MPFROMLONG(ASYNC_TEST),
           MPFROMP(pttiInfo));

WinDestroyMsgQueue(hmqQueue);
WinTerminate(habAnchor);

DosEnterCritSec();
pttiInfo->tiInfo.bThreadDead=TRUE;
return;
```

As with the call to *DosSleep* in MYM_ENDTHREAD given earlier, the critical section at the end of the thread is harmless because it exists only briefly.

What would happen if the *WinPostMsg* was changed to *WinSendMsg*? Looking at the code for MYM_ENDTHREAD, the window procedure would enter a loop waiting for the thread to die, but the thread is in the middle of a *WinSendMsg* call; a deadlock condition occurs, and killing the application requires precision timing and a little bit of luck.

**Gotcha!**
If a thread enters a critical section and then dies, the system automatically marks the critical section as having been exited.

A typical question that is asked is why an anchor block and a message queue are needed for such a simple thread. The answer is that they aren't. However, rather than try to determine if a thread needs a message queue or not, I decided long ago that my time was better spent by creating it anyway and continuing in my development.

## User Feedback

Earlier, we glossed over the issue of feedback to the user. How can we indicate that processing is being performed in the background? While the answer to this and other similar questions is "it depends on the application," here are some areas that need to be considered.

*Mouse pointers* are an immediate indicator that "something" is happening, and the system pointers SPTR_WAIT and SPTR_ARROW (whose handle is obtained via the *WinQuerySysPointer* function) come in handy. Where is the pointer changed? The answer appears to be in the processing for WM_MOUSEMOVE and WM_CONTROLPOINTER, but first we need to be able to tell if something is going on.

We need to introduce the only data item used for the duration of the window, which goes into the instance data. In Chapter 9, we explained how *window words* are used to hold information specific to a window *instance* (versus a window *class*). Storing a pointer to a dynamically allocated structure so that we can "attach" a lot of data to a window was also discussed. If, in the window words, we add a new field—

*ulAsync*, we can store either the number of threads owned by the window or (using the ASYNC_ constants) the types of threads owned by the window that are active.

This makes the WM_MOUSEMOVE and WM_CONTROLPOINTER messages trivial; we simply check the value of *ulAsync*. If it is nonzero, we set the pointer to SPTR_WAIT; otherwise we leave it alone.

*Menu items* are another issue. If the user requests that a file be opened, we (usually) do not want to allow them to try an print the file until we have finished reading the file's contents. This can be addressed again using *ulAsync* and the WM_INITMENU message; this message is sent whenever a menu or submenu is about to be displayed, allowing the application to disable, check, or perform any other operation on the (sub)menu before the user sees it. We could disable the menu items that are not valid according to the threads that are active.

## User Feedback Example

Let us now take a look at a revised version of THRD1 that includes feedback to the user. It changes the mouse pointer and disables the "Start thread" menu item if the thread is already active.

**THRD2.C**

```
#define INCL_DOSPROCESS
#define INCL_WINFRAMEMGR
#define INCL_WININPUT
#define INCL_WINMENUS
#define INCL_WINPOINTERS
#define INCL_WINSYS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include "thrd2rc.h"

#define CLS_MAIN                "Thread2Class"

#define ASYNC_TEST              0x00000001L

typedef VOID (* _Optlink PFNREQ)(PVOID);

#define MYM_BASE                (WM_USER)
#define MYM_STARTTHREAD         (MYM_BASE)
#define MYM_ENDTHREAD           (MYM_BASE+1)

typedef struct _THREADINFO {
   //-------------------------------------------------
   // Initialized by the main thread
   //-------------------------------------------------
   ULONG ulSzStruct;
   HWND hwndOwner;
   BOOL bKillThread;
   //-------------------------------------------------
   // Initialized by the secondary thread
   //-------------------------------------------------
   HAB habThread;
   BOOL bThreadDead;
   BOOL bResult;
} THREADINFO, *PTHREADINFO;

typedef struct _TESTTHREADINFO {
   THREADINFO tiInfo;
} TESTTHREADINFO, *PTESTTHREADINFO;
```

```
typedef struct _INSTDATA {
   ULONG ulSzStruct;
   HWND hwndMenu;
   ULONG ulAsync;
} INSTDATA, *PINSTDATA;

VOID _Optlink testThread(PTESTTHREADINFO pttiInfo)
//-----------------------------------------------------
// This is a do nothing thread that beeps, sleeps,
// and beeps again
//-----------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   pttiInfo->tiInfo.habThread=habAnchor;
   pttiInfo->tiInfo.bThreadDead=FALSE;
   pttiInfo->tiInfo.bResult=FALSE;

   WinAlarm(HWND_DESKTOP,WA_NOTE);
   DosSleep(2000);
   WinAlarm(HWND_DESKTOP,WA_NOTE);

   WinPostMsg(pttiInfo->tiInfo.hwndOwner,
              MYM_ENDTHREAD,
              MPFROMLONG(ASYNC_TEST),
              MPFROMP(pttiInfo));

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);

   DosEnterCritSec();
   pttiInfo->tiInfo.bThreadDead=TRUE;
   return;
}

MRESULT EXPENTRY wndProc(HWND hwndWnd,
                         ULONG ulMsg,
                         MPARAM mpParm1,
                         MPARAM mpParm2)
{
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);

   switch (ulMsg) {
   case WM_CREATE:
      pidData=(PINSTDATA)malloc(sizeof(INSTDATA));
      if (pidData==NULL) {
         WinAlarm(HWND_DESKTOP,WA_ERROR);
         return MRFROMSHORT(TRUE);
      } /* endif */

      WinSetWindowPtr(hwndWnd,0,(PVOID)pidData);

      pidData->ulSzStruct=sizeof(INSTDATA);
      pidData->hwndMenu=WinLoadMenu(HWND_OBJECT,
                                    NULLHANDLE,
                                    RES_CLIENT);

      pidData->ulAsync=0;
      break;
   case WM_DESTROY:
```

```
      WinDestroyWindow(pidData->hwndMenu);
      free(pidData);
      break;
   case WM_CONTEXTMENU:
      {
         POINTL ptlMouse;

         WinQueryPointerPos(HWND_DESKTOP,&ptlMouse);
         WinPopupMenu(HWND_DESKTOP,
                      hwndWnd,
                      pidData->hwndMenu,
                      ptlMouse.x,
                      ptlMouse.y,
                      0,
                      PU_HCONSTRAIN | PU_VCONSTRAIN |
                         PU_NONE | PU_KEYBOARD |
                         PU_MOUSEBUTTON1 |
                         PU_MOUSEBUTTON2);
      }
      break;
   case WM_COMMAND:
      switch (SHORT1FROMMP(mpParm1)) {
      case MI_THREAD:
         {
            TESTTHREADINFO ttiTest;

            //---------------------------------------
            // Request a thread
            //---------------------------------------
            WinSendMsg(hwndWnd,
                       MYM_STARTTHREAD,
                       MPFROMLONG(ASYNC_TEST),
                       MPFROMP(&ttiTest));
         }
         break;
      case MI_EXIT:
         WinPostMsg(hwndWnd,WM_CLOSE,0,0);
         break;
      default:
         return WinDefWindowProc(hwndWnd,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2);
      } /* endswitch */
      break;
   case WM_PAINT:
      {
         HPS hpsWnd;
         RECTL rclPaint;

         hpsWnd=WinBeginPaint(hwndWnd,
                              NULLHANDLE,
                              &rclPaint);
         WinFillRect(hpsWnd,&rclPaint,SYSCLR_WINDOW);
         WinEndPaint(hpsWnd);
      }
      break;
   case WM_INITMENU:
      switch (SHORT1FROMMP(mpParm1)) {
      case FID_MENU:
         if ((pidData->ulAsync & ASYNC_TEST)!=0) {
            WinEnableMenuItem(HWNDFROMMP(mpParm2),
                              MI_THREAD,
                              FALSE);
         } else {
```

```
            WinEnableMenuItem(HWNDFROMMP(mpParm2),
                              MI_THREAD,
                              TRUE);
         } /* endif */
         break;
      default:
         return WinDefWindowProc(hwndWnd,
                                 ulMsg,
                                 mpParm1,
                                 mpParm2);
      } /* endswitch */
      break;
   case WM_MOUSEMOVE:
      {
         HPOINTER hpPointer;

         if (pidData->ulAsync>0) {
            hpPointer=
               WinQuerySysPointer(HWND_DESKTOP,
                                  SPTR_WAIT,
                                  FALSE);
            WinSetPointer(HWND_DESKTOP,hpPointer);
            return MRFROMSHORT(TRUE);
         } else {
            return WinDefWindowProc(hwndWnd,
                                    ulMsg,
                                    mpParm1,
                                    mpParm2);
         } /* endif */
      }
   case WM_CONTROLPOINTER:
      {
         HPOINTER hpPointer;

         if (pidData->ulAsync>0) {
            hpPointer=
               WinQuerySysPointer(HWND_DESKTOP,
                                  SPTR_WAIT,
                                  FALSE);
            return MRFROMLONG(hpPointer);
         } else {
            return WinDefWindowProc(hwndWnd,
                                    ulMsg,
                                    mpParm1,
                                    mpParm2);
         } /* endif */
      }
   case MYM_STARTTHREAD:
      {
         ULONG ulBit;
         PTHREADINFO ptiInput;
         PFNREQ pfnThread;
         PVOID pvParm;

         ulBit=LONGFROMMR(mpParm1);
         ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);

         ptiInput->hwndOwner=hwndWnd;
         ptiInput->bKillThread=FALSE;

         switch (ulBit) {
         case ASYNC_TEST:
            {
               PTESTTHREADINFO pttiInfo;

               ptiInput->ulSzStruct=
                  sizeof(TESTTHREADINFO);
```

```
            pttiInfo=
               (PTESTTHREADINFO)malloc(
                  sizeof(TESTTHREADINFO));
            if (pttiInfo==NULL) {
               WinMessageBox(HWND_DESKTOP,
                             hwndWnd,
                             "There is not "
                                "enough memory.",
                             "Error",
                             0,
                             MB_OK |
                                MB_ICONEXCLAMATION|
                                MB_MOVEABLE);
               return MRFROMSHORT(FALSE);
            } /* endif */

            memcpy(pttiInfo,
                   ptiInput,
                   sizeof(TESTTHREADINFO));
            pfnThread=(PFNREQ)testThread;
            pvParm=(PVOID)pttiInfo;
         }
         break;
      default:
         WinMessageBox(HWND_DESKTOP,
                       hwndWnd,
                       "There is an internal "
                          "error.",
                       "Error",
                       0,
                       MB_OK |
                          MB_ICONEXCLAMATION |
                          MB_MOVEABLE);
         return MRFROMSHORT(FALSE);
      } /* endswitch */

      if (_beginthread(pfnThread,
                       NULL,
                       0x4000,
                       pvParm)==-1) {
         free(pvParm);
         WinMessageBox(HWND_DESKTOP,
                       hwndWnd,
                       "The thread could not be "
                          "created.",
                       "Error",
                       0,
                       MB_OK |
                          MB_ICONEXCLAMATION |
                          MB_MOVEABLE);
         return MRFROMSHORT(FALSE);
      } /* endif */

      pidData->ulAsync|=ulBit;
   }
   break;
case MYM_ENDTHREAD:
   {
      ULONG ulBit;
      PTHREADINFO ptiInput;

      ulBit=LONGFROMMR(mpParm1);
      ptiInput=(PTHREADINFO)PVOIDFROMMP(mpParm2);
```

```
         //----------------------------------------
         // Wait for the thread to finish dying.
         // There is a bug in DosSleep() such that if
         // 0 is the argument, nothing happens.  Call
         // it with 1 instead to achieve the same
         // result.
         //----------------------------------------
         while (!ptiInput->bThreadDead) {
            DosSleep(1);
         } /* endwhile */

         switch (ulBit) {
         case ASYNC_TEST:
            {
               PTESTTHREADINFO pttiInfo;

               pttiInfo=(PTESTTHREADINFO)ptiInput;
               free(pttiInfo);
            }
            break;
         default:
            return MRFROMSHORT(FALSE);
         } /* endswitch */

         pidData->ulAsync&=~ulBit;
      }
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(USHORT usArgs,PCHAR apchArgs[])
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_MAIN,
                    wndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
            FCF_SIZEBORDER | FCF_TASKLIST |
            FCF_SHELLPOSITION;

   if (WinQuerySysValue(HWND_DESKTOP,SV_ANIMATION)) {
      ulCreate|=WS_ANIMATE;
   } /* endif */
```

```
   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_MAIN,
                                "Asynchronous Call "
                                   "With Feedback",
                                0L,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,
                         &qmMsg,
                         NULLHANDLE,
                         0,
                         0);
      } /* endwhile */

      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

**THRD2.RC**

```
#include <os2.h>
#include "thrd2rc.h"

MENU RES_CLIENT
{
   MENUITEM "~Start thread", MI_THREAD
   MENUITEM SEPARATOR
   MENUITEM "E~xit", MI_EXIT
}
```

**THRD2RC.H**

```
#define RES_CLIENT              256
#define MI_THREAD               257
#define MI_EXIT                 258
```

**THRD2.MAK**

```
APP=THRD2

$(APP).EXE:                     $(APP).OBJ \
                                $(APP).RES
        LINK386 /A:16 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                     $(APP).RC \
                                $(APP)RC.H
        RC -r $(APP).RC $(APP).RES
```

```
$(APP).OBJ:                     $(APP).C \
                                $(APP)RC.H
        ICC -C+ -Gm+ -Kb+ -Ss+ $(APP).C
```

**THRD2.DEF**

```
NAME THRD2 WINDOWAPI

DESCRIPTION 'PM Threads Example 2
Copyright (c) 1993 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

As we discussed, the three messages that we are interested in are WM_INITMENU, WM_MOUSEMOVE, and WM_CONTROLPOINTER, which are grouped together just before the MYM_STARTTHREAD message.

```
case WM_INITMENU:
   switch (SHORT1FROMMP(mpParm1)) {
   case FID_MENU:
      if ((pidData->ulAsync & ASYNC_TEST)!=0) {
         WinEnableMenuItem(HWNDFROMMP(mpParm2),
                           MI_THREAD,
                           FALSE);
      } else {
         WinEnableMenuItem(HWNDFROMMP(mpParm2),
                           MI_THREAD,
                           TRUE);
      } /* endif */
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */
   break;
```

We first need to determine, in the preceding code, which menu is about to be displayed. In our application, this is unnecessary, since there are no submenus, but for illustrative purposes the check is included. If the ASYNC_TEST bit is set in *pidData->ulAsync*, then we disable the item; otherwise we reenable it.

This brings up an interesting point for programmers to consider: Suppose it is valid to have multiple threads of the same type active simultaneously. We can no longer set individual bits in *ulAsync*, but if we simply keep a thread count, we do not know what types of threads are active. The solution to this dilemma is left to readers as an exercise.

```
case WM_MOUSEMOVE:
   {
      HPOINTER hpPointer;

      if (pidData->ulAsync>0) {
         hpPointer=
            WinQuerySysPointer(HWND_DESKTOP,
                               SPTR_WAIT,
                               FALSE);
         WinSetPointer(HWND_DESKTOP,hpPointer);
         return MRFROMSHORT(TRUE);
      } else {
```

```
            return WinDefWindowProc(hwndWnd,
                                    ulMsg,
                                    mpParm1,
                                    mpParm2);
         } /* endif */
      }
   case WM_CONTROLPOINTER:
      {
         HPOINTER hpPointer;

         if (pidData->ulAsync>0) {
            hpPointer=
               WinQuerySysPointer(HWND_DESKTOP,
                                  SPTR_WAIT,
                                  FALSE);
            return MRFROMLONG(hpPointer);
         } else {
            return WinDefWindowProc(hwndWnd,
                                    ulMsg,
                                    mpParm1,
                                    mpParm2);
         } /* endif */
      }
```

The processing for these two messages is trivial but their effect is profound. By changing the pointer, the user is instantly notified of background processing. More important, by changing the pointer in these messages, only our application is affected, allowing other applications running to be used while the task is performed.

# Synchronicity

Ah ... back to the old days, when programming in DOS was considered exotic—there was only one process, text mode was considered an okay interface for most programs, and function calls were always synchronous. Well, the first two items might no longer hold true, but the last one is at least attainable for one-shot threads.

What? How can an asynchronous concept like multithreading be done synchronously? That idea is paradoxical in itself, much less the attempt at implementing it! The trick here is to reconsider the issue of synchronicity; it is, as Einstein would have said, based on frame of reference. In other words, something could not in reality be synchronous but appear so to the user (the application program).

In the beginning of the chapter we stated the one-tenth-second rule, which said that, in summary, the application must remain responsive to the user. What would happen if we wrote a function that started a thread and immediately went into a message loop until the thread was finished? Take a look at the *WinDlgBox* or *WinMessageBox* functions; they are both synchronous functions whose length of execution is dependent on the user, yet they do not "hang" the application. How do they do it? Now you know—they initialize their environment and then enter a message loop to insure that responsiveness is maintained.

In order to implement this concept in a modular fashion, we need to think carefully. It should be obvious that all "synchronous" threads are going to have a call to *_beginthread* followed by a message loop, in addition to other stuff. If we extract this portion out, we can write a generic "dispatch" function.

```
#define DT_NOERROR              0
#define DT_QUITRECEIVED         1
#define DT_ERROR                2
```

```
USHORT dispatchThread(HAB habAnchor,
                      PFNREQ pfnThread,
                      PTHREADINFO ptiInfo)
{
   TID tidThread;
   BOOL bLoop;
   QMSG qmMsg;

   ptiInfo->bKillThread=FALSE;
   ptiInfo->bThreadDead=FALSE;

   tidThread=_beginthread(pfnThread,
                          NULL,
                          0x4000,
                          ptiInfo);
   if (tidThread==(TID)-1) {
      return DT_ERROR;
   } /* endif */

   WinPeekMsg(habAnchor,
              &qmMsg,
              NULLHANDLE,
              0,
              0,
              PM_REMOVE);
   bLoop=((qmMsg.msg!=WM_QUIT) &&
          (!ptiInfo->bThreadDead));

   while (bLoop) {
      WinDispatchMsg(habAnchor,&qmMsg);
      WinPeekMsg(habAnchor,
                 &qmMsg,
                 NULLHANDLE,
                 0,
                 0,
                 PM_REMOVE);
      bLoop=((qmMsg.msg!=WM_QUIT) &&
             (!ptiInfo->bThreadDead));
   } /* endwhile */

   if (qmMsg.msg==WM_QUIT) {
      DosKillThread(tidThread);
      return DT_QUITRECEIVED;
   } /* endif */

   return DT_NOERROR;
}
```

The definitions of PFNREQ and THREADINFO are the same as before, so this function shouldn't be too hard to digest. There are a few things that aren't obvious, however.

The first is the initialization of *bThreadDead*. Before, this was done in the thread, but since we immediately start checking this value after the call to *_beginthread*, we should initialize this ourselves because conceivably the thread could have had no timeslices before we query this value.

The second item of note is the use of *WinPeekMsg* instead of *WinGetMsg*.

```
BOOL WinPeekMsg(HAB habAnchor,
                PQMSG pqmMsg,
                HWND hwndFilter,
                ULONG ulFilterFirst,
                ULONG ulFilterLast,
                ULONG ulFlags);
```

The parameters are all the same as with *WinGetMsg* (discussed in Chapter 11), with the exception of *ulFlags*, which is unique to *WinPeekMsg*. It can have the value PM_REMOVE or PM_NOREMOVE, which specifies that the message in the queue is to be removed or not removed, respectively. We are not interested in the parameters, however; our concern is with the behavior. If there are no messages in the queue, *WinPeekMsg* will return immediately, while *WinGetMsg* will not. This is significant because, if the user does not touch the mouse or the keyboard, and no timers are started, the *dispatchThread* function will never return, even though the thread might have finished.

Some of the PMWIN developers at IBM discouraged this use of *DosKillThread*, so it is not necessarily a good one. Supposedly, its use can cause stability problems if the thread being killed has a message queue. I use it here because I have never had any problems with it, but this isn't to say that the problem doesn't exist. Mileage may vary.

## Synchronous Threading Example

The following sample program illustrates this "synchronous" threading concept, which is applied it to THRD1, cited earlier.

**THRD3.C**

```
#define INCL_DOSPROCESS
#define INCL_WININPUT
#define INCL_WINMENUS
#define INCL_WINPOINTERS
#define INCL_WINSYS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdlib.h>
#include "thrd3rc.h"

#define CLS_MAIN                "Thread3Class"

#define DT_NOERROR              0
#define DT_QUITRECEIVED         1
#define DT_ERROR                2

typedef VOID (* _Optlink PFNREQ)(PVOID);

typedef struct _THREADINFO {
   //-------------------------------------------------
   // Initialized by the main thread
   //-------------------------------------------------
   ULONG ulSzStruct;
   HWND hwndOwner;
   BOOL bKillThread;
   //-------------------------------------------------
   // Initialized by the secondary thread
   //-------------------------------------------------
   HAB habThread;
   BOOL bThreadDead;
   BOOL bResult;
} THREADINFO, *PTHREADINFO;

typedef struct _TESTTHREADINFO {
   THREADINFO tiInfo;
} TESTTHREADINFO, *PTESTTHREADINFO;
```

```
typedef struct _INSTDATA {
   ULONG ulSzStruct;
   HWND hwndMenu;
} INSTDATA, *PINSTDATA;

USHORT dispatchThread(HAB habAnchor,
                      PFNREQ pfnThread,
                      PTHREADINFO ptiInfo);
VOID _Optlink _testThread(PTESTTHREADINFO pttiInfo);
BOOL testThread(HWND hwndWnd,
                PTESTTHREADINFO pttiInfo);

USHORT dispatchThread(HAB habAnchor,
                      PFNREQ pfnThread,
                      PTHREADINFO ptiInfo)
//-----------------------------------------------------
// This is the thread dispatch procedure.  It calls
// _beginthread() and goes into a
// WinPeekMsg()/WinDispatchMsg() loop until the
// thread is finished or WM_QUIT is received.  Note
// the semantics of the latter event:  if WM_QUIT is
// received, then it is assumed that the application
// will kill itself on return and thus any system
// resources will automatically be unallocated by the
// system when the application ends.  So we do not
// set bKillThread=TRUE and wait but instead call
// DosKillThread() and return.
//-----------------------------------------------------
{
   TID tidThread;
   BOOL bLoop;
   QMSG qmMsg;

   ptiInfo->bKillThread=FALSE;
   ptiInfo->bThreadDead=FALSE;

   tidThread=_beginthread(pfnThread,
                          NULL,
                          0x4000,
                          ptiInfo);
   if (tidThread==(TID)-1) {
      return DT_ERROR;
   } /* endif */

   //--------------------------------------------------
   // WinGetMsg() cannot be used because it blocks if
   // there is no message waiting.  When the thread
   // dies, therefore, the function will never return
   // if the user takes his/her hands off of the
   // keyboard, mouse, and no timers are started
   // because we will never get a message!
   //--------------------------------------------------
   WinPeekMsg(habAnchor,
              &qmMsg,
              NULLHANDLE,
              0,
              0,
              PM_REMOVE);
   bLoop=((qmMsg.msg!=WM_QUIT) &&
          (!ptiInfo->bThreadDead));

   while (bLoop) {
      WinDispatchMsg(habAnchor,&qmMsg);
```

```
      WinPeekMsg(habAnchor,
                 &qmMsg,
                 NULLHANDLE,
                 0,
                 0,
                 PM_REMOVE);
      bLoop=((qmMsg.msg!=WM_QUIT) &&
             (!ptiInfo->bThreadDead));
   } /* endwhile */

   if (qmMsg.msg==WM_QUIT) {
      DosKillThread(tidThread);
      return DT_QUITRECEIVED;
   } /* endif */

   return DT_NOERROR;
}

VOID _Optlink _testThread(PTESTTHREADINFO pttiInfo)
//-----------------------------------------------------
// This is a do nothing thread that beeps, sleeps,
// and beeps again
//-----------------------------------------------------
{
   HAB habAnchor;
   HMQ hmqQueue;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   pttiInfo->tiInfo.habThread=habAnchor;
   pttiInfo->tiInfo.bThreadDead=FALSE;
   pttiInfo->tiInfo.bResult=FALSE;

   WinAlarm(HWND_DESKTOP,WA_NOTE);
   DosSleep(2000);
   WinAlarm(HWND_DESKTOP,WA_NOTE);

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);

   DosEnterCritSec();
   pttiInfo->tiInfo.bThreadDead=TRUE;
   return;
}

BOOL testThread(HWND hwndWnd,PTESTTHREADINFO pttiInfo)
//-----------------------------------------------------
// This function simply initializes any thread-
// specific fields and calls dispatchThread().
//-----------------------------------------------------
{
   pttiInfo->tiInfo.ulSzStruct=sizeof(TESTTHREADINFO);

   return dispatchThread(WinQueryAnchorBlock(hwndWnd),
                         (PFNREQ)_testThread,
                         (PTHREADINFO)pttiInfo);
}

MRESULT EXPENTRY wndProc(HWND hwndWnd,
                         ULONG ulMsg,
                         MPARAM mpParm1,
                         MPARAM mpParm2)
{
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);
```

```
switch (ulMsg) {
case WM_CREATE:
   pidData=(PINSTDATA)malloc(sizeof(INSTDATA));
   if (pidData==NULL) {
      WinAlarm(HWND_DESKTOP,WA_ERROR);
      return MRFROMSHORT(TRUE);
   } /* endif */

   WinSetWindowPtr(hwndWnd,0,(PVOID)pidData);

   pidData->ulSzStruct=sizeof(INSTDATA);
   pidData->hwndMenu=WinLoadMenu(HWND_OBJECT,
                                 NULLHANDLE,
                                 RES_CLIENT);
   break;
case WM_DESTROY:
   WinDestroyWindow(pidData->hwndMenu);
   free(pidData);
   break;
case WM_CONTEXTMENU:
   {
      POINTL ptlMouse;

      WinQueryPointerPos(HWND_DESKTOP,&ptlMouse);
      WinPopupMenu(HWND_DESKTOP,
                   hwndWnd,
                   pidData->hwndMenu,
                   ptlMouse.x,
                   ptlMouse.y,
                   0,
                   PU_HCONSTRAIN | PU_VCONSTRAIN |
                      PU_NONE | PU_KEYBOARD |
                      PU_MOUSEBUTTON1 |
                      PU_MOUSEBUTTON2);
   }
   break;
case WM_COMMAND:
   switch (SHORT1FROMMP(mpParm1)) {
   case MI_THREAD:
      {
         TESTTHREADINFO ttiTest;

         //---------------------------------------
         // Call testThread() directly
         //---------------------------------------
         if (testThread(hwndWnd,&ttiTest)==
               DT_QUITRECEIVED) {
            WinPostMsg(hwndWnd,WM_CLOSE,0,0);
         } /* endif */
      }
      break;
   case MI_EXIT:
      WinPostMsg(hwndWnd,WM_CLOSE,0,0);
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */
   break;
case WM_PAINT:
   {
      HPS hpsWnd;
      RECTL rclPaint;
```

```
         hpsWnd=WinBeginPaint(hwndWnd,
                              NULLHANDLE,
                              &rclPaint);
         WinFillRect(hpsWnd,&rclPaint,SYSCLR_WINDOW);
         WinEndPaint(hpsWnd);
      }
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(USHORT usArgs,PCHAR apchArgs[])
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_MAIN,
                    wndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
            FCF_SIZEBORDER | FCF_TASKLIST |
            FCF_SHELLPOSITION;

   if (WinQuerySysValue(HWND_DESKTOP,SV_ANIMATION)) {
      ulCreate|=WS_ANIMATE;
   } /* endif */

   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_MAIN,
                                "Synchronous Call",
                                0L,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,
                         &qmMsg,
                         NULLHANDLE,
                         0,
                         0);
      } /* endwhile */
```

```
      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

**THRD3.RC**

```
#include <os2.h>
#include "thrd3rc.h"

MENU RES_CLIENT
{
   MENUITEM "~Start thread", MI_THREAD
   MENUITEM SEPARATOR
   MENUITEM "E~xit", MI_EXIT
}
```

**THRD3RC.H**

```
#define RES_CLIENT              256
#define MI_THREAD               257
#define MI_EXIT                 258
```

**THRD3.MAK**

```
APP=THRD3

$(APP).EXE:                     $(APP).OBJ \
                                $(APP).RES
        LINK386 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                     $(APP).RC \
                                $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                     $(APP).C \
                                $(APP)RC.H
        ICC -C+ -Gm+ -Kb+ -Ss+ $(APP).C
```

**THRD3.DEF**

```
NAME THRD3 WINDOWAPI

DESCRIPTION 'PM Threads Example 3
Copyright (c) 1993 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

# Object Windows

The final method that we will look at here for performing long tasks asynchronously is the use of *object windows*. An object window is like any other window used in other applications with the following, very important exceptions:

- Object windows do not receive any system messages other than WM_CREATE and WM_DESTROY.
- Object windows are not subject to the one-tenth of a second rule.

The second point is simply a consequence of the first. Remember, the 1/10th of a second rule came about to insure that input messages (keyboard and mouse) were transferred from the system input queue to the message queue of the application. However, the first point says that object windows receive only the two messages listed; this means that they never receive the mouse or keyboard messages from the system.

**Gotcha!**
Although an object window can take more than one-tenth of a second to process a request, a call to *WinSendMsg* will not return until the object window exits its window procedure. Thus, *WinPostMsg* should be used to communicate with an object window unless it is absolutely necessary to send the message instead. This same logic applies to the *WinDispatchMsg* function, as we'll see.

An object window typically is not used for one-shot threads because of its ability to send and receive messages and its persistence due to the message loop in the thread. Object windows instead lean toward client/server applications, although there is nothing that object windows can do that cannot be done with the one-shot architecture already discussed.

# Building a Blind Window

Now we know what an object window is and for what it is used, but how do we use it in our application? Since an object window can take as long as it feels necessary to process a message, we cannot use the message loop of the main thread to dispatch messages to it (as was explained in the last "Gotcha"). What is needed is the creation of a second thread that has its own message loop in it.

Communication with the object window is done through user messages, as we see in the next example.

**THRD4.C**

```
#define INCL_DOSPROCESS
#define INCL_WININPUT
#define INCL_WINMENUS
#define INCL_WINPOINTERS
#define INCL_WINSYS
#define INCL_WINWINDOWMGR
#include <os2.h>
#include <stdlib.h>
#include "thrd4rc.h"

#define CLS_MAIN                "Thread4Class"
#define CLS_OBJECT              "Thread4ObjectClass"

#define MYM_BASE                (WM_USER)
#define MYM_STARTREQUEST        (MYM_BASE)
#define MYM_ENDREQUEST          (MYM_BASE+1)

#define ASYNC_NOTE              0
#define ASYNC_WARNING           1
#define ASYNC_ERROR             2

typedef VOID (* _Optlink PFNREQ)(PVOID);
```

```
typedef struct _INSTDATA {
   ULONG ulSzStruct;
   HWND hwndMenu;
   HWND hwndObject;
} INSTDATA, *PINSTDATA;

MRESULT EXPENTRY objectProc(HWND hwndWnd,
                            ULONG ulMsg,
                            MPARAM mpParm1,
                            MPARAM mpParm2)
{
   switch (ulMsg) {
   case MYM_STARTREQUEST:
      {
         ULONG ulRequest;
         HWND hwndOwner;
         ULONG ulNote;

         //-----------------------------------------
         // Get the specifics for this request
         //-----------------------------------------
         ulRequest=LONGFROMMP(mpParm1);
         hwndOwner=HWNDFROMMP(mpParm2);

         switch (ulRequest) {
         case ASYNC_NOTE:
            ulNote=WA_NOTE;
            break;
         case ASYNC_WARNING:
            ulNote=WA_WARNING;
            break;
         case ASYNC_ERROR:
         default:
            ulNote=WA_ERROR;
            break;
         } /* endswitch */

         //-----------------------------------------
         // Perform a dummy action
         //-----------------------------------------
         WinAlarm(HWND_DESKTOP,ulNote);
         DosSleep(2000);
         WinAlarm(HWND_DESKTOP,ulNote);

         //-----------------------------------------
         // Notify the owner that we're done
         //-----------------------------------------
         WinPostMsg(hwndOwner,
                    MYM_ENDREQUEST,
                    MPFROMLONG(ulRequest),
                    0);
      }
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}
```

```
VOID _Optlink objectThread(PINSTDATA pidData)
{
   HAB habAnchor;
   HMQ hmqQueue;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_OBJECT,
                    objectProc,
                    0,
                    0);

   //-------------------------------------------------
   // Create the object window
   //-------------------------------------------------
   pidData->hwndObject=WinCreateWindow(HWND_OBJECT,
                                       CLS_OBJECT,
                                       "",
                                       0,
                                       0,
                                       0,
                                       0,
                                       0,
                                       HWND_OBJECT,
                                       HWND_TOP,
                                       WND_OBJECT,
                                       NULL,
                                       NULL);
   if (pidData->hwndObject!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,
                         &qmMsg,
                         NULLHANDLE,
                         0,
                         0);
      } /* endwhile */

      WinDestroyWindow(pidData->hwndObject);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
}

VOID disableMenuItems(HWND hwndMenu,BOOL bDisable)
//-------------------------------------------------
// This function disables or enables the "send
// request" menuitems.
//-------------------------------------------------
{
   SHORT sDisable;

   sDisable=bDisable?MIA_DISABLED:0;
```

```
   WinSendMsg(hwndMenu,
              MM_SETITEMATTR,
              MPFROM2SHORT(MI_NOTETHREAD,TRUE),
              MPFROM2SHORT(MIA_DISABLED,sDisable));
   WinSendMsg(hwndMenu,
              MM_SETITEMATTR,
              MPFROM2SHORT(MI_WARNINGTHREAD,TRUE),
              MPFROM2SHORT(MIA_DISABLED,sDisable));
   WinSendMsg(hwndMenu,
              MM_SETITEMATTR,
              MPFROM2SHORT(MI_ERRORTHREAD,TRUE),
              MPFROM2SHORT(MIA_DISABLED,sDisable));
}

MRESULT EXPENTRY wndProc(HWND hwndWnd,
                         ULONG ulMsg,
                         MPARAM mpParm1,
                         MPARAM mpParm2)
{
   PINSTDATA pidData;

   pidData=(PINSTDATA)WinQueryWindowPtr(hwndWnd,0);

   switch (ulMsg) {
   case WM_CREATE:
      {
         TID tidThread;

         pidData=(PINSTDATA)malloc(sizeof(INSTDATA));
         if (pidData==NULL) {
            WinAlarm(HWND_DESKTOP,WA_ERROR);
            return MRFROMSHORT(TRUE);
         } /* endif */

         WinSetWindowPtr(hwndWnd,0,(PVOID)pidData);

         pidData->ulSzStruct=sizeof(INSTDATA);
         pidData->hwndMenu=WinLoadMenu(HWND_OBJECT,
                                       NULLHANDLE,
                                       RES_CLIENT);
         pidData->hwndObject=NULLHANDLE;

         //-------------------------------------------
         // Start the object thread
         //-------------------------------------------
         tidThread=_beginthread((PFNREQ)objectThread,
                                NULL,
                                0x4000,
                                pidData);
         if (tidThread==(TID)-1) {
            WinDestroyWindow(pidData->hwndMenu);
            free(pidData);
            WinAlarm(HWND_DESKTOP,WA_ERROR);
            return MRFROMSHORT(TRUE);
         } /* endif */
      }
      break;
   case WM_DESTROY:
      WinDestroyWindow(pidData->hwndMenu);
      free(pidData);
      break;
   case WM_CONTEXTMENU:
      {
         POINTL ptlMouse;
```

```
        WinQueryPointerPos(HWND_DESKTOP,&ptlMouse);
        WinPopupMenu(HWND_DESKTOP,
                     hwndWnd,
                     pidData->hwndMenu,
                     ptlMouse.x,
                     ptlMouse.y,
                     0,
                     PU_HCONSTRAIN | PU_VCONSTRAIN |
                        PU_NONE | PU_KEYBOARD |
                        PU_MOUSEBUTTON1 |
                        PU_MOUSEBUTTON2);
     }
     break;
  case WM_COMMAND:
     switch (SHORT1FROMMP(mpParm1)) {
     case MI_NOTETHREAD:
        //-----------------------------------------
        // Disable the menu items and send the
        // request to the object window
        //-----------------------------------------
        disableMenuItems(pidData->hwndMenu,TRUE);

        WinPostMsg(pidData->hwndObject,
                   MYM_STARTREQUEST,
                   MPFROMLONG(ASYNC_NOTE),
                   MPFROMHWND(hwndWnd));
        break;
     case MI_WARNINGTHREAD:
        disableMenuItems(pidData->hwndMenu,TRUE);

        WinPostMsg(pidData->hwndObject,
                   MYM_STARTREQUEST,
                   MPFROMLONG(ASYNC_WARNING),
                   MPFROMHWND(hwndWnd));
        break;
     case MI_ERRORTHREAD:
        disableMenuItems(pidData->hwndMenu,TRUE);

        WinPostMsg(pidData->hwndObject,
                   MYM_STARTREQUEST,
                   MPFROMLONG(ASYNC_ERROR),
                   MPFROMHWND(hwndWnd));
        break;
     case MI_EXIT:
        WinPostMsg(hwndWnd,WM_CLOSE,0,0);
        break;
     default:
        return WinDefWindowProc(hwndWnd,
                                ulMsg,
                                mpParm1,
                                mpParm2);
     } /* endswitch */
     break;
  case WM_PAINT:
     {
        HPS hpsWnd;
        RECTL rclPaint;

        hpsWnd=WinBeginPaint(hwndWnd,
                             NULLHANDLE,
                             &rclPaint);
        WinFillRect(hpsWnd,&rclPaint,SYSCLR_WINDOW);
        WinEndPaint(hpsWnd);
     }
     break;
```

```
   case MYM_ENDREQUEST:
      //-----------------------------------------------
      // The object window has finished our request
      // so enable the menuitems
      //-----------------------------------------------
      disableMenuItems(pidData->hwndMenu,FALSE);
      break;
   default:
      return WinDefWindowProc(hwndWnd,
                              ulMsg,
                              mpParm1,
                              mpParm2);
   } /* endswitch */

   return MRFROMSHORT(FALSE);
}

INT main(USHORT usArgs,PCHAR apchArgs[])
{
   HAB habAnchor;
   HMQ hmqQueue;
   ULONG ulCreate;
   HWND hwndFrame;
   HWND hwndClient;
   BOOL bLoop;
   QMSG qmMsg;

   habAnchor=WinInitialize(0);
   hmqQueue=WinCreateMsgQueue(habAnchor,0);

   WinRegisterClass(habAnchor,
                    CLS_MAIN,
                    wndProc,
                    CS_SIZEREDRAW,
                    sizeof(PVOID));

   ulCreate=FCF_SYSMENU | FCF_TITLEBAR | FCF_MINMAX |
            FCF_SIZEBORDER | FCF_TASKLIST |
            FCF_SHELLPOSITION;

   if (WinQuerySysValue(HWND_DESKTOP,SV_ANIMATION)) {
      ulCreate|=WS_ANIMATE;
   } /* endif */

   hwndFrame=WinCreateStdWindow(HWND_DESKTOP,
                                WS_VISIBLE,
                                &ulCreate,
                                CLS_MAIN,
                                "Object Window",
                                0L,
                                NULLHANDLE,
                                RES_CLIENT,
                                &hwndClient);
   if (hwndFrame!=NULLHANDLE) {
      bLoop=WinGetMsg(habAnchor,
                      &qmMsg,
                      NULLHANDLE,
                      0,
                      0);
      while (bLoop) {
         WinDispatchMsg(habAnchor,&qmMsg);
         bLoop=WinGetMsg(habAnchor,
                         &qmMsg,
                         NULLHANDLE,
                         0,
                         0);
      } /* endwhile */
```

```
      WinDestroyWindow(hwndFrame);
   } /* endif */

   WinDestroyMsgQueue(hmqQueue);
   WinTerminate(habAnchor);
   return 0;
}
```

**THRD4.RC**

```
#include <os2.h>
#include "thrd4rc.h"

MENU RES_CLIENT
{
   MENUITEM "~Note thread", MI_NOTETHREAD
   MENUITEM "~Warning thread", MI_WARNINGTHREAD
   MENUITEM "~Error thread", MI_ERRORTHREAD
   MENUITEM SEPARATOR
   MENUITEM "E~xit", MI_EXIT
}
```

**THRD4RC.H**

```
#define RES_CLIENT              256
#define WND_OBJECT              257
#define MI_NOTETHREAD           258
#define MI_WARNINGTHREAD        259
#define MI_ERRORTHREAD          260
#define MI_EXIT                 261
```

**THRD4.MAK**

```
APP=THRD4

$(APP).EXE:                     $(APP).OBJ \
                                $(APP).RES
        LINK386 $(APP),$(APP),NUL,OS2386,$(APP);
        RC $(APP).RES $(APP).EXE

$(APP).RES:                     $(APP).RC \
                                $(APP)RC.H
        RC -r $(APP).RC $(APP).RES

$(APP).OBJ:                     $(APP).C \
                                $(APP)RC.H
        ICC -C+ -Gm+ -Kb+ -Ss+ $(APP).C
```

**THRD4.DEF**

```
NAME THRD4 WINDOWAPI

DESCRIPTION 'PM Threads Example 4
Copyright (c) 1993 by Larry Salomon, Jr.
All rights reserved.'

STACKSIZE 0x4000
```

Careful observation will show that *objectThread* is almost identical to *main*. The creation of the object window is done with a call to *WinCreateWindow*.

```
pidData->hwndObject=WinCreateWindow(HWND_OBJECT,
                                    CLS_OBJECT,
                                    "",
                                    0,
                                    0,
                                    0,
                                    0,
                                    0,
                                    HWND_OBJECT,
                                    HWND_TOP,
                                    WND_OBJECT,
                                    NULL,
                                    NULL);
```

What tells PM to make this an object window is that the parent (the first parameter) is the predefined constant HWND_OBJECT.

# Design Considerations

Before wrapping this topic up, let us consider the following issues.

*Who displays messages during the processing of the task?* To answer this question, we must consider the purpose of the message. If the message is event-specific within the thread (e.g., "file could not be opened"), then it makes sense to have the thread display the message, since the message is associated with a thread-specific event. However, general result messages (e.g., "printing was unsuccessful") probably are better left to the owner thread, since they usually can be grouped together in a function that checks the return information in the thread-specific structure.

*How is the thread halted because the user has requested it?* Say, for example, a user wants to print a 50-page document and then after realizing that it will take 20 minutes to complete (!), changes his or her mind. The THREADINFO structure contains a mild-mannered field *bKillThread.* The purpose of this is to inform the thread that it should halt processing and exit.

Of course, because the various thread structures are allocated dynamically and then forgotten about until the thread finishes, actually getting access to this field to set it to TRUE might be a task in itself. Also, setting this field to TRUE only *signals* the thread that it should kill itself; it is up to the thread to monitor this field so that it can stop itself if needed.

In *dispatchThread,* we ignored the issue of WM_QUIT and what should be done if it is received. While the function will kill the thread and return, what does the application do? A WM_QUIT is sent to an application only as the result of another action, whether it is the default processing for WM_CLOSE or because the system is shutting down. In any case, usually it can be safely assumed that, if this message is received, the application should quit as soon as it is safely possible.

# Appendix A

# Window Messages

### PL_ALTERED

This message is broadcast to all frame window when the user has altered system settings.

**Parameter 1:**
HINI — Handle to new user profile
**Parameter 2:**
HINI — Handle to new system profile
**Reply:**
ULONG — Reserved, 0

### WM_ACTIVATE

This message is sent from the system when a window is being activated or deactivated.

**Parameter 1:**
USHORT — Active? TRUE:FALSE
**Parameter 2:**
HWND — If Parameter 1 is TRUE, this is the window being activated, else this is the window being deactivated
**Reply:**
ULONG — Reserved, 0

### WM_APPTERMINATENOTIFY

This message is posted when a child application is terminated.

**Parameter 1:**
HAPP — Handle to terminating application
**Parameter 2:**
ULONG — Return code from terminating application
**Reply:**
ULONG — Reserved, 0

## WM_ADJUSTWINDOWPOS

This message is sent when a window is sized or positioned by *WinSetWindowPos*.

**Parameter 1:**

PSWP — Pointer to new swap structure

```
typedef struct _SWP
{
   ULONG   fl;
   LONG    cy;
   LONG    cx;
   LONG    y;
   LONG    x;
   HWND    hwndInsertBehind;
   HWND    hwnd;
   ULONG   ulReserved1;
   ULONG   ulReserved2;
} SWP;
typedef SWP *PSWP;
```

**Parameter 2:**

ULONG — Reserved, 0

**Reply:**

ULONG — Move status

| | |
|---|---|
| 0 | No changes made |
| SWP_MINIMIZED | Window is minimized |
| SWP_MAXIMIZED | Window is maximized |
| SWP_RESTORED | Window is restored |
| SWP_ACTIVATE | Window is activated |
| SWP_DEACTIVATE | Window is deactivated |

## WM_BEGINDRAG

This message is sent when a user starts a drag operation.

**Parameter 1:**

POINTS — Pointer position

```
typedef struct _POINTS          /* pts */
{
   SHORT x;
   SHORT y;
} POINTS;
typedef POINTS *PPOINTS;
```

**Parameter 2:**

USHORT — Input device

| | |
|---|---|
| TRUE | Input came from mouse |
| FALSE | Input came from keyboard |

**Reply:**

BOOL — Message processed? TRUE: FALSE

## WM_BEGINSELECT

This message is sent when the user starts a selection operation.

**Parameter 1:**

| | |
|---|---|
| USHORT | Input device |
| TRUE | Input came from mouse |
| FALSE | Input came from keyboard |

**Parameter 2:**

| | |
|---|---|
| POINTS | Pointer position |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON1CLICK

This message is sent when the user clicks on mouse button 1.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

Parameter 2:

| | |
|---|---|
| USHORT | Hit test result |
| HT_NORMAL | The message belongs to this window |
| HT_TRANSPARENT | The part of the window underneath the mouse is not visible; keep checking other windows |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON2CLICK

This message is sent when the user clicks on mouse button 2.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON3CLICK

This message is sent when the user clicks on mouse button 3.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON1DBCLK

This message is sent when the user double-clicks on mouse button 1.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON2DBCLK

This message is sent when the user double-clicks on mouse button 2.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_BUTTON3DBCLK

This message is sent when the user double-clicks on mouse button 3.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON1DOWN

This message is sent when the user presses mouse button 1 down.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON1MOTIONEND

This message is sent when the user finishes a drag operation using mouse button 1.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON1MOTIONSTART

This message is sent when the user starts a drag operation using mouse button 1.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON2DOWN

This message is sent when the user presses mouse button 2 down.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

### WM_BUTTON2MOTIONEND

This message is sent when the user finishes a drag operation using mouse button 2.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

### WM_BUTTON2MOTIONSTART

This message is sent when the user starts a drag operation using mouse button 2.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

### WM_BUTTON3DOWN

This message is sent when the user presses mouse button 3 down.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

### WM_BUTTON3MOTIONEND

This message is sent when the user finishes a drag operation using mouse button 3.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON3MOTIONSTART

This message is sent when the user starts a drag operation using mouse button 3.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON1UP

This message is sent when the user releases mouse button 1.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_BUTTON2UP

This message is sent when the user releases mouse button 2.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed ? TRUE: FALSE

## WM_BUTTON3UP

This message is sent when the user releases mouse button 3.

**Parameter 1:**

| | |
|---|---|
| POINTS | Mouse position |

**Parameter 2:**

| | |
|---|---|
| USHORT | Hit test result |
| USHORT | Keyboard control code |

**Reply:**

| | |
|---|---|
| BOOL | Message processed? TRUE: FALSE |

## WM_CALCFRAMERECT

This message is sent when the window size is calculated using *WinCalcFrameRect.*

**Parameter 1:**

| | |
|---|---|
| PRECTL | Window size |

```
typedef struct _RECTL          /* rcl */
{
   LONG  xLeft;
   LONG  yBottom;
   LONG  xRight;
   LONG  yTop;
} RECTL;
typedef RECTL *PRECTL;
```

**Parameter 2:**

| | |
|---|---|
| USHORT | Frame indicator |

| | |
|---|---|
| TRUE | calculate client size; prclWindow is size of frame |
| FALSE | calculate frame size; prclWindow is size of client |

**Reply**:

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

## WM_CALCVALIDRECTS

This message is sent from the system to determine which window region needs to be invalidated if the window is moved or sized.

**Parameter 1:**

| | |
|---|---|
| PRECTL | Points to a RECTL that contains the window size prior to resizing. |
| PRECTL | Points to a RECTL that contains the window size after resizing |

**Parameter 2:**

| | |
|---|---|
| PSWP | New window position |

```
typedef struct _SWP    /* swp */
{
   ULONG   fl;
   LONG    cy;
   LONG    cx;
   LONG    y;
   LONG    x;
   HWND    hwndInsertBehind;
   HWND    hwnd;
   ULONG   ulReserved1;
   ULONG   ulReserved2;
} SWP;
typedef SWP *PSWP;
```

**Reply:**

| | |
|---|---|
| USHORT | How to align the window; these values can be OR'ed together |
| CVR_ALIGNLEFT | The valid window region is aligned along with the left edge of the window |
| CVR_ALIGNBOTTOM | The valid window region is aligned along with the bottom edge of the window |
| CVR_ALIGNTOP | The valid window region is aligned along with the top edge of the window |
| CVP_ALIGNRIGHT | The valid window region is aligned along with the right edge of the window |
| CVR_REDRAW | The whole window is invalidated |
| 0 | Use the values specified in the second PRECTL |

## WM_CHAR

This message is sent to indicate a key has been pressed.

**Parameter 1:**

| | |
|---|---|
| USHORT | Keyboard control codes |
| KC_CHAR | Character key was hit, value is in usCharCode |
| KC_SCANCODE | Value of key is in KC_SCANCODE |
| KC_VIRTUALKEY | Indicates a virtual key was hit, value is in usVKeyCode |
| KC_KEYUP | Key is released |
| KC_PREVDOWN | Usually precedes the KC_KEYUP flag |
| KC_DEADKEY | Key was a dead key |
| KC_COMPOSITE | Key was a combination of the previous dead key and this key |
| KC_INVALIDCOMP | Key was an invalid combination |
| KC_LONEKEY | A key was pressed and released, and no other keys were involved |
| KC_SHIFT | The SHIFT key was pressed |
| KC_ALT | The ALT key was pressed |
| KC_CTRL | The CTRL key was pressed |
| UCHAR | Number of times key was pressed |

**Parameter 2:**

| | |
|---|---|
| USHORT | Character code |
| USHORT | Virtual key codes |

**Reply:**

| | |
|---|---|
| BOOL | Processed? TRUE: FALSE |

## WM_CHORD

This message is sent when the user presses both mouse button 1 and 2 at the same time.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
USHORT Hit test result
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_CLOSE

This message is sent to a frame when the user is closing the window.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0
*Note:* If you intercept this message, make sure to post a WM_QUIT to the appropriate window. This is what the default window procedure does.

## WM_COMMAND

This message is sent from a control to its owner whenever it has to notify its owner about a significant event.

**Parameter 1:**
USHORT Application defined ID of item that generated message
**Parameter 2:**
USHORT Can be one of the following:

| | |
|---|---|
| CMDSRC_PUSHBUTTON | Pushbutton-generated message |
| CMDSRC_MENU | Menu-generated message |
| CMDSRC_ACCELERATOR | Accelerator key-generated message |
| CMDSRC_FONTDLG | Font dialog-generated message |
| CMDSRC_OTHER | Some other type of control-generated message |

USHORT Input device

| | |
|---|---|
| TRUE | Input came from mouse |
| FALSE | Input came from keyboard |

**Reply:**
ULONG Reserved, 0

### WM_CONTEXTMENU

This message is sent when the user performs an operation that initiates a popup menu.

**Parameter 1:**
USHORT Can be one of the following:

TRUE Operation was performed by pointer
FALSE Operation was performed by keystroke
**Parameter 2:**
POINTS Mouse position if *usPoint* == TRUE
**Reply:**
BOOL Processed? TRUE: FALSE

### WM_CONTROL

This message is sent to a control's owner in order to signal a significant event that has occurred to the control.

**Parameter 1:**
USHORT Control ID
USHORT Control notify code
**Parameter 2:**
ULONG Control specific information
**Reply:**
ULONG Reserved, 0

### WM_CONTROLPOINTER

This message is sent to a control's owner when the user moves the mouse over the control window.

**Parameter 1:**
USHORT Control ID
**Parameter 2:**
HPOINTER Current mouse pointer
**Reply:**
HPOINTER Pointer to new mouse pointer
*Note:* This message gives you the opportunity to change the standard mouse pointer to something else, for instance, an hourglass pointer. Trap the message when in a wait situation, and return the HPOINTER you want the pointer to change to.

## WM_CREATE

This message is sent to a window at the first stage of its creation.

**Parameter 1:**
PVOID Data passed from *WinCreateWindow*
**Parameter 2:**
PCREATESTRUCT Pointer to a create structure

```
typedef struct _CREATESTRUCT
{
   PVOID   pPresParams;
   PVOID   pCtlData;
   ULONG   id;
   HWND    hwndInsertBehind;
   HWND    hwndOwner;
   LONG    cy;
   LONG    cx;
   LONG    y;
   LONG    x;
   ULONG   flStyle;
   PSZ     pszText;
   PSZ     pszClass;
   HWND    hwndParent;
} CREATESTRUCT;
typedef CREATESTRUCT *PCREATESTRUCT;
```

**Reply:**
BOOL Continue creating window? TRUE: FALSE
*Note:* This message is not a good place to do much except transfer data from *WinCreateWindow*. At this point in time a window has no shape, size, or focus, so most changes to any of these qualities will fail. A better idea is to post yourself a message from the WM_CREATE processing. Do all the initialization in the user-defined message.

## WM_DESTROY

This message is sent when a window is being destroyed.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0
*Note:* This is the last message that is sent to a window before it is destroyed. If you have allocated memory for a window word, this is the place to free it.

## WM_DRAWITEM

Sent to the owner of a control when an item with *S_OWNERDRAW is to be drawn.

**Parameter 1:**
USHORT Window ID
**Parameter 2:**
POWNERITEM Pointer to OWNERITEM structure

```
typedef struct _owneritem
{
   HWND    hwnd;
   HPS     hps;
   ULONG   fsState;
   ULONG   fsAttribute;
   ULONG   fsStateOld;
   ULONG   fsAttributeOld;
   RECTL   rclItem;
   LONG    idItem;
   ULONG   hItem;
} OWNERITEM;
typedef OWNERITEM *POWNERITEM;
```

**Reply**:
BOOL Drawn? TRUE:FALSE

## WM_ENABLE

This message is sent to enable, or disable, a window.

**Parameter 1:**
USHORT Enable window? TRUE: FALSE
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0
*Note:* This is the recommended way to prevent a user from selecting a control. A control window should not be permanently disabled, only disabled while it is not available.

## WM_ENDDRAG

This message is sent when a drag operation is completed.

**Parameter 1:**
USHORT Mouse indicator

TRUE Message came from pointer input
FALSE Message came from keyboard input
**Parameter 2:**
POINTS Mouse position
**Reply:**
ULONG Message processed? TRUE: FALSE

## WM_ENDSELECT

This message is sent when a select operation is completed.

**Parameter 1:**
USHORT Mouse indicator

TRUE Message came from pointer input
FALSE Message came fromkeyboard input
**Parameter 2:**
POINTS Mouse position
**Reply:**
ULONG Message processed? TRUE: FALSE

## WM_ERASEWINDOW

This message is sent to a window after a region has been invalidated, but before it has been painted.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Window region erased? TRUE: FALSE

## WM_ERROR

This message is sent when an error is detected.

**Parameter 1:**
USHORT Error code (list in PMERR.H)
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

## WM_FOCUSCHANGE

This message is sent when a window is losing focus or gaining focus.

**Parameter 1:**
HWND Window handle
**Parameter 2:**
USHORT Focus flag

| | |
|---|---|
| TRUE | Window is gaining focus, hwndwindow is window losing focus |
| FALSE | Window is losing focus, hwndwindow is window gaining focus |

| | |
|---|---|
| USHORT | Focus changing flags |

| | |
|---|---|
| FC_NOSETFOCUS | A WM_SETFOCUS message is not sent to the window receiving focus |
| FC_NOLOSEFOCUS | A WM_SETFOCUS message is not sent to the window losing focus |
| FC_NOSETACTIVE | A WM_SETACTIVE message is not sent to the window becoming active |
| FC_NOLOSEACTIVE | A WM_SETACTIVE message is not sent to the window being deactivated |
| FC_NOSETSELECTION | A WM_SETSELECTION message is not sent to the window being selected |
| FC_NOLOSESELECTION | A WM_SETSELECTION message is not sent to the window being deselected |
| FC_NOBRINGTOTOP | No window is brought to the top |
| FC_NOBRINGTOTOPFIRSTWINDOW | The first frame window is not brought to the top |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_FORMATFRAME

This message is sent to the frame to adjust all the frame controls and the client window.

**Parameter 1:**

| | |
|---|---|
| PSWP | Array of sizes and positions of controls |

**Parameter 2:**

| | |
|---|---|
| PRECTL | Client window size |

**Reply:**

| | |
|---|---|
| USHORT | Count of SWP structures returned in swpSizes |

*Note:* This message is used to adjust the sizes and positions of the controls on the frame. If you subclass the frame window and add controls to the frame, you need to intercept this message and add the SWP structures of your controls. You also will need to intercept the WM_QUERYFRAMECTLCOUNT message and increment the return value by the number of the controls you are adding. It is this value that controls the number of SWP structures in the WM_FORMATFRAME. Also, if you put a sizing border on a dialog box, the dialog box does not receive a WM_SIZE message; instead it receives a WM_FORMATFRAME.

## WM_HELP

This message is sent to the owner of a control when the user has hit the help key.

**Parameter 1:**
USHORT Command value
**Parameter 2:**
USHORT Control type

| | |
|---|---|
| CMDSRC_PUSHBUTTON | Push-button was source of help message. Parameter 1 is the id of the push-button. |
| CMDSRC_MENU | Menu was source of help message. Parameter 1 is the id of the menu item. |
| CMDSRC_ACCELERATOR | Accelerator key was source of help message. Parameter 1 is the accelerator command value. |
| CMDSRC_OTHER | Other source. |

USHORT Input device

| | |
|---|---|
| TRUE | Input came from mouse |
| FALSE | Input came from keyboard |

**Reply:**
ULONG Reserved, 0

### WM_HITTEST

This message is used to find the window that the user was responding to when mouse input occurred.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Can be one of the following:

| | |
|---|---|
| HT_NORMAL | Message belongs to this window |
| HT_TRANSPARENT | Message does not belong to this window, keep checking |
| HT_DISCARD | Discard message |
| HT_ERROR | Discard message, message resulted from a button down message on a disabled window |

### WM_HSCROLL

This message is sent to the owner of a horizontal scroll bar when the user initiated some scrolling action.

**Parameter 1:**
USHORT Scroll bar ID

**Parameter 2:**

| | |
|---|---|
| SHORT | Slider position |
| USHORT | Command |

| | |
|---|---|
| SB_LINELEFT | The user clicked on left arrow, or a VK_LEFT key was pressed |
| SB_LINERIGHT | The user clicked on right arrow, or a VK_RIGHT key was pressed |
| SB_PAGELEFT | The user clicked to the left of the slider or VK_PAGELEFT key was pressed |
| SB_PAGERIGHT | The user clicked to the right of the slider or VK_PAGERIGHT key was pressed |
| SB_SLIDERPOSITION | This is the final position of the slider |
| SB_SLIDERTRACK | The user used the mouse to change the position of the slider |
| SB_ENDSCROLL | The user has finished scrolling |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_INITDLG

Sent when a dialog box is being created.

**Parameter 1:**

| | |
|---|---|
| HWND | Control window to receive focus |

**Parameter 2:**

| | |
|---|---|
| PVOID | Application defined data passed by *WinLoadDlg*, *WinCreateDlg*, or *WinDlgBox* |

**Reply:**

| | |
|---|---|
| BOOL | Focus set indicator |

| | |
|---|---|
| TRUE | Focus window has been changed |
| FALSE | Focus window has not been changed |

## WM_INITMENU

This message is sent when a window is becoming active.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu control ID |

**Parameter 2:**

| | |
|---|---|
| HWND | Menu window handle |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_JOURNALNOTIFY

This message is sent during journal playback.

**Parameter 1:**

| | |
|---|---|
| ULONG | Can be one of the following: |
| | |
| JRN_QUEUESTATUS | Journal the queue status |
| JRN_PHYSKEYSTATE | Journal the physical key state |

**Parameter 2:**

| | |
|---|---|
| if Parameter 1 is JRN_QUEUESTATUS | |
| USHORT | Queue status |
| if Parameter 1 is JRN_PHYSKEYSTATE | |
| USHORT | Key scan code |
| USHORT | Key state |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_MEASUREITEM

This message is sent to the owner of a control in order to determine the height and width of the specified item.

**Parameter 1:**

| | |
|---|---|
| SHORT | ID of control window |

**Parameter 2:**

| | |
|---|---|
| ULONG | Control specific information |

**Reply:**

| | |
|---|---|
| SHORT | Height of control item of interest |
| SHORT | Width of control item of interest |

### WM_MENUEND

This message is sent when a menu is ending.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu control ID |

**Parameter 2:**

| | |
|---|---|
| HWND | Menu window handle |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_MENUSELECT

This message is sent when the user has selected a new menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Selected menuitem ID |
| USHORT | Post command flag |

| | |
|---|---|
| TRUE | A command message will be posted to the owner |
| FALSE | No message will be posted |

**Parameter 2:**

| | |
|---|---|
| HWND | Menu window handle |

**Reply:**

| | |
|---|---|
| BOOL | Post message? TRUE: FALSE |

## WM_MINMAXFRAME

This message is sent when a frame is being maximized, minimized, or restored.

**Parameter 1:**

| | |
|---|---|
| PSWP | Window position |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Process message? TRUE:FALSE |

## WM_MOUSEMAP

This message is sent by application that wish to remap mouse messages in the Presentation Manager input queue. The mapping will be in affect until another WM_MOUSEMAP message is received that remaps the mouse buttons.

**Parameter 1:**

| | |
|---|---|
| ULONG | Indicates the physical mouse button to map. |

**Parameter 2:**

| | |
|---|---|
| ULONG | Indicates the button specified in Parameter is to be mapped to. |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0. |

## WM_MSGBOXDISMISS

This message is sent when a non-modal message box is being dismissed.

**Parameter 1:**

| | |
|---|---|
| HWND | Non-modal message box window handle. |

**Parameter 2:**

| | |
|---|---|
| ULONG | ID of the selected button in message box. |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0. |

## WM_MSGBOXINIT

This message is sent to the owner of a non-modal message box has been created.

**Parameter 1:**

| | |
|---|---|
| HWND | Non-modal message box window handle. |

**Parameter 2:**
LONG ID of the message box.
**Reply:**
ULONG Reserved, 0.

### WM_MOUSEMOVE

This mouse has moved.

**Parameter 1:**
POINTS Mouse position
**Parameter 2:**
USHORT Hit test result
USHORT Keyboard control code
**Reply:**
BOOL Message processed? TRUE: FALSE

### WM_MOVE

This message is sent when a window has moved.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

*Note:* This message is sent only if the window has the style CS_MOVENOTIFY.

### WM_NEXTMENU

This message is sent to the owner of the menu to indicate either the beginning or the end of the menu has been reached.

**Parameter 1:**
HWND Menu window handle
**Parameter 2:**
USHORT At beginning of menu? TRUE:FALSE
**Reply:**
HWND New menu window handle

### WM_NULL

This is a NULL message used for an application-defined purpose.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

## WM_OPEN

This message is sent when a user opens a window.

**Parameter 1:**
USHORT Input device

TRUE Input came from mouse
FALSE Input came from keyboard

**Parameter 2:**
POINTS Mouse position
**Reply:**
BOOL Message processed? TRUE: FALSE

## WM_PACTIVATE

This is an NLS message sent when a WM_ACTIVATE message is received.

**Parameter 1:**
USHORT Window activation flag

TRUE Window was made active
FALSE Window was deactivated
**Parameter 2:**
HWND Window handle

Parameter 1 == TRUE Window handle of window being made active
Parameter 1 == FALSE Window handle of window being deactivated
**Reply:**
ULONG Reserved, 0

## WM_PAINT

This message is sent when a window needs painting.

**Parameter 1:**
ULONG Reserved, 0

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_PCONTROL

This message is an NLS message sent when a WM_CONTROL message is received.

**Parameter 1:**
USHORT Control window ID
USHORT Notification code
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_PPAINT

This is an NLS message sent when a WM_PAINT message is received.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_PRESPARAMCHANGED

This message is sent when a window's Presentation Parameters are changed.

**Parameter 1:**
ULONG Attribute type ID
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_PSETFOCUS

This is an NLS message sent when a WM_SETFOCUS is received.

**Parameter 1:**
HWND Focus window handle

**Parameter 2:**

| | |
|---|---|
| USHORT | Can be one of the following: |
| TRUE | Window is gaining focus, Parameter 1 is window losing focus |
| FALSE | Window is losing focus, Parameter 1 is window gaining focus |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_PSIZE

This is an NLS message sent when a WM_SIZE is received.

**Parameter 1:**

| | |
|---|---|
| SHORT | Old window width |
| SHORT | Old window height |

**Parameter 2:**

| | |
|---|---|
| SHORT | New window width |
| SHORT | New window height |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_PSYSCOLORCHANGE

This is an NLS message sent after a WM_SYSCOLORCHANGE message is received.

**Parameter 1:**

| | |
|---|---|
| ULONG | Can be one of the following: |
| LCOL_RESET | The colors are reset to the default |
| LCOL_PURECOLOR | Color-dithering is not allowed |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_QUERYACCELTABLE

This message is sent to retrieve a handle to a window's accelerator table.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| HACCEL | Accelerator table handle, or NULLHANDLE if not available |

## WM_QUERYCONVERTPOS

This message is sent to determine whether to begin DBCS character conversion.

**Parameter 1:**
PRECTL — Cursor position
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
USHORT — Can be one of the following:

| | |
|---|---|
| QCP_CONVERT | Conversion can be performed |
| QCP_NOCONVERT | Conversion should not be performed |

## WM_QUERYHELPINFO

This message is sent to query the help instance associated with a frame window.

**Parameter 1:**
ULONG — Reserved, 0
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
HWND — Help instance window handle

## WM_QUERYTRACKINFO

This message is sent after a frame receives a WM_TRACKFRAME.

**Parameter 1:**
USHORT — Tracking flags

| | |
|---|---|
| TF_TOP | Track the top of the rectangle |
| TF_BOTTOM | Track the bottom of the rectangle |
| TF_RIGHT | Track the right side of the rectangle |
| TF_LEFT | Track the left side of the rectangle |
| TF_MOVE | Track all sides |
| TF_SETPOINTERPOS | Reposition the mouse pointer |
| TF_GRID | Rectangle is snapped to the grip coordinates in cxGrid and cyGrid |
| TF_STANDARD | The grid coordinates are multiples of the border width and height |
| TF_ALLINBOUNDARY | No part of the tracking is outside the rclBoundary rectangle |
| TF_PARTINBOUNDARY | Some part of the tracking is inside the rclBoundary rectangle |
| TF_VALIDATETRACKRECT | The tracking rectangle is validated |

**Parameter 2:**
PTRACKINFO — Pointer to track info structure

```
typedef struct _TRACKINFO    /* ti */
{
   LONG    cxBorder;
   LONG    cyBorder;
   LONG    cxGrid;
   LONG    cyGrid;
   LONG    cxKeyboard;
   LONG    cyKeyboard;
   RECTL   rclTrack;
   RECTL   rclBoundary;
   POINTL  ptlMinTrackSize;
   POINTL  ptlMaxTrackSize;
   ULONG   fs;
} TRACKINFO;
typedef TRACKINFO *PTRACKINFO;
```

**Reply:**

BOOL Continue moving or sizing: TRUE: FALSE

*Note:* This message can be used to limit the sizing of a frame window. Adjust the max and min track size POINTL structures and add the TF_ALLINBOUNDARY flag. The IMAGE example of the Developer's Toolkit gives a nice example of the proper processing of the message.

## WM_QUERYWINDOWPARAMS

This message is used to query a window's Presentation Parameters

**Parameter 1:**

PWNDPARAMS Pointer to WNDPARAMS structure

```
typedef struct _WNDPARAMS
{
   ULONG   fsStatus;
   ULONG   cchText;
   PSZ     pszText;
   ULONG   cbPresParams;
   PVOID   pPresParams;
   ULONG   cbCtlData;
   PVOID   pCtlData;
} WNDPARAMS;
typedef WNDPARAMS *PWNDPARAMS;
```

**Parameter 2:**

ULONG Reserved, 0

**Reply:**

BOOL Successful? TRUE: FALSE

## WM_QUIT

This message is used to tell an application to terminate itself.

**Parameter 1:**

ULONG Reserved, 0

**Parameter 2:**

ULONG Reserved, 0

**Reply:**

ULONG Reserved, 0

*Note:* This is the message that causes *WinGetMsg* to return FALSE and end the message processing loop. If you processing this message, do not pass this through to *WinDefWindowProc*.

### WM_REALIZEPALETTE

This message is sent when another application changes the display hardware color palette.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_SAVEAPPLICATION

This message is sent to the application to give the application a chance to save the current state.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_SEM1

This message is sent from the application for its own use.

**Parameter 1:**
ULONG This is a collection of this variable from any other existing WM_SEM1 messages in the queue, OR'ed together
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0
*Note:* This message has a higher priority than any other message, and will be retrieved from the message queue first.

### WM_SEM2

This message is sent from the application for its own use.

**Parameter 1:**
ULONG — This is a collection of this variable from any other existing WM_SEM2 messages in the queue, OR'ed together
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
ULONG — Reserved, 0
*Note:* This message has a higher priority than WM_PAINT or WM_TIMER, but lower than the user I/O messages.

## WM_SEM3

This message is sent from the application for its own use.

**Parameter 1:**
ULONG — This is a collection of this variable from any other existing WM_SEM3 messages in the queue, OR'ed together
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
ULONG — Reserved, 0
*Note:* This message has a higher priority than WM_PAINT, but lower than almost all other messages.

## WM_SEM4

This message is sent from the application for its own use.

**Parameter 1:**
ULONG — This is a collection of this variable from any other existing WM_SEM4 messages in the queue, OR'ed together
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
ULONG — Reserved, 0
*Note:* This message has a lower priority than all other messages.

## WM_SETACCELTABLE

This message is sent to associate an accelerator table to a window.

**Parameter 1:**
HACCEL — Handle to new accelerator table
**Parameter 2:**
ULONG — Reserved, 0

**Reply:**
BOOL Successful? TRUE: FALSE

### WM_SETFOCUS

This message is sent when a window is losing or gaining focus.

**Parameter 1:**
HWND Window handle
**Parameter 2:**
USHORT Can be one of the following:

TRUE Window is gaining focus, Parameter 1 is window losing focus
FALSE Window is losing focus, Parameter 1 is window gaining focus
**Reply:**
ULONG Reserved, 0

### WM_SETHELPINFO

This message is sent to associate a help instance with a window.

**Parameter 1:**
HWND Help instance handle
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### WM_SETSELECTION

This message is sent when a window is selected or deselected.

**Parameter 1:**
USHORT Selection flag

TRUE Window is selected
FALSE Window is deselected
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_SETWINDOWPARAMS

This message is sent to set the window's Presentation Parameters.

**Parameter 1:**
PWNDPARAMS Pointer to WNDPARAMS structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### WM_SHOW

This message is sent when a window is to be made visible or invisible.

**Parameter 1:**
USHORT Visible flag

TRUE Show window
FALSE Hide window
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### WM_SINGLESELECT

This message is sent when the user selects an object.

**Parameter 1:**
USHORT Mouse indicator

TRUE Selection came from mouse
FALSE Selection came from keyboard
**Parameter 2:**
POINTS Mouse position
**Reply:**
BOOL Message processed? TRUE: FALSE

### WM_SIZE

This message is sent when a window changes its size.

**Parameter 1:**
SHORT Old window width
SHORT Old window height
**Parameter 2:**
SHORT New window width
SHORT New window height

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_SUBSTITUTESTRING

This message is sent from the *WinSubstituteStrings* call.

**Parameter 1:**

| | |
|---|---|
| USHORT | Indicates a value corresponding to the decimal character in the substitution phrase. |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0. |

**Reply:**

| | |
|---|---|
| PSZ | String to be substituted. |

### WM_SYSCOLORCHANGE

This message is sent to all main windows when *WinSetSysColors* has been used to change the system colors.

**Parameter 1:**

| | |
|---|---|
| ULONG | Options specified in *WinSetSysColors* |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0. |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0. |

### WM_SYSCOMMAND

This message is sent from a control to its owner whenever it has to notify its owner about a significant event.

**Parameter 1:**

| | |
|---|---|
| USHORT | One of the following: |
| SC_SIZE | Window has been sized. |
| SC_MOVE | Window has been moved. |
| SC_MINIMIZE | Window has been minimized. |
| SC_MAXIMIZE | Window has been maximized. |
| SC_CLOSE | Window has been closed |
| SC_NEXT | Moves the active window to the next main window. |
| SC_APPMENU | Initiates the menu control with the ID, FID_MENU. |
| SC_SYSMENU | Initiates the menu control with the ID, FID_SYSMENU. |
| SC_RESTORE | Restores a maximized window to its previous size and position. |
| SC_NEXTFRAME | Moves the active window to the next frame window that is a child of the desktop. |
| SC_NEXTWINDOW | Moves the active window to the next window with the same owner. |
| SC_TASKMANAGER | Activate the Task List. |

| | |
|---|---|
| SC_HELPKEYS | Sends a HM_KEYS_HELP to the help manager object window. |
| SC_HELPINDEX | Sends a HM_HELP_INDEX to the help manager object window. |
| SC_HELPEXTENDED | Sends a HM_EXT_HELP to the help mananger object window. |
| SC_HIDE | Hides the window. |

**Parameter 2:**

USHORT Can be one of the following:

| | |
|---|---|
| CMDSRC_PUSHBUTTON | Pushbutton-generated message |
| CMDSRC_MENU | Menu-generated message |
| CMDSRC_ACCELERATOR | Accelerator key-generated message |
| CMDSRC_FONTDLG | Font dialog-generated message |
| CMDSRC_OTHER | Some other type of control-generated message |

USHORT Can be one of the following:

| | |
|---|---|
| TRUE | Mouse was used to initiate event |
| FALSE | Keyboard was used to initiate event |

**Reply:**

ULONG Reserved, 0

## WM_SYSVALUECHANGED

This message is sent when a system value (SV_*) has been changed.

**Parameter 1:**

USHORT ID of first value that has changed

**Parameter 2:**

USHORT ID of last value that has changed

**Reply:**

ULONG Reserved, 0

*Note:* All values between *usFirstValue* and *usLastValue* will have been changed.

## WM_TEXTEDIT

This message is sent when a user is doing direct edit.

**Parameter 1:**

USHORT Can be one of the following:

| | |
|---|---|
| TRUE | Selection came from mouse |
| FALSE | Selection came from keyboard |

**Parameter 2:**

POINTS Mouse position

**Reply:**

BOOL Message processed? TRUE: FALSE

### WM_TIMER

This message is sent when a timer expires.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | Timer ID |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| ULONG | Reserved, 0 |

### WM_TRACKFRAME

This message is sent when a frame window is moved or sized.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | TF_* flags |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| BOOL | Successful? TRUE: FALSE |

### WM_TRANSLATEACCEL

This message is sent to determine if a character key is an accelerator key.

| | |
|---|---|
| **Parameter 1:** | |
| PQMSG | QMSG structure |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| BOOL | Character exists in accelerator table? TRUE: FALSE |

### WM_TRANSLATEMNEMONIC

This message is sent after a WM_TRANSLATEACCEL message.

| | |
|---|---|
| **Parameter 1:** | |
| PQMSG | Pointer to a QMSG structure |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| BOOL | Key is an accelerator key? TRUE: FALSE |

## WM_UPDATEFRAME

This message is sent when a frame window needs updating.

**Parameter 1:**
ULONG The FCF_* flags
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Processed message? TRUE:FALSE
*Note:* This message should be sent by the application when the frame window is customized by adding or modifying controls.

## WM_VRNDISABLED

This message is sent when a window's visible region has been disabled. This can indicate the window is being sized or *WinLockWindowUpdate* was called.

**Parameter 1:**
VOID Reserved.
**Parameter 2:**
VOID Reserved.
**Reply:**
ULONG Reserved, 0.

## WM_VRNENABLED

This message is sent when a window's visible region has been unlocked.

**Parameter 1:**
BOOL Has the visible region been altered? TRUE: FALSE.
**Parameter 2:**
VOID Reserved.
**Reply:**
ULONG Reserved, 0.

## WM_VSCROLL

This message is sent when a user modifies the position of a vertical scroll bar.

**Parameter 1:**
USHORT Scroll bar ID
**Parameter 2:**
SHORT Slider position
USHORT Command

| | |
|---|---|
| SB_LINEUP | The user clicked on up arrow, or a VK_UP key was pressed |
| SB_LINEDOWN | The user clicked on down arrow, or a VK_DOWN key was pressed |
| SB_PAGEUP | The user clicked to the top of the slider, or VK_PAGEUP key was pressed |
| SB_PAGEDOWN | The user clicked to the bottom of the slider, or VK_PAGEDOWN key was pressed |
| SB_SLIDERPOSITION | This is the final position of the slider |
| SB_SLIDERTRACK | The user used the mouse to change the position of the slider |
| SB_ENDSCROLL | The user has finished scrolling |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_WINDOWPOSCHANGED

This message is sent when a window's position is changed.

**Parameter 1:**

| | |
|---|---|
| PSWP | Pointer to two SWP structures:the first is the new structure, the second is the old structure |

**Parameter 2:**

| | |
|---|---|
| ULONG | AWF_* flags returned from WM_ADJUSTWINDOWPOS |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## Dialog Box Messages

### WM_INITDLG

This message is sent to the dialog at the time of creation.

**Parameter 1:**

| | |
|---|---|
| HWND | Handle of control window that will receive focus |

**Parameter 2:**

| | |
|---|---|
| PCREATEPARAMS | Application defined data passed through dialog box function |

**Reply:**

| | |
|---|---|
| BOOL | Focus window is to be changed? TRUE: FALSE |

### WM_MATCHMNEMONIC

This message is sent by the dialog to the control to determine whether a mnemonic belongs to that control.

**Parameter 1:**

| | |
|---|---|
| USHORT | Character to match |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Successful match? TRUE: FALSE |

### WM_QUERYDLGCODE

This message is sent by the dialog box to query what kinds of controls it has.

**Parameter 1:**

| | |
|---|---|
| PQMSG | Message queue |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Can be one of the following: |
| | |
| DLGC_ENTRYFIELD | Controls is an entryfield. |
| DLGC_BUTTON | Control is a button. |
| DLGC_CHECKBOX | Control is a checkbox. |
| DLGC_RADIOBUTTON | Control is a radio-button. |
| DLGC_STATIC | Control is a static text field. |
| DLGC_DEFAULT | Control is a default control. |
| DLGC_PUSHBUTTON | Control is a push-button. |
| DLGC_SCROLLBAR | Control is a scrollbar. |
| DLGC_MENU | Control is a menu. |
| DLGC_MLE | Control is an MLE. |

## Button Messages

### WM_CONTROL

This messages occurs when a control has to notify its owner of an event. The message is sent, whereas the WM_COMMAND message is posted.

**Parameter 1:**

| | |
|---|---|
| USHORT | Button control ID |
| USHORT | Can be one of the following: |
| | |
| BN_CLICKED | Button has been pressed |
| BN_DBLCLICKED | Button has been double clicked. |
| BN_PAINT | Button requires painting |

**Parameter 2:**

| | |
|---|---|
| ULONG | Specific information; normally the window handle of the button; however, when Parameter 1 equals BN_PAINT, this parameter is PUSERBUTTON |

```
typedef struct _USERBUTTON     /* ubtn */
      {
         HWND    hwnd;
         HPS     hps;
         ULONG   fsState;
         ULONG   fsStateOld;
      } USERBUTTON;
      typedef USERBUTTON *PUSERBUTTON;
```

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### BM_CLICK

An application sends this message to simulate the clicking of a button.

**Parameter 1:**

| | |
|---|---|
| USHORT | Up and down indicator |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### BM_QUERYCHECK

This message returns whether a button control is checked or not.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| USHORT | Button checked? TRUE:FALSE |

### BM_QUERYCHECKINDEX

This message returns the index of a checked radio button with a group of radio buttons (zero-based). A group is defined by the style WS_GROUP.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| SHORT | Radio button index |

### BM_QUERYHILITE

This message returns whether a button control is highlighted or not.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Button highlighted? TRUE:FALSE |

### BM_SETCHECK

This message checks or unchecks a button control.

**Parameter 1:**
USHORT — Check button? TRUE:FALSE
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
USHORT — Previous state of button

### BM_SETDEFAULT

This message sets the default state of a button.

**Parameter 1:**
USHORT — Default state? TRUE:FALSE
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
BOOL — Successful? TRUE:FALSE

### BM_SETHILITE

This message either highlights or unhighlights a button.

**Parameter 1:**
USHORT — Highlighted? TRUE: FALSE
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
BOOL — Previously highlighted? TRUE:FALSE

### BM_QUERYCONVERTPOS

Sent to determine if it is OK to convert DBCS characters.

**Parameter 1:**
PRECTL — Cursor position
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
USHORT — Conversion code

# List Box Messages

## WM_CONTROL

This message is sent when a list box needs to notify its owner of some significant event.

**Parameter 1:**

| | |
|---|---|
| USHORT | Control window ID |
| USHORT | Can be one of the following: |

| | |
|---|---|
| LN_ENTER | Enter or return key has been pressed |
| LN_KILLFOCUS | List box is losing focus |
| LN_SCROLL | List box is about to scroll horizontally |
| LN_SETFOCUS | List box is gaining focus |
| LN_SELECT | An item is being selected or deselected |

**Parameter 2:**

| | |
|---|---|
| HWND | Window handle of list box |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_MEASUREITEM

This message is sent to the owner of a list box in order to determine the height and width of the specified item.

**Parameter 1:**

| | |
|---|---|
| SHORT | ID of list box window |

**Parameter 2:**

| | |
|---|---|
| SHORT | Index of the list box item to determine size |

**Reply:**

| | |
|---|---|
| SHORT | Height of list box item of interest |

*Note:* This message only needs to be handled when LS_OWNERDRAW or LS_HORZSCROLL styles are specified.

## LM_DELETEALL

This message is sent to delete all items in a list box.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

## LM_DELETEITEM

This message is sent to a list box to delete a specified item.

**Parameter 1:**
SHORT — Index of list box item to delete
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
SHORT — Number of items remaining in the list box

## LM_INSERTITEM

This message is sent to a list box in order to insert a list box item.

**Parameter 1:**
SHORT — Can be one of the following:

| | |
|---|---|
| LIT_END | Add item to end of list box |
| LIT_SORTASCENDING | Insert item sorted in ascending order |
| LIT_DESCENDING | Insert the item sorted in descending order |
| Other | Insert the item at specified index |

**Parameter 2:**
PSZ — Text to enter in list box item
**Reply:**
SHORT — Index of newly inserted item

## LM_INSERTMULTITEMS

This message is used to insert multiple items into a listbox.

**Parameter 1:**
PLBOXINFO — Pointer to listbox info structure

```
typedef struct _LBOXINFO          /* lboxinfo */
   {
      LONG  lItemIndex;             /* Item index */
      ULONG ulItemCount;            /* Item count */
      ULONG reserved;               /* Reserved - must be zero */
      ULONG reserved2;              /* Reserved - must be zero */
   } LBOXINFO;

   typedef LBOXINFO * PLBOXINFO;
```

**Parameter 2:**
PSZ * — Pointer to an array of strings. For ownerdrawn listboxes that do not use text strings, set this parameter to NULL. See the *ulItemCount* entry in Parameter 1 equal to the number of listbox entries.
**Reply:**
LONG — Number of inserted items

## LM_QUERYITEMCOUNT

This message is sent to the list box to determine the number of items in the list box.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Number of items in list box

### LM_QUERYITEMHANDLE

This message is sent to the list box in order to query the 4 bytes that can be associated with a list box item.

**Parameter 1:**
SHORT Index of list box item to query
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Pointer (or any other 4 byte data type ) to application-defined data

### LM_QUERYITEMTEXT

This message is sent to the list box to query the list box item text.

**Parameter 1:**
SHORT Index of list box item
SHORT Length of buffer (including NULL); 0 returns length of text
**Parameter 2:**
PSZ Buffer for list box item text
**Reply:**
SHORT Length of text string, minus the NULL

### LM_QUERYITEMTEXTLENGTH

This message is sent to the list box to determine the length of a specified list box item.

**Parameter 1:**
SHORT Index of list box item
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Text length of specified item

### LM_QUERYSELECTION

This message is sent to the list box to determine which items are selected.

**Parameter 1:**
SHORT Index of item to start search from; LIT_FIRST indicates to start search at first item

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Index of selected Item; LIT_NONE indicates no more selected items

### LM_QUERYTOPINDEX

This message is sent to the list box to determine the index of the item currently visible at the top of the list box.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Index of item currently at top of list box

### LM_SEARCHSTRING

This message is sent to the list box to find the index of the item that matches the search string.

**Parameter 1:**
SHORT Could be one of the following:

LSS_CASESENSITIVE Search is case sensitive
LSS_PREFIX Leading characters of item match the search string
LSS_SUBSTRING Match if item contains search string

SHORT Index of item to start search from
**Parameter 2:**
PSZ String to use as search criteria
**Reply:**
SHORT Index of item that matches string

### LM_SELECTITEM

This message is sent to the list box to select or deselect an item.

**Parameter 1:**
SHORT Index of item to select or deselect; LIT_NONE indicates all items are to be deselected
**Parameter 2:**
SHORT Can be one of the following:(ignored if *sItemIndex* == LIT_NONE)

TRUE Item is selected
FALSE Item is deselected

**Reply:**
BOOL Successful? TRUE: FALSE

### LM_SETITEMHANDLE

This message is sent to the list box to associate 4 bytes of data with a specific list box item.

**Parameter 1:**
SHORT Index of list box item
**Parameter 2:**
ULONG Application-defined data
**Reply:**
BOOL Successful? TRUE: FALSE

### LM_SETITEMHEIGHT

This message is sent to the list box to set the height of the list box items.

**Parameter 1:**
ULONG New height of list box items, in pixels
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### LM_SETITEMTEXT

This message is sent to the list box to set the text of a specified list box item.

**Parameter 1:**
SHORT Index of list box item
**Parameter 2:**
PSTRL Pointer to list box item text
**Reply:**
BOOL Successful? TRUE: FALSE

### LM_SETITEMWIDTH

This message is used to set the width of the items in a list box.

**Parameter 1:**
ULONG Width of list box items.
**Parameter 2:**
ULONG Reserved, 0.
**Reply:**
BOOL Sucessful?TRUE : FALSE

### LM_SETTOPINDEX

This message is sent to the list box to scroll the list box to the specified item.

**Parameter 1:**
SHORT Index of item to put as currently visible top item
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

## Notebook Messages

### WM_CONTROL

This message is sent when a control has a to notify its owner of a significant event.

**Parameter 1:**
USHORT Control window ID
USHORT Can be one of the following:

| | |
|---|---|
| BKN_HELP | The notebook has received a WM_HELP message |
| BKN_NEWPAGESIZE, | The dimensions of the page window have changed |
| BKN_PAGESELECTED | A new page has been selected as the topmost page of the notebook. This message is sent after the page is turned. |
| BKN_PAGEDELETED | A page has been deleted from the notebook. |
| BKN_PAGESELECTEDPENDING | This notification is sent prior to the notebook page actually turning. |

**Parameter 2:**
ULONG Described in the following table:

| Parameter 1 Value | Parameter 2 Value |
|---|---|
| BKN_HELP | ID of selected page; |
| BKN_PAGESELECTED | a pointer to PAGESELECTNOTIFY structure; |
| BKN_PAGEDELETED | a pointer to DELETENOTIFY structure; |
| BKN_NEWPAGESIZE | the notebook window handle |
| BKN_PAGESELECTEDPENDING | a pointer to PAGESELECTNOTIFY structure |

**Reply:**
ULONG Reserved, 0

### BKM_CALCPAGERECT

This message is used to determine the size of the notebook page, or the application page.

**Parameter 1:**
PRECTL Pointer to RECTL structure

| Parameter 2 Value | Parameter 1 Value |
|---|---|
| TRUE | Input will be the coordinates of the notebook window, output will be the size of application page window |
| FALSE | Input will be the coordinates of an application page window, output will be the size of notebook window. |

**Parameter 2:**
BOOL — Can be one of the following:

| | |
|---|---|
| TRUE | Calculate size of application page window |
| FALSE | Calculate size of notebook window |

**Reply:**
BOOL — Successful? TRUE:FALSE

### BKM_DELETEPAGE

This message is used to delete a page form the notebook.

**Parameter 1:**
ULONG — Page identifier
**Parameter 2:**
USHORT — Can be one of the following:

| Value | Description |
|---|---|
| BKA_SINGLE | Delete a single page |
| BKA_TAB | If Parameter 1 is a page that contains a tab, delete all subsequent pages to the next tab |
| BKA_ALL | Delete all pages in the notebook |

**Reply:**
BOOL — Successful? TRUE:FALSE

### BKM_INSERTPAGE

This message is sent to insert a page into the notebook.

**Parameter 1:**
ULONG — Page ID used as a reference point if BKA_FIRST or BKA_LAST is specified for *usPageOrder*
**Parameter 2:**
USHORT — Can be one of the following:

| Value | Description |
|---|---|
| BKA_AUTOPAGESIZE | Notebook will size the placement and position of page window |
| BKA_STATUSTEXTON | Page will contain status text |
| BKA_MAJOR | Page will have major tab |
| BKA_MINOR | Page will have a minor tab |

USHORT — Can be one of the following:

| Value | Description |
|---|---|
| BKA_FIRST | Page is inserted as the first page of the notebook |
| BKA_LAST | Page is inserted as the last page of the notebook |
| BKA_NEXT | Page is inserted behind the page specified in Parameter 1 |
| BKA_PREV | Page is inserted before the page specified in Parameter 1 |

**Reply:**
ULONG Page ID for the page inserted, NULL if unsuccessful

### BKM_INVALIDATETABS

This message is sent to repaint the notebook tabs.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful repaint? TRUE:FALSE

### BKM_QUERYPAGECOUNT

This message is sent to query the number of pages in the notebook.

**Parameter 1:**
ULONG Page identifier to start counting at; 0 indicates to start at first page
**Parameter 2:**
ULONG Can be one of the following:

BKA_MAJOR Returns the number of pages between ulPageID and the next page that contains a major tab
BKA_MINOR Returns the number of pages between ulPageID and the next page containing a minor tab
BKA_END Returns the number of pages between ulPageID and the last page
**Reply:**
USHORT Number of pages

### BKM_QUERYPAGEDATA

This message is sent to query the data associated with the specified page.

**Parameter 1:**
ULONG The specified page to retrieve the data
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG The 4 bytes of data associated with the page

## BKM_QUERYPAGEINFO

This message is sent to query the page information associated with the specified page.

**Parameter 1:**
ULONG — The specified page to retrieve the page information
**Parameter 2:**
PPAGEINFO — Pointer to PAGEINFO structure

```
typedef struct _BOOKPAGEINFO     /* bkpginfo */
   {
      ULONG     cb;                          /* Page flags - BFA_          */
      ULONG     fl;                          /* Page flags - BFA_          */
      BOOL      bLoadDlg;                    /* TRUE: Load dialog now      */
      /* FALSE: Load dialog on turn*/
      ULONG     ulPageData;                  /* data to associate w/page   */
      HWND      hwndPage;                    /* hwnd to associate w/ page  */
      PFN       pfnPageDlgProc;              /* auto load of dialogs for   */
      ULONG     idPageDlg;                   /* the application.           */
      HMODULE   hmodPageDlg;                 /* Resource info used for     */
      PVOID     pPageDlgCreateParams;
      PDLGTEMPLATE pdlgtPage;
      ULONG     cbStatusLine;                /* Page flags - BFA_          */
      PSZ       pszStatusLine;               /* Status line text string    */
      HBITMAP   hbmMajorTab;                 /* Major tab bitmap handle    */
      HBITMAP   hbmMinorTab;                 /* Minor tab bitmap handle    */
      ULONG     cbMajorTab;                  /* Page flags - BFA_          */
      PSZ       pszMajorTab;                 /* Major tab text string      */
      ULONG     cbMinorTab;                  /* Page flags - BFA_          */
      PSZ       pszMinorTab;                 /* Minor tab text string      */
      PVOID     pBidiInfo;                   /* Reserved: Bidirectional    */
      /* language support.         */
   } BOOKPAGEINFO;
   typedef BOOKPAGEINFO *PBOOKPAGEINFO;
```

**Reply:**
ULONG — The 4 bytes of data associated with the page

## BKM_QUERYPAGEID

This message is sent to find the page ID for a page in the notebook.

**Parameter 1:**
ULONG — Page ID of page used as reference point
**Parameter 2:**
USHORT — Can be one of the following:

| | |
|---|---|
| BKA_FIRST | Queries the page id of the first page |
| BKA_LAST | Queries the page id of the last page |
| BKA_NEXT | Queries the page id of the page after the page specified in parameter 1 |
| BKA_PREV | Queries the page before the page specified in parameter 1 |
| BKA_TOP | Queries the page id of the currently visible page |

USHORT — BKA_MAJOR or BKA_MINOR
**Reply:**
ULONG — Page ID of specified page

### BKM_QUERYPAGESTYLE

This message queries the style of a page of interest.

| | |
|---|---|
| **Parameter 1:** | |
| ULONG | Page identifier |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| USHORT | Page style of specified page |

### BKM_QUERYPAGEWINDOWHWND

This message is used to query the page window handle of the page of interest.

| | |
|---|---|
| **Parameter 1:** | |
| ULONG | Page ID |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| HWND | Handle of the page window associated with the page specified in Parameter 1 |

### BKM_QUERYSTATUSLINETEXT

This message is used to query the status text associated with the specified page.

| | |
|---|---|
| **Parameter 1:** | |
| ULONG | Page ID of page of interest |
| **Parameter 2:** | |
| PBOOKTEXT | Pointer to BOOKTEXT structure |
| **Reply:** | |
| USHORT | Length of returned status string |

### BKM_QUERYTABBITMAP

This message is used to query the bitmap of the page of interest.

| | |
|---|---|
| **Parameter 1:** | |
| ULONG | Page ID of page to retrieve bitmap handle |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| HBITMAP | Handle of bit map |

### BKM_QUERYTABTEXT

This message is used to query the text of the tab of the page of interest.

**Parameter 1:**
ULONG Page ID of the page of interest
**Parameter 2:**
PBOOKTEXT Pointer to BOOKTEXT structure for page of interest
**Reply:**
USHORT Length of the tab text string

### BKM_SETDIMENSIONS

This message is used to set the size of the major and minor tabs.

**Parameter 1:**
USHORT Desired width of tab, in pixels
USHORT Desired height of tab, in pixels
**Parameter 2:**
USHORT Can be one of the following:

BKA_MAJOR Set tab size of major tabs
BKA_MINOR Set tab size of minor tabs
BKA_PAGEBUTTON Set size of page buttons
**Reply:**
BOOL Successful? TRUE:FALSE

### BKM_SETNOTEBOOKCOLORS

This message is used to set the colors of the notebook.

**Parameter 1:**
ULONG Color index or RGB value to be used
**Parameter 2:**
USHORT Notebook region to set

BKA_BACKGROUNDPAGECOLOR Page background
BKA_BACKGROUNDPAGECOLORINDEX Page background
BKA_BACKGROUNDMAJORCOLOR Major tab background
BKA_BACKGROUNDMAJORCOLORINDEX Major tab background
BKA_BACKGROUNDMINORCOLOR Minor tab background
BKA_BACKGROUNDMINORCOLORINDEX Minor tab background
BKA_FOREGROUNDMAJORCOLOR Major tab foreground
BKA_FOREGROUNDMAJORCOLORINDEX Major tab foreground
BKA_FOREGROUNDMINORCOLOR Minor tab foreground
BKA_FOREGROUNDMINORCOLORINDEX Minor tab foreground

**Reply:**
BOOL Successful? TRUE:FALSE

### BKM_SETPAGEINFO

This message is used to set the page information of a notebook page.

**Parameter 1:**
ULONG — Page ID of page whose page information is to be set.
**Parameter 2:**
PPAGEINFO — Pointer to PAGEINFO structure
**Reply:**
BOOL — Successful? TRUE: FALSE

### BKM_SETPAGEDATA

This message is used to associate 4 bytes of data with a page.

**Parameter 1:**
ULONG — Page ID of page of interest
**Parameter 2:**
ULONG — Four bytes of data to be associated with specified page
**Reply:**
BOOL — Successful? TRUE:FALSE

### BKM_SETPAGEWINDOWHWND

This message is used to associate a window handle with a page.

**Parameter 1:**
ULONG — Page ID to be associated with handle
**Parameter 2:**
HWND — Window handle to be associated with specified page
**Reply:**
BOOL — Successful? TRUE:FALSE

### BKM_SETSTATUSLINETEXT

This message is used to associate a text string with a page's status line.

**Parameter 1:**
ULONG — Page ID of page to set status line text
**Parameter 2:**
PSZ — Pointer to text string
**Reply:**
BOOL — Successful? TRUE:FALSE

### BKM_SETTABBITMAP

This message is used to set a bitmap with a page.

**Parameter 1:**
ULONG — Page ID to associate with bitmap
**Parameter 2:**
HBITMAP — Bitmap handle to be used

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE:FALSE |

### BKM_SETTABTEXT

This message is used to set the text of a major or minor tab.

**Parameter 1:**

| | |
|---|---|
| ULONG | Page containing tab of interest |

**Parameter 2:**

| | |
|---|---|
| PSZ | Pointer to tab text string |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE:FALSE |

### BKM_TURNTOPAGE

This message is used to set the currently visible notebook page.

**Parameter 1:**

| | |
|---|---|
| ULONG | Page ID to be brought to top |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE:FALSE |

## Value Set Messages

### WM_CONTROL

Sent when a control is to notify its owner of a significant event.

**Parameter 1:**

| | |
|---|---|
| USHORT | Value set control ID |
| USHORT | Can be one of the following: |

| | |
|---|---|
| VN_DRAGLEAVE | Value set receives a DM_DRAGLEAVE message. |
| VN_DRAGOVER | Value set receives a DM_DRAGOVER message. |
| VN_DROP | Value set receives a DM_DROP message. |
| VN_DROPHELP | Value set receives a DM_DROPHELP message. |
| VN_ENTER | Enter key was pressed on an item. |
| VN_HELP | Value set receives a WM_HELP message. |
| VN_INITDRAG | Drag was initiated on the value set. |
| VN_KILLFOCUS | Value set is losing focus. |
| VN_SELECT | A value set item has been selected. |
| VN_SETFOCUS | Value set is receiving focus. |

**Parameter 2:**

ULONG — For VN_DRAGOVER, VN_DRAGLEAVE, VN_DROP, or VN_DROPHELP pointer to VSDRAGINFO structure

```
typedef struct _VSDRAGINFO {
        PDRAGINFO pDragInfo; /* pointer to DRAGINFO structure */
        USHORT usRow;        /* Row */
        USHORT usColumn;     /* Column */
} VSDRAGINFO

typedef struct _DRAGINFO {
        ULONG ulDraginfo;    /* structure size */
        ULONG usDragitem;    /* DRAGITEM structures sizes */
        SHORT usOperation;   /* modified drag operations */
        HWND hwndSource;     /* window handle */
        SHORT xDrop;         /* x-coordinate */
        SHORT yDrop;         /* y-coordinate */
        USHORT cditem;       /* count of DRAGITEM structures */
        USHORT usReserved;   /* reserved */
} DRAGINFO;
```

For VN_INITDRAG, pointer to VSDRAGINIT structure

```
typedef struct _VSDRAGINIT {
        HWND hwndVS;         /* value set window handle */
        LONG x;              /* x-coordinate */
        LONG y;              /* y-coordinate */
        LONG cx;             /* x-offset */
        LONG cy;             /* y-offset */
        USHORT usRow;        /* row */
        USHORT usColumn;     /* column */
} VSDRAGINIT;
```

For VN_ENTER, VN_HELP, or VN_SELECT, contains row and column of selected item; low byte is row, and high byte is column

**Reply:**

ULONG — Reserved, 0

## VM_QUERYITEM

Queries the contents of the specified item.

**Parameter 1:**

USHORT — Row

USHORT — Column

**Parameter 2:**

PVSTEXT — Pointer to VSTEXT structure

```
typedef struct _VSTEXT {
        PSZ pszItemText;     /* pointer */
        USHORT usBufLen;     /* buffer size */
} VSTEXT;
```

**Reply:**

ULONG — Can be one of the following:

| Item Attribute | Information returned |
|---|---|
| VIA_TEXT | USHORT length of text. |
| VIA_BITMAP | HBITMAP handle of bitmap associated with item |
| VIA_ICON | HPOINTER handle of icon associated with item. |
| VIA_RGB | ULONG rgb value of item. |
| VIA_COLORINDEX | ULONG index of color index of item. |

### VM_QUERYITEMATTR

Queries the attribute(s) of the item specified.

**Parameter 1:**

| | |
|---|---|
| USHORT | Row |
| USHORT | Column |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| USHORT | Can be one of the following: |
| | |
| VIA_BITMAP | Value set item is a bitmap. |
| VIA_COLORINDEX | Value set item is a colorindex. |
| VIA_ICON | Value set item is an icon. |
| VIA_RGB | Value set item is an RGB value. |
| VIA_TEXT | Value set item is a text item. |
| VIA_DISABLED | Value set item is disabled. |
| VIA_DRAGGABLE | Value set item is draggable. |
| VIA_DROPONABLE | Value set item is droponable. |
| VIA_OWNERDRAW | Value set item is ownerdrawn. |

### VM_QUERYMETRICS

Queries for the current size of each value set item, or the spacing between them.

**Parameter 1:**

| | |
|---|---|
| USHORT | Control metrics VMA_ITEMSIZE or VMA_ITEMSPACING |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | *usWidth*, in pixels, and *usHeight*, in pixels |

### VM_QUERYSELECTEDITEM

Queries for the currently selected item in the value set.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
USHORT Row
USHORT Column

### VM_SELECTITEM

Selects the specified value, deselecting the previously selected item.

**Parameter 1:**
USHORT Row
USHORT Column
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE:FALSE

### VM_SETITEM

Specifies type of information contained by an item.

**Parameter 1:**
USHORT Row
USHORT Column
**Parameter 2:**
ULONG Item information

| Item attribute | Type of information returned |
|---|---|
| VIA_TEXT | PSZ pszItemTextString |
| VIA_BITMAP | HBITMAP hbmItemBitamp |
| VIA_ICON | HPOINTER hptrItemPointer |
| VIA_RGB | ULONG rbgItem |
| VIA_COLORINDEX | ULONG ulColorIndex |

**Reply:**
BOOL Successful? TRUE:FALSE

### VM_SETITEMATTR

Sets the attribute(s) of the specified item.

**Parameter 1:**
USHORT Row
USHORT Column
**Parameter 2:**
USHORT Item attributes. Can be one of the following:

VIA_BITMAP
VIA_ICON
VIA_TEXT
VIA_RGB
VIA_COLORINDEX
VIA_OWNERDRAW
VIA_DISABLED
VIA_DRAGGABLE
VIA_DROPONABLE

| | |
|---|---|
| USHORT | Set? TRUE:FALSE |
| **Reply:** | |
| BOOL | Successful? TRUE:FALSE |

### VM_SETMETRICS

Sets the size of the value set items, or the spacing between them.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | VMA_ITEMSIZE, or VMA_ITEMSPACING |
| **Parameter 2:** | |
| USHORT | Width of item if VMA_ITEMSIZE, space between horizontal if VMA_ITEMSPACING |
| USHORT | Height of item if VMA_ITEMSIZE, space between vertical if VMA_ITEMSPACING |
| **Reply:** | |
| BOOL | Successful? TRUE:FALSE |

## Slider Messages

### WM_CONTROL

This message is sent to a control's owner, in order to signal a significant event that has occurred to the control.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | Slider control ID |
| USHORT | Notification code; can be any of the following for the slider control: |
| SLN_CHANGE | Slider arm position changed |
| SLN_KILLFOCUS | Slider is losing focus |
| SLN_SETFOCUS | Slider is receiving the focus |
| SLN_SLIDERTRACK | Slider arm is being dragged, but not released |
| **Parameter 2:** | |
| ULONG | When Parameter 1 is SLN_CHANGE, or SLN_SLIDERTRACK, this equals the new arm position, in pixels when Parameter 1 is SLN_SETFOCUS, or SLN_KILLFOCUS, this is the slider window handle |

**Reply:**
ULONG Reserved, 0

### SLM_ADDETENT

This message is sent to add a detent to a slider.

**Parameter 1:**
USHORT Number of pixels detent is positioned from home position
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Detent ID.

### SLM_QUERYDETENDPOS

Queries the slider for the text on a specific tick mark.

**Parameter 1:**
USHORT Tick mark number
USHORT Length of tick text buffer
**Parameter 2:**
PSZ Pointer to tick text buffer
**Reply:**
SHORT Count of bytes returned

### SLM_QUERYSCALETEXT

This message is sent to query the scale text of a particular location on the slider.

**Parameter 1:**
USHORT Tick location to query text.
USHORT Length of input text buffer.
**Parameter 2:**
PSZ Input text buffer.
**Reply:**
SHORT Length of string returned in Parameter 2.

### SLM_QUERYSLIDERINFO

Sends a query to the slider for the position or size of some part of the slider.

**Parameter 1:**
USHORT Can be one of the following:

| | |
|---|---|
| SMA_SHAFTDIMENSIONS | Returns length and breadth of the slider shaft. |
| SMA_SHAFTPOSITION | Returns x, y position of the lower-left corner of the slider shaft. |
| SMA_SLIDERARMDIMENSIONS | Returns the length and bread of the slider arm. |
| SMA_SLIDERARMPOSITION | Returns the position of the slider arm. |

| | |
|---|---|
| USHORT | Can be one of the following: |
| SMA_RANGEVALUE | Reply represents number of pixels between home and current arm position in low byte; high byte is pixel length of slider. |
| SMA_INCREMENTVALUE | Reply represents position as increment value. |
| **Parameter 2:** | |
| ULONG | Reserved, 0. |
| **Reply:** | |
| ULONG | See second USHORT of Parameter 1. |

### SLM_QUERYTICKPOS

This message is used to query the slider for the current position of a specified tick mark.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | Tick mark number |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| USHORT | *x* - coordinate |
| USHORT | *y* - coordinate |

### SLM_QUERYTICKSIZE

This message is used to query the slider for the size of the tick mark.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | Tick mark number |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| USHORT | Tick mark length |

### SLM_REMOVEDETENT

This message is used to remove a detent.

| | |
|---|---|
| **Parameter 1:** | |
| ULONG | Detent ID |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| BOOL | Successful? TRUE:FALSE |

### SLM_SETSCALETEXT

This message is sent to the slider to set the text above a tick mark. Text is centered on the tick mark.

**Parameter 1:**
USHORT Tick mark number
**Parameter 2:**
PSZ Pointer to tick mark text
**Reply:**
BOOL Successful? TRUE:FALSE

### SLM_SETSLIDERINFO

This message is sent to set the current position or dimensions of a specific part of the slider.

**Parameter 1:**
USHORT Information attribute
USHORT Format attribute
**Parameter 2:**
ULONG New value
**Reply:**
BOOL Successful?TRUE:FALSE

### SLM_SETTICKSIZE

This message is used to set the size of a tick mark.

**Parameter 1:**
USHORT Tick mark number
USHORT Tick mark length
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE:FALSE

## Circular Slider Messages

### WM_CONTROL

This message is sent from the circular slider to notify its owner of an important event.

**Parameter 1:**
USHORT Circular slider ID.
USHORT Notification code. Can be one of the following:

| | |
|---|---|
| CSN_SETFOCUS | Indicates the circular slider is gaining or losing focus. |
| CSN_CHANGED | Indicates the circular slider value has been changed. |
| CSN_TRACKING | Indicates the circular slider is being tracked by the mouse. |
| CSN_QUERYBACKGROUNDCOLOR | Gives the application an opportunity to set the background color of the circular slider. |

**Parameter 2:**
ULONG Notification information. Can be one of the following:

| Parameter 1 Notification code | Parameter 2 |
|---|---|
| CSN_SETFOCUS | Contains TRUE if slider is gaining focus |
| CSN_CHANGED | Contains the new value of the slider. |
| CSN_TRACKING | Contains the intermediate value of the slider. |
| CSN_QUERYBACKGROUNDCOLOR | NULL |

**Reply:**
ULONG Reserved, 0.

### CSM_QUERYINCREMENT

This message is sent to query the value and tick mark increments.

**Parameter 1:**
PUSHORT Returns the increment value used in scrolling the slider.
**Parameter 2:**
PUSHORT Returns the increment value used to draw the tick marks.
**Reply:**
ULONG Successful? TRUE: FALSE.

### CSM_QUERYRADIUS

This message is sent to query the radius of the circular slider.

**Parameter 1:**
PUSHORT Returns the slider radius.
**Parameter 2:**
ULONG Reserved, 0.
**Reply:**
ULONG Successful? TRUE: FALSE.

### CSM_QUERYRANGE

This message is sent to query the range of the circular slider.

**Parameter 1:**
PSHORT Returns the low range value.
**Parameter 2:**
PSHORT Returns the high range value.
**Reply:**
ULONG Successful? TRUE: FALSE.

### CSM_QUERYVALUE

This message is sent to query the current value of the circular slider.

**Parameter 1:**
PSHORT Returns the value of the slider.

**Parameter 2:**
ULONG Reserved, 0.
**Reply:**
ULONG Successful? TRUE: FALSE.

### CSM_SETBITMAPDATA

This message is sent to set the bitmap data associated with the circular slider.

**Parameter 1:**
PCSBITMAPDATA Contains a pointer to a CSBITMAPDATA structure.

```
typedef struct _CSBITMAPDATA   /* csbitmap */
   {
      HBITMAP hbmLeftUp;
      HBITMAP hbmLeftDown;
      HBITMAP hbmRightUp;
      HBITMAP hbmRightDown;
   } CSBITMAPDATA;
   typedef CSBITMAPDATA *PCSBITMAPDATA;
```

**Parameter 2:**
ULONG Reserved, 0.
**Reply:**
ULONG Successful? TRUE: FALSE.

### CSM_SETINCREMENT

This message is sent to set the scroll and tick mark increments of the circular slider.

**Parameter 1:**
USHORT Indicate the scroll increment desired.
**Parameter 2:**
USHORT Indicate the tick mark increment desired.
**Reply:**
ULONG Successful? TRUE: FALSE

### CSM_SETRANGE

This message is sent to set the range of values of the circular slider.

**Parameter 1:**
SHORT Indicate the minimum value of the slider.
**Parameter 2:**
SHORT Indicate the maximum value of the slider.
**Reply:**
ULONG Successful? TRUE: FALSE

### CSM_SETVALUE

This message is sent to set the current value of the circular slider.

| | |
|---|---|
| **Parameter 1:** | |
| SHORT | Indicates the new value to which to set the slider. |
| **Parameter 2:** | |
| ULONG | Reserved, 0. |
| **Reply:** | |
| ULONG | Successful? TRUE: FALSE. |

# File Dialog Messages

### FDM_ERROR

This message is sent from the file dialog before the dialog displays an error message.

| | |
|---|---|
| **Parameter 1:** | |
| USHORT | Error message ID |
| **Parameter 2:** | |
| ULONG | Reserved, 0 |
| **Reply:** | |
| USHORT | If 0, bring up error message dialog; else no error message dialog is brought up, and the following values can be returned to the file dialog: MBID_OK, MBID_CANCEL, or MBID_RETRY |

### FDM_FILTER

This message is sent from the file dialog before a file that meets the filtering criteria is added to the list of selectable files.

| | |
|---|---|
| **Parameter 1:** | |
| PSZ | Pointer to the file name |
| **Parameter 2:** | |
| PSZ | Pointer to the .TYPE EA |
| **Reply:** | |
| BOOL | Add file? TRUE: FALSE |

### FDM_VALIDATE

This message is sent from the file dialog when the user has selected a file and presses OK.

| | |
|---|---|
| **Parameter 1:** | |
| PSZ | Name of selected file |
| **Parameter 2:** | |
| USHORT | Either FDS_EFSELECTION for an entry field selection, or FDS_LBSELECTION for a selection from the list box |
| **Reply:** | |
| BOOL | Valid file? TRUE: FALSE |

# Font Dialog Messages

## FNTM_FACENAMECHANGED

This message is sent from the font dialog whenever the font name is changed by the user.

**Parameter 1:**
PSZ — Pointer to selected font name
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
ULONG — Reserved, 0

## FNTM_FILTERLIST

This message is sent from the font dialog whenever a new font name is to be added to the list.

**Parameter 1:**
PSZ — Font family name
**Parameter 2:**
USHORT — Can be one of the following:

| | |
|---|---|
| FNTI_FAMILYNAME | Indicates a font is being added to the family name combo box. |
| FNTI_STYLENAME | Indicates a style is being added to the style combo box. |
| FNTI_POINTSIZE | Indicates a point size is being added to the point size combo box. |

USHORT — Can be one of the following:

| | |
|---|---|
| FNTI_BITMAPFONT | Item being added is bitmap font information. |
| FNTI_VECTORFONT | Item being added is vector font. |
| FNTI_SYNTHESIZED | Item being added is synthesized font. |
| FNTI_FIXEDWIDTHFONT | Item being added is fixed width font. |
| FNTI_PROPORTIONALFONT | Item being added is proportional width font. |
| FNTI_DEFAULTLIST | A point size from the default list is being added. |

**Reply:**
BOOL — Accept new font? TRUE: FALSE

## FNTM_POINTSIZECHANGED

This message is sent from the font dialog when the user has changed the selected point size.

**Parameter 1:**
PSZ — Pointer to point size string
**Parameter 2:**
FIXED — Point size as FIXED data type
**Reply:**
USHORT — Reserved, 0

### FNTM_STYLECHANGED

This message is sent from the font dialog whenever the user changes the font style attributes.

**Parameter 1:**
PSTYLECHANGE Pointer to style change structure

```
typedef struct _STYLECHANGE     /* stylc */
{
   USHORT      usWeight;
   USHORT      usWeightOld;
   USHORT      usWidth;
   USHORT      usWidthOld;
   ULONG       flType;
   ULONG       flTypeOld;
   ULONG       flTypeMask;
   ULONG       flTypeMaskOld;
   ULONG       flStyle;
   ULONG       flStyleOld;
   ULONG       flStyleMask;
   ULONG       flStyleMaskOld;
} STYLECHANGE;
typedef STYLECHANGE *PSTYLECHANGE;
```

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### FNTM_UPDATEPREVIEW

This message is sent from the font dialog whenever the preview window needs to be changed.

**Parameter 1:**
HWND Preview window handle
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

## Menu Messages

### WM_INITMENU

This message is sent when a menu is about to become active.

**Parameter 1:**
USHORT ID of menu that is becoming active
**Parameter 2:**
HWND Menu window handle
**Reply:**
ULONG Reserved, 0

## WM_MENUEND

This message is sent to the menu's owner to indicate the menu is about to end.

**Parameter 1:**
USHORT — Terminating menu ID
**Parameter 2:**
HWND — Menu window handle
**Reply:**
ULONG — Reserved, 0

## WM_MENUSELECT

This message is sent to the owner of the menu when a menu item has been selected.

**Parameter 1:**
USHORT — ID of selected menu item
USHORT — TRUE — indicates a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message will be posted to the owner
FALSE — no message will be posted
**Parameter 2:**
HWND — Menu window handle
**Reply:**
BOOL — TRUE — indicates the COMMAND messages are to be posted, and menu is dismissed, unless MIA_NODISMISS is set
FALSE — indicates no COMMAND messages are to be posted, and the menu is not dismissed

## WM_NEXTMENU

This message is sent to the owner of the menu to indicate that either the beginning or the end of the menu has been reached.

**Parameter 1:**
HWND — Menu window handle
**Parameter 2:**
USHORT — TRUE — at beginning of menu
FALSE — at end of menu
**Reply:**
HWND — New menu window handle

## MM_DELETEITEM

This message is sent to the window in order to delete a menu item.

**Parameter 1:**
USHORT — Menu item ID
USHORT — TRUE — search submenus for item
FALSE — do not search submenus for item

**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
SHORT — Number of menu items remaining

### WM_ENDMENUMODE

This message is sent to the menu to end the menu.

**Parameter 1:**
USHORT — TRUE — Dismiss the submenu
FALSE — do not dismiss the submenu
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
ULONG — Reserved, 0

### WM_INSERTITEM

This message is sent to the menu to insert an item into the menu.

**Parameter 1:**
PMENUITEM — Pointer to menu-item structure

```
typedef struct _MENUITEM
{
   SHORT   iPosition; /* position in menu or sub-menu */
   USHORT  afStyle; /* menu item style */
   USHORT  afAttribute; /* menu item attributes */
   USHORT  id; /* menu item ID */
   HWND    hwndSubMenu; /* handle for sub-menu, if item is a member of sub-menu
*/
   ULONG   hItem; /* menu item handle */
} MENUITEM;
typedef MENUITEM *PMENUITEM;
```

**Parameter 2:**
PSTRL — Menu item text
**Reply:**
SHORT — Index of newly inserted item

### MM_ISITEMVALID

This message is sent to the menu to determine the selectability of a menu item.

**Parameter 1:**
USHORT — Menu item ID
USHORT — TRUE — search submenus for item
FALSE — do not search submenus for item
**Parameter 2:**
ULONG — Reserved, 0

**Reply:**

| | |
|---|---|
| BOOL | TRUE — item is selectable<br>FALSE — item is not selectable |

### MM_ITEMIDFROMPOSITION

This message is sent to the menu to determine the menu item ID from its position in the menu.

**Parameter 1:**

| | |
|---|---|
| SHORT | Zero-based item index |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| SHORT | Menu item ID |

### MM_ITEMPOSITIONFROMID

This message is sent to the menu to determine a menu item's position from its ID.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID |
| USHORT | TRUE — search submenus for item<br>FALSE — do not search submenus for item |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| SHORT | Item index |

### MM_QUERYDEFAULTITEMID

This message is sent to query the default item ID for a conditional cascade menu.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0. |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0. |

**Reply:**

| | |
|---|---|
| ULONG | Menu ID of default menu item. |

### MM_QUERYITEM

This message is sent to query the definition of a menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID |
| USHORT | TRUE — search submenus for item<br>FALSE — do not search submenus for item |

**Parameter 2:**

| | |
|---|---|
| PMENUITEM | Menu item structure; the menu item definition is copied to this structure on a successful return |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### MM_QUERYITEMATTR

This message queries the menu attributes for a menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID |
| USHORT | TRUE — search submenus for item<br>FALSE — do not search submenus for item |

**Parameter 2:**

| | |
|---|---|
| USHORT | Attributes to return |

**Reply:**

| | |
|---|---|
| USHORT | State of selected attributes |

### MM_QUERYITEMCOUNT

This message is sent to determine the number of items in a menu.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| SHORT | Number of menu items |

### MM_QUERYITEMRECT

This message is sent to determine the rectangle (RECTL) coordinates of a menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID |
| USHORT | TRUE — search submenus for item.<br>FALSE — do not search submenus for item |

**Parameter 2:**

| | |
|---|---|
| PRECTL | Rectangle of the menu item |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### MM_QUERYITEMTEXT

This message is sent to the menu to query the menu item text.

**Parameter 1:**
USHORT Menu item ID
SHORT Size of text input buffer
**Parameter 2:**
PSTRL Menu item text buffer
**Reply:**
SHORT Length of text string

### MM_QUERYITEMTEXTLENGTH

This message is sent to the menu to determine the length of the specified menu item text.

**Parameter 1:**
USHORT Menu item ID
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Length of menu item text, including NULL

### MM_QUERYSELITEMID

This message is sent to determine the selected menu item ID.

**Parameter 1:**
USHORT Reserved, 0
USHORT Can be one of the following:

TRUE Search submenus for item
FALSE Do not search submenus for item

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Selected item ID

### MM_REMOVEITEM

This message is sent to the menu to remove a menu item.

**Parameter 1:**
USHORT Menu item ID
USHORT Can be one of the following:

TRUE Search submenus for item
FALSE Do not search submenus for item

**Parameter 2:**
ULONG Reserved, 0

**Reply:**

| | |
|---|---|
| SHORT | Number of remaining items |

### MM_SELECTITEM

This message is sent to the menu to select or deselect a menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID of interest or MIT_NONE — deselects all menu items |

**Parameter 2:**

| | |
|---|---|
| USHORT | Reserved, 0 |
| USHORT | Can be one of the following: |

| | |
|---|---|
| TRUE | Dismiss menu |
| FALSE | Do not dismiss the menu |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### MM_SETDEFAULTITEMID

This message is sent to set the default item in a conditional cascade menu.

**Parameter 1:**

| | |
|---|---|
| ULONG | ID of the new default menu item. |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0. |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE. |

### MM_SETITEM

This message is sent to the menu to update the definition of a menu item.

**Parameter 1:**

| | |
|---|---|
| USHORT | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| PMENUITEM | New menu item structure |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### MM_SETITEMATTR

This message is sent to the menu to set a menu item attribute.

**Parameter 1:**

| | |
|---|---|
| USHORT | Menu item ID |
| USHORT | Can be one of the following: |

TRUE Search submenus for item
FALSE Do not search submenus for item

**Parameter 2:**
USHORT Masks of attributes to set
USHORT Attribute data
**Reply:**
BOOL Successful? TRUE: FALSE

### MM_SETITEMHANDLE

This message is sent to the menu to set an item handle for a menu item. This is used for the menu items that have a style of MIS_BITMAP or MIS_OWNERDRAW.

**Parameter 1:**
USHORT Menu item ID
**Parameter 2:**
ULONG Item handle
**Reply:**
BOOL Successful? TRUE: FALSE

### MM_SETITEMTEXT

This message is sent to the menu to set, or change, the text of a menu item.

**Parameter 1:**
USHORT Menu item ID
**Parameter 2:**
PSTRL Item text to associate with menu item
**Reply:**
BOOL Successful? TRUE: FALSE

### MM_STARTMENUMODE

This message is sent to the menu to begin menu mode.

**Parameter 1:**
USHORT Can be one of the following:

TRUE Show pull-down menu
FALSE Do not show pull-down menu
**Parameter 2:**
USHORT Can be one of the following:

TRUE Resume where menu had left off
FALSE Begin new user interaction
**Reply:**
BOOL Successful? TRUE: FALSE

# Entryfield Messages

## WM_CONTROL

This message is sent when an entryfield needs to notify its owner of some significant event.

**Parameter 1:**

| | |
|---|---|
| USHORT | Control window ID |
| USHORT | Can be one of the following: |

| | |
|---|---|
| EN_CHANGE | Text was changed and the change has been displayed on the screen |
| EN_KILLFOCUS | Entryfield is losing the focus |
| EN_MEMERROR | Entryfield cannot allocate the necessary memory to accomodate the EM_SETTEXTLIMIT message |
| EN_OVERFLOW | An attempt was made to insert more characters than allowed by the current text limit. If TRUE is returned, the operation is retried; otherwise, it is stopped. |
| EN_SCROLL | Entryfield is about to be scrolled. This can happen if the text has been changed, WinScrollWindow was called, the cursor has moved, or the entryfield must scroll to show the current cursor position. |
| EN_SETFOCUS | Entryfield is receiving the focus |

**Parameter 2:**

| | |
|---|---|
| HWND | Window handle of entryfield |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## EM_QUERYCHANGED

This message is sent to determine if the contents have changed since this message was last sent or since the entryfield was created.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Changed? TRUE: FALSE |

## EM_QUERYSEL

This message is sent to determine the current selection, if one exists.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| SHORT | Offset of the first character selected |
| SHORT | Offset of the last character selected |

### EM_SETSEL

This message is sent to set the current selection to the values specified.

**Parameter 1:**

| | |
|---|---|
| USHORT | Offset of the first character to be selected |
| USHORT | Offset of the last character to be selected |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### EM_SETTEXTLIMIT

This message is sent to specify the maximum number of characters that can be contained in the entryfield.

**Parameter 1:**

| | |
|---|---|
| SHORT | New text limit |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

### EM_CUT

This message is sent to copy the selected text from the entryfield onto the clipboard and then to delete the selected text from the entryfield.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | success? TRUE: FALSE |

### EM_COPY

This message is sent to copy the selected text from the entryfield onto the clipboard. Unlike EM_CUT, however, the selected text is not deleted afterward.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| BOOL | success? TRUE: FALSE |

### EM_CLEAR

This message is sent to delete the selected text from the entryfield.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### EM_PASTE

This message is sent to copy the text from the clipboard into the entryfield. If there is selected text, it is replaced with the clipboard text. Otherwise, the text is inserted at the current cursor position.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### EM_QUERYFIRSTCHAR

This message is sent to determine the zero-based offset of the first visible character.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
SHORT Zero-based offset

### EM_SETFIRSTCHAR

This message is sent to specify the first visible character in the entryfield.

**Parameter 1:**
SHORT New zero-based offset
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

### EM_QUERYREADONLY

This message is sent to determine the read-only status of the entryfield.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Read-only? TRUE: FALSE

### EM_SETREADONLY

This message is sent to specify the new read-only state of the entryfield.

**Parameter 1:**
BOOL New read-only state
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Previously read-only? TRUE:FALSE

### EM_SETINSERTMODE

This message is sent to specify the new insert state of the entryfield.

**Parameter 1:**
BOOL TRUE specifies insert mode
FALSE specifies overwrite mode
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Previously insert mode? TRUE:FALSE

## Spin Button Messages

### WM_CONTROL

This message is sent when a control has to notify its owner of a significant event.

**Parameter 1:**
USHORT Spin button ID
USHORT SPBN_UPARROW, SPBN_DOWNARROW, SPBN_SETFOCUS, SPBN_KILLFOCUS, SPBN_ENDSPIN, or SPBN_CHANGE
**Parameter 2:**
HWND For SPBN_UPARROW, SPBN_DOWNARROW, and SPBN_ENDSPIN this is handle of active spin button for SPBN_SETFOCUS this is handle of current active spin button; for SPBN_KILLFOCUS, this is NULLHANDLE
**Reply:**
ULONG Reserved, 0

### SPBM_OVERRIDESETLIMITS

This message is sent to the spin button to change the upper or lower limit.

**Parameter 1:**
LONG — Upper limit
**Parameter 2:**
LONG — Lower limit
**Reply:**
BOOL — Successful? TRUE:FALSE

### SPBM_QUERYLIMITS

This message is sent to the spin button to query the limits of a numeric spin field.

**Parameter 1:**
LONG — Upper limit
**Parameter 2:**
LONG — Lower limit
**Reply:**
BOOL — Successful? TRUE:FALSE

### SPBM_QUERYVALUE

This message is sent to the spin button to query the current value of the spin button.

**Parameter 1:**
PVOID — Storage for returned value
**Parameter 2:**
USHORT — Size of buffer; if 0, *Parameter 1* is assumed to be address of a longvariable

USHORT — Can be one of the following:

| | |
|---|---|
| SPBQ_UPDATEIFVALID | accept contents of spin button field if within limits |
| SPBN_ALWAYSUPDATE | update contents of the field, even if not valid |
| SPBN_DONOTUPDATE | do not change the current value of the spin button field |

**Reply:**
BOOL — Successful? TRUE:FALSE

### SPBM_SETARRAY

This message is sent to the spin button to set or reset a data array.

**Parameter 1:**
PSZ — Pointer to data array
**Parameter 2:**
USHORT — Number of items in array
**Reply:**
BOOL — Successful? TRUE:FALSE

## SPBM_SETCURRENTVALUE

This message is sent to the spin button to set the current value to either a numeric value or to a data array index.

**Parameter 1:**
LONG — Array index or numeric value
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
BOOL — Successful? TRUE:FALSE

## SPBM_SETLIMITS

This message is sent to the spin button to set or reset numeric limits.

**Parameter 1:**
LONG — Upper limit
**Parameter 2:**
LONG — Lower limit
**Reply:**
BOOL — Successful? TRUE:FALSE

## SPBM_SETMASTER

This message is sent to the spin button to set the master of that spin button.

**Parameter 1:**
HWND — Window handle of master
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
BOOL — Successful? TRUE:FALSE

## SPBM_SETTEXTLIMIT

This message is sent to the spin button to set the maximum number of characters allowed.

**Parameter 1:**
USHORT — Number of characters allowed
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
BOOL — Successful? TRUE:FALSE

## SPBM_SPINDOWN

This message is sent to the spin button to spin backward.

**Parameter 1:**
ULONG Number of items to spin
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE:FALSE

### SPBM_SPINUP

This message is sent to the spin button to spin forward.

**Parameter 1:**
ULONG Number of items to spin
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE:FALSE

## Help Manager Messages

### HM_ACTIONBAR_COMMAND

This message is sent to the active window by the help manager when the user selects an action bar help item.

**Parameter 1:**
USHORT ID of selected action bar item
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_CONTROL

This message is sent by the help manager to the application to add a control in the control area of a window.

**Parameter 1:**
USHORT Reserved, 0
USHORT Resource ID of selected control
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_CREATE_HELP_TABLE

This message is sent to the help manager to create a new help table.

**Parameter 1:**
PHELPTABLE Pointer to HELPTABLE structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_DISMISS_WINDOW

This message is sent to the help manager to remove the active help window.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_DISPLAY_HELP

This message is sent to the help manager to display a specified help window.

**Parameter 1:**
USHORT ID of help panel to display if Parameter 2 == HM_RESOURCEID or a PSTRL pointer to the name of help panel to display if Parameter 2 == HM_PANELNAME
**Parameter 2:**
USHORT HM_RESOURCEID or HM_PANELNAME
**Reply:**
ULONG Successful? 0 : error code

### HM_ERROR

This message is sent by the help manager when an error has occurred.

**Parameter 1:**
ULONG Error code

| Error code | Cause |
|---|---|
| HMERR_LOAD_DLL | Resource DLL could not be loaded |
| HMERR_NO_FRAME_WND_IN_CHAIN | A frame window in the window chain could not be found to set the associated help instance |
| HMERR_INVALID_ASSOC_APP_WND | The window hwnd specified in *WinAssociateHelpInstance* is not a valid window handle |
| HMERR_INVALID_ASSOC_HELP_INST | The help instance handle specified in *WinAssociateHelpInstance* is not a valid help instance |

| Error code | Cause |
|---|---|
| HMERR_INVALID_DESTROY_HELP_INST | The help instance window to destroy is not a help instance class object |
| HMERR_NO_HELP_INST_IN_CHAIN | The parent of the application window specified does not have a help manager instance |
| HMERR_INVALID_HELP_INSTANCE_HDL | The handle of the help manager instance does not belong to the help instance class |
| HMERR_INVALID_QUERY_APP_WND | The window handle specified in *WinQueryHelpInstance* is not a valid window handle |
| HMERR_HELP_INST_CALLED_INVALID | The help instance handle does not belong to the help instance class |
| HMERR_HELPTABLE_UNDEFINE | There is no help table provided for context-sensitive help |
| HMERR_HELP_INSTANCE_UNDEFINE | The help instance handle is invalid |
| HMERR_HELPITEM_NOT_FOUND | The specified help item ID was not found in the help table |
| HMERR_INVALID_HELPSUBITEM_SIZE | The help subtable has less than two items |
| HMERR_INDEX_NOT_FOUND | The index is not in the help library file |
| HMERR_CONTENT_NOT_FOUND | The libary file does not have any content |
| HMERR_OPEN_LIB_FILE | The help library file cannot be opened |
| HMERR_READ_LIB_FILE | The help library file cannot be read |
| HMERR_CLOSE_LIB_FILE | The help library file cannot be closed |
| HMERR_INVALID_LIB_FILE | The help library file is improper |
| HMERR_NO_MEMORY | The help manager is unable to allocate the necessary memory |
| HMERR_ALLOCATE_SEGMENT | The help manager is unable to allocate the necessary memory segments |
| HMERR_FREE_MEMORY | The help manager cannot free the requested memory |
| HMERR_PANEL_NOT_FOUND | The help manager is unable to find the specified help panel |
| HMERR_DATABASE_NOT_OPEN | Unable to read the unopened database |

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_EXT_HELP

This message is sent to the help manager to display the extended help panel.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0

**Reply:**
ULONG Successful? 0 : error code

### HM_EXT_HELP_UNDEFINED

This message is sent from the help manager to indicate that the extended help has not been defined.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_GENERAL_HELP

This message is sent to the help manager to display the general help window.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_GENERAL_HELP_UNDEFINED

This message is sent from the help manager to indicate that the general help has not been defined.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_HELP_CONTENTS

This message is sent to the help manager to display the help contents.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_HELP_INDEX

This message is sent to the help manager to display the help index.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_HELPSUBITEM_NOT_FOUND

This message is sent from the help manager when the user requested help on a field that has no related entry in the help subtable.

**Parameter 1:**
USHORT Can be one of the following:

HLPM_WINDOW Help was requested on an application window
HLPM_FRAME Help was requested on a frame window
HLPM_MENU Help was requested on a menu window
**Parameter 2:**
SHORT Window or menu ID requesting help
SHORT Control or menuitem ID
**Reply:**
BOOL Can be one of the following:

FALSE Display extended help
TRUE Do nothing

### HM_INFORM

This message is sent by the help manager that the user selected a help field with the reftype = inform.

**Parameter 1:**
USHORT Resource ID of the help field
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_INVALIDATE_DDF_DATA

This message is sent to the help manager to indicate that the previous dynamic data formatting data is no longer valid.

**Parameter 1:**
ULONG Count of DDFs to be invalidated

**Parameter 2:**
PUSHORT Array of USHORTs that indicate the DDFs to be invalidated
**Reply:**
ULONG Successful? 0 : error code

### HM_KEYS_HELP

This message is sent to the help manager to display the keys help.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Successful? 0 : error code

### HM_LOAD_HELP_TABLE

This message is sent to the help manager to load a new help table.

**Parameter 1:**
USHORT ID of the help table
USHORT Reserved, 0
**Parameter 2:**
HMODULE Handle of DLL that contains help table, NULL for .EXE
**Reply:**
ULONG Successful? 0 : error code

### HM_NOTIFY

This message is sent from the help manager to notify the application of events it may which to subclass.

**Parameter 1:**
USHORT Selected control ID, if *event* is CONTROL_SELECTED or HELP_REQUESTED

USHORT Event

| | |
|---|---|
| CONTROL_SELECTED | A control was selected |
| HELP_REQUESTED | The user requested help |
| OPEN_COVERPAGE | The coverpage was displayed |
| OPEN_PAGE | The coverpage child was displayed |
| SWAP_PAGE | The coverpage child was swapped |
| OPEN_INDEX | The index was opened |
| OPEN_TOC | The table of contents was opened |
| OPEN_HISTORY | The history window was opened |
| OPEN_LIBRARY | The library was opened |
| OPEN_SEARCH_HIT_LIST | The search list was displayed |

**Parameter 2:**

| | |
|---|---|
| HWND | Window handle |

**Reply:**

| | |
|---|---|
| BOOL | Can be one of the following: |

| | |
|---|---|
| TRUE | Help manager will not format the controls |
| FALSE | Help manager will process as normal |

## HM_QUERY

This message to sent to the help manager to query help manager-specific information.

**Parameter 1:**

| | |
|---|---|
| USHORT | message ID |

| | |
|---|---|
| HMQW_INDEX | Query the index window handle |
| HMQW_TOC | Query the table of contents window handle |
| HMQW_SEARCH | Query the search hitlist window handle |
| HMQW_VIEWEDPAGES | Query the viewed pages window handle |
| HMQW_LIBRARY | Query the library list window handle |
| HMQW_OBJCOM_WINDOW | Query the handle of the active communication window |
| HMQW_INSTANCE | Query the handle of the help instance |
| HMQW_COVERPAGE | Query the handle of the coverpage |
| HMQW_VIEWPORT | Query the handle of the viewport window specified in param2 |
| HMQW_GROUP_VIEWPORT | Query the group number of the window specified in param2 |
| HMQW_RES_VIEWPORT | Query the res number of the window specified in param2 |
| HMQW_ACTIVEVIEWPORT | Query the handle of the active window |
| USERDATA | Query the user data |

| | |
|---|---|
| USHORT | HMQVP_NUMBER, HMQVP_NAME, or HMQVP_GROUP |

**Parameter 2:**

| | |
|---|---|
| PVOID | If Parameter 1 is HMQW_VIEWPORT,this is a pointer to either a res number, ID, or group ID<br>If Parameter 1 is HMQW_GROUP_VIEWPORT, this is a pointer to the viewport window of interest<br>If Parameter 1 is HMQW_RES_VIEWPORT, this is a viewport handleof interest |

**Reply:**

| | |
|---|---|
| ULONG | 0 if error, else a window handle, group ID, res number, group number, or user data |

## HM_QUERY_DDF_DATA

This message is sent from the help manager when it sees the :ddf. tag in the help file.

**Parameter 1:**

| | |
|---|---|
| HWND | Client window handle |

**Parameter 2:**
ULONG Resource ID
**Reply:**
HDDF DDF handle to be displayed, or 0 if error

### HM_QUERY_KEYS_HELP

This message is sent from the help manager when the user selects keys help.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
USHORT ID of keys help panel to display

### HM_REPLACE_HELP_FOR_HELP

This message is sent to the help manager to display the application-defined Help for Help instead of the regular Help for Help.

**Parameter 1:**
USHORT ID of application defined Help for Help panel
**Parameter 2:**
ULONG Reserved. 0
**Reply:**
ULONG Reserved, 0

### HM_REPLACE_USING_HELP

This message is sent to the help manager to display the application-defined Using Help instead of the regular Using Help.

**Parameter 1:**
USHORT ID of application defined Using Help panel
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

### HM_SET_ACTIVE_WINDOW

This message is sent to the help manager to indicate the window with which the help manager should communicate.

**Parameter 1:**
HWND The handle of the window to be made active
**Parameter 2:**
HWND The handle of the window to position help window next to

**Reply:**

| | |
|---|---|
| ULONG | Successful? 0 : error code |

### HM_SET_COVERPAGE_SIZE

This message is sent to the help manager to set the coverpage size.

**Parameter 1:**

| | |
|---|---|
| PRECTL | Pointer to RECTL structure containing the coverpage size |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Successful? 0 : error code |

### HM_SET_LIBRARY_NAME

This message is sent to the help manager to define a list of help manager libraries (.HLP files).

**Parameter 1:**

| | |
|---|---|
| PSTRL | List of help manager files, each name separated by a blank |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Successful? 0 : error code |

### HM_SET_HELP_WINDOW_TITLE

This message is sent to the help manager to change the help window title.

**Parameter 1:**

| | |
|---|---|
| PSTRL | New help window title |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Successful? 0 : error code |

### HM_SET_OBJCOM_WINDOW

This message is sent to the help manager to identify the window to which the help manager is to send the HM_INFORM and HM_QUERY_DDF_DATA messages.

**Parameter 1:**

| | |
|---|---|
| HWND | Handle of communication window |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Handle of previous communication window |

## HM_SET_SHOW_PANEL_ID

This message is sent to the help manager to display or not display the panel id of the help manager windows.

**Parameter 1:**

| | |
|---|---|
| USHORT | Can be one of the following: |
| CMIC_HIDE_PANEL_ID | Hides panel ID |
| CMIC_SHOW_PANEL_ID | Shows panel ID |
| CMIC_TOGGLE_PANEL_ID | Changes hide/show state of panel ID |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Successful? 0 : error code |

## HM_SET_USERDATA

This message is sent to the help manager to store data in the help manager user-defined data area.

**Parameter 1:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Parameter 2:**

| | |
|---|---|
| PVOID | 4 bytes of user-defined data |

**Reply:**

| | |
|---|---|
| ULONG | Successful? TRUE: FALSE |

## HM_TUTORIAL

This message is sent from the help manager to indicate the user selected tutorial.

**Parameter 1:**

| | |
|---|---|
| PSTRL | Default tutorial name |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## HM_UPDATE_OBJCOM_WINDOW_CHAIN

This message is sent from the help manager to the active communication window when a communication object wants to withdraw from the window chain.

**Parameter 1:**

| | |
|---|---|
| HWND | Handle of withdrawing object |

**Parameter 2:**

| | |
|---|---|
| HWND | Window that contains withdrawing object |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

# Drag and Drop Messages

## DM_DISCARDOBJECT

This message is sent to a source object if DRM_DISCARD rendering is allowed.

**Parameter 1:**
PDRAGINFO Pointer to DRAGINFO structure

```
typedef struct _DRAGINFO {
   ULONG cbDraginfo;
   USHORT cbDragitem;
   USHORT usOperation;
   HWND hwndSource;
   SHORT xDrop;
   SHORT yDrop;
   USHORT cditem;
   USHORT usReserved;
} DRAGINFO, *PDRAGINFO
```

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Can be one of the following:

| | |
|---|---|
| DRR_SOURCE | Source takes action on object |
| DRR_TARGET | Target takes action on object |
| DRR_ABORT | Abort action |

## DM_DRAGERROR

This message is sent when an error occurs during a move or copy file operation.

**Parameter 1:**
USHORT Error code
USHORT Failing function

| | |
|---|---|
| DFF_MOVE | *DosMove* |
| DFF_COPY | *DosCopy* |
| DFF_DELETE | *DosDelete* |

**Parameter 2:**
HSTR File with problem
**Reply:**
ULONG Action

| | |
|---|---|
| DME_IGNORECONTINUE | No retry on this file, continue with the rest |
| DME_IGNOREABORT | No retry on this file, abort operation |
| DME_RETRY | Retry |
| DME_REPLACE | Replace the file at target |
| Other | New file name to try; this is a HSTR variable |

## DM_DRAGFILECOMPLETE

This message is sent when a file operation is complete.

**Parameter 1:**
HSTR File handle
**Parameter 2:**
USHORT Result flags OR'ed together

| | |
|---|---|
| DF_MOVE | Operation was move operation. If not set, operation was copy. |
| DF_SOURCE | Receiving window was source. If not set, receiving window was target. |
| DF_SUCCESSFUL | Operation was successful. If not set, operation was not successful. |

**Reply:**
ULONG Reserved, 0

## DM_DRAGLEAVE

This message is sent when an object that was being dragged over a window's boundaries is now either outside those boundaries, or else the operation was aborted.

**Parameter 1:**
PDRAGINFO Pointer to DRAGINFO structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

## DM_DRAGOVER

This message is sent to determine whether objects can be dropped at current destination.

**Parameter 1:**
PDRAGINFO Pointer to DRAGINFO structure
**Parameter 2:**
SHORT Mouse *x* position in desktop coordinates
SHORT Mouse *y* position in desktop coordinates
**Reply:**
USHORT Drop flag
USHORT Default drop operation

| Drop flag | Meaning |
|---|---|
| DOR_DROP | Drop operation OK. |
| DOR_NODROP | Drop operation not OK at current time. Type of drop operation is OK. |
| DOR_NODROPOP | Drop operation not OK. Type of drop operation is not acceptable. |
| DOR_NEVERDROP | Drop operation not OK, now or ever. |

| Operation | Meaning |
|---|---|
| DO_COPY | Copy operation |
| DO_LINK | Link operation |
| DO_MOVE | Move operation |
| Other | User-defined operation |

### DM_DRAGOVERNOTIFY

This message is sent to source after DM_DRAGOVER message has been sent.

**Parameter 1:**

| | |
|---|---|
| PDRAGINFO | Pointer to DRAGINFO structure |

**Parameter 2:**

| | |
|---|---|
| USHORT | Drop flag |
| USHORT | Default drop operation |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### DM_DROP

This message is sent when an object is being dropped on a target.

**Parameter 1:**

| | |
|---|---|
| PDRAGINFO | Pointer to DRAGINFO structure |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### DM_DROPHELP

This message is sent when a user requests help for the drag operation.

**Parameter 1:**

| | |
|---|---|
| PDRAGINFO | Pointer to DRAGINFO structure |

**Parameter 2:**

| | |
|---|---|
| ULONG | Reserved, 0 |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### DM_EMPHASIZETARGET

This message is sent to the target to inform it to either add or remove emphasis from itself.

**Parameter 1:**

| | |
|---|---|
| SHORT | *x* coordinate of mouse in window coordinates |
| SHORT | *y* coordinate of mouse in window coordinates |

**Parameter 2:**

| | |
|---|---|
| USHORT | TRUE — apply emphasis<br>FALSE — remove emphasis |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## DM_ENDCONVERSATION

This message is sent to the source to indicate that the drop operation is over.

**Parameter 1:**

| | |
|---|---|
| ULONG | Item ID that was dropped |

**Parameter 2:**

| | |
|---|---|
| ULONG | DMFL_TARGETSUCCESSFUL — successful<br>DMFL_TARGETFAIL — failed |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## DM_FILERENDERED

This message is sent when a rendering operation is complete.

**Parameter 1:**

| | |
|---|---|
| PRENDERFILE | Pointer to RENDERFILE structure |

```
typedef struct _RENDERFILE {
   HWND hwndDragFiles;
   HSTR hstrSource;
   HSTR hstrTarget;
   USHORT fMove;
   USHORT usRsvd;
} RENDERFILE, *PRENDERFILE;
```

**Parameter 2:**

| | |
|---|---|
| USHORT | Operation successful? TRUE: FALSE |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## DM_PRINTOBJECT

This message is sent when a object is dropped on the printer object.

**Parameter 1:**

| | |
|---|---|
| PDRAGINFO | pointer to DRAGINFO structure |

```
typedef struct _DRAGINFO {
   ULONG cbDraginfo;
   USHORT cbDragitem;
   USHORT usOperation;
   HWND hwndSource;
   SHORT xDrop;
   SHORT yDrop;
   USHORT cditem;
   USHORT usReserved;
} DRAGINFO, *PDRAGINFO
```

**Parameter 2:**

PPRINTDEST Pointer to PRINTDEST structure

```
typedef struct _PRINTDEST {
   ULONG cb;
   LONG lType;
   PSZ pszToken;
   LONG lCount;
   PDEVOPENDATA pdopData;
   ULONG fl;
   PSZ pszPrinter;
} PRINTDEST, *PPRINTDEST;
```

**Reply:**

ULONG Action flag

| Flag | Meaning |
|---|---|
| DRR_SOURCE | Source takes responsiblity for printing object |
| DRR_TARGET | Target (printer object) takes responsibility for printing object |
| DRR_ABORT | Abort the operation |

## DM_RENDER

This message is used to request that an object render another object.

**Parameter 1:**

PDRAGTRANSFER Pointer to DRAGTRANSFER structure

```
typedef struct _DRAGTRANSFER {
   ULONG cb;
   HWND hwndClient;
   PDRAGITEM pditem;
   HSTR hstrSelectedRMF;
   HSTR hstrRenderToName;
   ULONG ulTargetInfo;
   USHORT usOperation;
   USHORT fsReply;
} DRAGTRANSFER, *PDRAGTRANSFER;
```

**Parameter 2:**

ULONG Reserved, 0

**Reply:**

BOOL Successful? TRUE: FALSE

## DM_RENDERCOMPLETE

This message is sent to the source when a rendering operation is complete.

**Parameter 1:**
PDRAGTRANSFER Pointer to DRAGTRANSFER structure
**Parameter 2:**
USHORT Action flag

| Flag | Meaning |
|---|---|
| DMFL_RENDERFAIL | This source cannot render the operation. |
| DMFL_RENDEROK | The source has completed the rendering successfully. |
| DMFL_RENDEDRRETRY | The source has completed the rendering and will allow the target to retry if the target portion of the operation fails. |

**Reply:**
ULONG Reserved, 0

### DM_RENDERFILE

This message is sent to tell an object to render a file.

**Parameter 1:**
PRENDERFILE Pointer to RENDERFILE structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL TRUE — receiver handles rendering
FALSE — caller handles endering

### DM_RENDERPREPARE

This message is sent to tell the source object to prepare for a rendering on an object.

**Parameter 1:**
PDRAGTRANSFER Pointer to DRAGTRANSFER structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

## Container Messages

### WM_CONTROL/CN_BEGINEDIT

This message is sent when a container is about to be edited.

**Parameter 1:**
USHORT Container ID
USHORT CN_BEGINEDIT

**Parmeter 2:**

PCNREDITDATA Pointer to CNREDITDATA structure

```
typedef struct _CNREDITDATA
{
   ULONG        cb;
   HWND         hwndCnr;
   PRECORDCORE  pRecord;
   PFIELDINFO   pFieldInfo;
   PSZ *ppszText;
   ULONG        cbText;
   ULONG        id;
} CNREDITDATA;
typedef CNREDITDATA *PCNREDITDATA;
```

**Reply:**

ULONG Reserved, 0

## WM_CONTROL/CN_COLLAPSETREE

This message is sent to the container's owner whenever a parent item in the container is collapsed.

**Parameter 1:**

USHORT Container ID

USHORT CN_COLLAPSETREE

**Parameter 2:**

PRECORDCORE pointer to RECORDCOREstructure

```
typedef struct _RECORDCORE
{
   ULONG        cb;
   ULONG        flRecordAttr;
   POINTL       ptlIcon;
   struct _RECORDCORE *preccNextRecord;
   PSZ          pszIcon;
   HPOINTER     hptrIcon;
   HPOINTER     hptrMiniIcon;
   HBITMAP      hbmBitmap;
   HBITMAP      hbmMiniBitmap;
   PTREEITEMDESC pTreeItemDesc;
   PSZ          pszText;
   PSZ          pszName;
   PSZ          pszTree;
} RECORDCORE;
typedef RECORDCORE *PRECORDCORE;
```

**Reply:**

ULONG Reserved, 0

## WM_CONTROL/CN_CONTEXTMENU

This message is sent to the container's owner when a container receives a WM_CONTEXTMENU message.

**Parameter 1:**

UHORT Container ID

USHORT CN_CONTEXTMENU

**Parameter 2:**
PRECORDCORE Pointer to RECORDCORE structure
**Reply:**
ULONG Reserved, 0

### WM_CONTROL/CN_DRAGAFTER

This message is sent to the container's owner whenever the container receives a DM_DRAGOVER message.

**Parameter 1:**
USHORT Container ID
USHORT CN_DRAGAFTER
**Parameter 2:**
PCNRDRAGINFO Pointer to CNRDRAGINFOstructure

```
typedef struct _CNRDRAGINFO     /* cdrginfo */
{
   PDRAGINFO   pDragInfo;
   PRECORDCORE pRecord;
} CNRDRAGINFO;
typedef CNRDRAGINFO *PCNRDRAGINFO;
```

**Reply:**
USHORT drop flag

| Flag | Meaning |
|---|---|
| DOR_DROP | Drop operation OK. |
| DOR_NODROP | Drop operation not OK at current time. Type of drop operation is OK. |
| DOR_NODROPOP | Drop operation not OK. Type of drop operation is not acceptable. |
| DOR_NEVERDROP | Drop operation not OK, now or ever. |

USHORT Default drop operation

| Operation | Meaning |
|---|---|
| DO_COPY | Copy operation |
| DO_LINK | Link operation |
| DO_MOVE | Move operation |
| Other | User-defined operation |

### WM_CONTROL/CN_DRAGLEAVE

This message is sent to the container's owner whenever the container receives a DM_DRAGLEAVE message.

**Parameter 1:**
USHORT Container ID
USHORT CN_DRAGLEAVE
**Parameter 2:**
PCNRDRAGINFO Pointer to CNRDRAGINFO structure
**Reply:**
ULONG Reserved, 0

## WM_CONTROL/CN_DRAGOVER

This message is sent to the container's owner whenever the container receives a DM_DRAGOVER message.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_DRAGOVER |

**Parameter 2:**

| | |
|---|---|
| PCNRDRAGINFO | Pointer to CNRDRAGINFO structure |

**Reply:**

| | |
|---|---|
| USHORT | Drop flag |

| Flag | Meaning |
|---|---|
| DOR_DROP | Drop operation OK. |
| DOR_NODROP | Drop operation not OK at current time. Type of drop operation is OK. |
| DOR_NODROPOP | Drop operation not OK. Type of drop operation is not acceptable. |
| DOR_NEVERDROP | Drop operation not OK, now or ever. |

| | |
|---|---|
| USHORT | default drop operation |

| Operation | Meaning |
|---|---|
| DO_COPY | Copy operation |
| DO_LINK | Link operation |
| DO_MOVE | Move operation |
| Other | User-defined operation |

## WM_CONTROL/CN_DROP

This message is sent to the container's owner whenever the container receives a DM_DROP message.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_DROP |

**Parameter 2:**

| | |
|---|---|
| PCNRDRAGINFO | Pointer to CNRDRAGINFO structure |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_CONTROL/CN_DROPHELP

This message is sent to the container's owner whenever the container receives a DM_DROPHELP message.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_DROPHELP |

**Parameter 2:**

PCNRDRAGINFO Pointer to CNRDRAGINFO structure
**Reply:**
ULONG Reserved, 0

## WM_CONTROL/CN_EMPHASIS

This message is sent to the container's owner whenever a record in the container changes its emphasis attribute.

**Parameter 1:**
USHORT Container ID
USHORT CN_DROPEMPHASIS
**Parameter 2:**
PNOTIFYRECORDEMPHASIS Pointer to NOTIFYRECORDEMPHASIS structure

```
typedef struct _NOTIFYRECORDEMPHASIS
{
   HWND        hwndCnr;
   PRECORDCORE pRecord;
   ULONG       fEmphasisMask;
} NOTIFYRECORDEMPHASIS;
typedef NOTIFYRECORDEMPHASIS *PNOTIFYRECORDEMPHASIS;
```

**Reply:**
ULONG Reserved, 0

## WM_CONTROL/CN_ENDEDIT

This message is sent when a container's direct editing of text has ended.

**Parameter 1:**
USHORT Container ID
USHORT CN_ENDEDIT
**Parameter 2:**
PCNREDITDATA Pointer to CNREDITDATA structure
**Reply:**
ULONG Reserved, 0

## WM_CONTROL/CN_ENTER

This message is sent when the Enter key is pressed while the container window has the focus, or the select button is double-clicked while the pointer is over the container window.

**Parameter 1:**
USHORT Container ID
USHORT CN_ENTER
**Parameter 2:**
PNOTIFYRECORDENTER Pointer to NOTIFYRECORDENTER structure

```
typedef struct _NOTIFYRECORDENTER
{
   HWND        hwndCnr;
   ULONG       fKey;
   PRECORDCORE pRecord;
} NOTIFYRECORDENTER;
typedef NOTIFYRECORDENTER *PNOTIFYRECORDENTER;
```

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_CONTROL/CN_EXPANDTREE

This message is sent when a subtree is expanded in the tree view.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_EXPANDTREE |

**Parameter 2:**

| | |
|---|---|
| PRECORDCORE | Pointer to RECORDCORE structure |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_CONTROL/CN_HELP

This message is sent to the owner of the container whenever the container receives a WM_HELP message.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_HELP |

**Parameter 2:**

| | |
|---|---|
| PRECORDCORE | Pointer to RECORDCORE structure |

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

### WM_CONTROL/CN_INITDRAG

Sent when the system-defined drag button was pressed and the pointer was moved while the pointer was over the container control.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_INITDRAG |

**Parameter 2:**

PCNRDRAGINIT Pointer to CNRDRAGINIT structure

```
typedef struct _CNRDRAGINIT
{
   HWND        hwndCnr;
   PRECORDCORE pRecord;
   LONG        x;
   LONG        y;
   LONG        cx;
   LONG        cy;
} CNRDRAGINIT;
typedef CNRDRAGINIT *PCNRDRAGINIT;
```

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_CONTROL/CN_KILLFOCUS

This message is sent when the container loses the focus.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_KILLFOCUS |
| **Parameter 2:** | |
| HWND | Container window handle |
| **Reply:** | |
| ULONG | Reserved, 0 |

## WM_CONTROL/CN_QUERYDELTA

This message is sent to query for more data when a user scrolls to a preset delta value. This value is set via the CNRINFO structure in the *cDelta* field.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_QUERYDELTA |
| **Parameter 2:** | |
| PNOTIFYDELTA | Pointer to NOTIFYDELTA structure |

```
typedef struct _NOTIFYDELTA
{
   HWND        hwndCnr;
   ULONG       fDelta;
} NOTIFYDELTA;
typedef NOTIFYDELTA *PNOTIFYDELTA;
```

**Reply:**

| | |
|---|---|
| ULONG | Reserved, 0 |

## WM_CONTROL/CN_REALLOCPSZ

This message is sent after container text is edited. In order for the changed text to be saved, this message must be processed and TRUE returned. The container then copies the changed text into the new memory area before destroying the MLE.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_REALLOCPSZ |
| **Parameter 2:** | |
| PCNREDITDATA | Pointer to CNREDITDATA structure |
| **Reply:** | |
| BOOL | TRUE — memory is sufficient for copy; go ahead and do it<br>FALSE — memory is insufficient for copy; don't copy string |

## WM_CONTROL/CN_SCROLL

This message is sent when the container window scrolls.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_SCROLL |

**Parameter 2:**

PNOTIFYSCROLL Pointer toNOTIFYSCROLL structure

```
typedef struct _NOTIFYSCROLL
{
   HWND        hwndCnr;
   LONG        lScrollInc;
   ULONG       fScroll;
} NOTIFYSCROLL;
typedef NOTIFYSCROLL *PNOTIFYSCROLL;
```

**Reply:**

ULONG Reserved, 0

## WM_CONTROL/CN_SETFOCUS

This message is sent when the container receives the focus.

**Parameter 1:**

| | |
|---|---|
| USHORT | Container ID |
| USHORT | CN_SETFOCUS |

**Parameter 2:**

HWND Container window handle

**Reply:**

ULONG Reserved, 0

## CM_ALLOCDETAILFIELDINFO

This message is sent to the container to allocate memory for the FIELDINFO structures for its details view.

**Parameter 1:**

USHORT Number of FIELDINFO structures to allocate

**Parameter 2:**

ULONG Reserved, 0

**Reply:**

PFIELDINFO Pointer to FIELDINFO structures

```
typedef struct _FIELDINFO
{
   ULONG       cb;
   ULONG       flData;
   ULONG       flTitle;
   PVOID       pTitleData;
   ULONG       offStruct;
   PVOID       pUserData;
   struct _FIELDINFO *pNextFieldInfo;
   ULONG       cxWidth;
} FIELDINFO;
typedef FIELDINFO *PFIELDINFO;
```

## CM_ALLOCRECORD

This message is sent to allocate memory for the RECORDCORE or MINIRECORDCORE structures.

**Parameter 1:**
ULONG Number of bytes of data to allocate for application-defined use
**Parameter 2:**
USHORT Number of structures to allocate
**Reply:**
PRECORDCORE Pointer to RECORDCORE or MINIRECORDCORE structures that have been allocated

```
typedef struct _RECORDCORE     /* recc */
{
   ULONG        cb;
   ULONG        flRecordAttr;
   POINTL       ptlIcon;
   struct _RECORDCORE *preccNextRecord;
   PSZ          pszIcon;
   HPOINTER     hptrIcon;
   HPOINTER     hptrMiniIcon;
   HBITMAP      hbmBitmap;
   HBITMAP      hbmMiniBitmap;
   PTREEITEMDESC pTreeItemDesc;
   PSZ          pszText;
   PSZ          pszName;
   PSZ          pszTree;
} RECORDCORE;
typedef RECORDCORE *PRECORDCORE;

typedef struct _MINIRECORDCORE
{
   ULONG        cb;
   ULONG        flRecordAttr;     /
   POINTL       ptlIcon;
   struct _MINIRECORDCORE *preccNextRecord;
   PSZ          pszIcon;
   HPOINTER     hptrIcon;
} MINIRECORDCORE;
typedef MINIRECORDCORE *PMINIRECORDCORE;
```

*Note:* If CCS_MINIRECORD is specified as the container style, the return is a PMINIRECORDCORE. If it is not specified, the return is a PRECORDCORE.

## CM_ARRANGE

This message is sent to the container to have it arrange the records in the icon view.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
ULONG Reserved, 0

## CM_CLOSEEDIT

This message is sent to close the MLE window that is used to edit the container text.

**Parameter 1:**
ULONG Reserved, 0

**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful edit? TRUE:FALSE

### CM_COLLAPSETREE

This message is sent to the container to tell it to collapse a specified item in the tree view.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure that is to be collapsed
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful collapse? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

### CM_ERASERECORD

This message is sent to erase a record from the current view.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure to erase
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful erase? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE. Erasing a record does not delete the record, or free the RECORDCORE structure; instead the record is only visually erased.

### CM_EXPANDTREE

This message is sent to the container to expand a parent item in the tree view.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure to expand
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful expansion? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

### CM_FILTER

This message is sent to the container items, so the subset becomes viewable.

**Parameter 1:**
PFN Pointer to user-defined filter function

**Parameter 2:**
PVOID — Pointer to application data space
**Reply:**
BOOL — Successful? TRUE: FALSE

*Note:* The filter function should be prototyped in the following manner:

```
BOOL PFN pFilterFunction( PRECORDCORE pRecord, PVOID pExtra );
```

If the style CCS_MINIRECORD is specified, *pRecord* is a PMINIRECORDCORE.

## CM_FREEDETAILFIELDINFO

This message is sent to the container to free the FIELDINFO structures.

**Parameter 1:**
PVOID — Pointer to an array of FIELDINFO structures to be freed
**Parameter 2:**
USHORT — Number of structures to free
**Reply:**
BOOL — Successful? TRUE: FALSE

## CM_FREERECORD

This message is sent to the container to free the memory associated with the RECORDCORE structures.

**Parameter 1:**
PVOID — Pointer to an array of RECORDCORE or MINIRECORDCORE structures
**Parameter 2:**
USHORT — Number of structures to free
**Reply:**
BOOL — Successful? TRUE: FALSE

## CM_HORZSCROLLSPLITWINDOW

This message is sent to the container when the user scrolls a split window in the details view.

**Parameter 1:**
USHORT — CMA_LEFT — the left window is being scrolled
CMA_RIGHT — the right window is being scrolled
**Parameter 2:**
LONG — The number of pixels to scroll the window; this can be a plus or minus value
**Reply:**
BOOL — Successful scroll? TRUE:FALSE

## CM_INSERTDETAILFIELDINFO

This message is sent to the container to insert a FIELDINFO structure.

**Parameter 1:**

PFIELDINFO Pointer to FIELDINFO structures to insert

```
typedef struct _FIELDINFO
{
   ULONG      cb;
   ULONG      flData;
   ULONG      flTitle;
   PVOID      pTitleData;
   ULONG      offStruct;
   PVOID      pUserData;
   ULONG      cxWidth;
} FIELDINFO;
typedef FIELDINFO *PFIELDINFO;
```

**Parameter 2:**

PFIELDINFOINSERT Pointer to FIELDINFOINSERT structure

```
typedef struct _FIELDINFOINSERT
{
   ULONG       cb;
   PFIELDINFO  pFieldInfoOrder;
   ULONG       fInvalidateFieldInfo;
   ULONG       cFieldInfoInsert;
} FIELDINFOINSERT;
typedef FIELDINFOINSERT *PFIELDINFOINSERT;
```

**Reply:**

USHORT Number of FIELDINFO structures in the container

## CM_INSERTRECORD

This message is sent to insert RECORDCORE structures into the container.

**Parameter 1:**

PRECORDCORE Pointer to RECORDCORE structure

**Parameter 2:**

PRECORDINSERT Pointer to RECORDINSERT structure

```
typedef struct _RECORDINSERT
{
   ULONG       cb;
   PRECORDCORE pRecordOrder;
   PRECORDCORE pRecordParent;
   ULONG       fInvalidateRecord;
   ULONG       zOrder;
   ULONG       cRecordsInsert;
} RECORDINSERT;
typedef RECORDINSERT *PRECORDINSERT;
```

**Reply:**

ULONG Number of RECORDCORE structure in the container

*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

## CM_INVALIDATEDETAILFIELDINFO

This message is sent to the container to indicate that not all of the FIELDINFO structures are valid and the details view should be refreshed.

**Parameter 1:**
ULONG Reserved, 0
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful?TRUE: FALSE

## CM_INVALIDATERECORD

This message is sent to the container to indicate that some of the RECORDCORE structures are invalid and should be refreshed.

**Parameter 1:**
PVOID Pointer to an array of pointers to RECORDCORE structures that should be refreshed
**Parameter 2:**
USHORT Number of records to be refreshed; 0 indicates all records are to be refreshed
USHORT Flags used to optimize the refresh

| Flag | Meaning |
|---|---|
| CMA_ERASE | Erase the background when the display is refreshed in the icon view. Default is no erase. |
| CMA_REPOSITION | Used to indicate that the RECORDCORE structures need to be reordered within the container. |
| CMA_NOREPOSITION | Used to indicate that the RECORDCORE structures do not need to be reordered within the container. |
| CMA_TEXTCHANGED | Used to indicate the text within the container records has changed. |

**Reply:**
BOOL Successful? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

## CM_OPENEDIT

This message is sent to open the MLE window to edit the container.

**Parameter 1:**
PCNREDITDATA Pointer to CNREDITDATA structure
**Parameter 2:**
ULONG Reserved, 0
**Reply:**
BOOL Successful? TRUE: FALSE

## CM_PAINTBACKGROUND

This message is sent whenever the container's background is painted.

**Parameter 1:**

POWNERBACKGROUND Pointer to OWNERBACKGROUND structure

```
typedef struct _OWNERBACKGROUND
{
   HWND     hwnd;
   HPS      hps;
   RECTL    rclBackground;
   LONG     idWindow;
} OWNERBACKGROUND;
typedef OWNERBACKGROUND *POWNERBACKGROUND;
```

**Parameter 2:**

ULONG Reserved, 0

**Reply:**

BOOL Background drawn? TRUE: FALSE

## CM_QUERYCNRINFO

This message is sent to the container to query the CNRINFO structure.

**Parameter 1:**

PCNRINFO Pointer to CNRINFO structure

```
typedef struct _CNRINFO      /* ccinfo */
{
   ULONG        cb;
   PVOID        pSortRecord;
   PFIELDINFO   pFieldInfoLast;
   PFIELDINFO   pFieldInfoObject;
   PSZ          pszCnrTitle;
   ULONG        flWindowAttr;
   POINTL       ptlOrigin;
   ULONG        cDelta;
   ULONG        cRecords;
   SIZEL        slBitmapOrIcon;
   SIZEL        slTreeBitmapOrIcon;
   HBITMAP      hbmExpanded;
   HBITMAP      hbmCollapsed;
   HPOINTER     hptrExpanded;    /
   HPOINTER     hptrCollapsed;
   LONG         cyLineSpacing;
   LONG         cxTreeIndent;
   LONG         cxTreeLine;
   ULONG        cFields;
   LONG         xVertSplitbar;
} CNRINFO;
typedef CNRINFO *PCNRINFO;
```

**Parameter 2:**

USHORT Size of CNRINFO structure

**Reply:**

USHORT Number of bytes copied

## CM_QUERYDETAILFIELDINFO

This message is sent to the container to return a FIELDINFO structure.

**Parameter 1:**
PFIELDINFO — Pointer to FIELDINFO structure to use as reference if CMA_NEXT or CMA_PREV is specified
**Parameter 2:**
USHORT — CMA_FIRST, CMA_LAST, CMA_NEXT, or CMA_PREV
**Reply:**
PFIELDINFO — pointer to requested FIELDINFO structures

## CM_QUERYDRAGIMAGE

This message is sent to the container to query the image displayed for a specified record in the container.

**Parameter 1:**
PRECORDCORE — Pointer to RECORDCORE structure to be queried
**Parameter 2:**
ULONG — Reserved, 0
**Reply:**
LHANDLE — Handle of icon or bit map
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

## CM_QUERYRECORD

This message is sent to the container to return a specified RECORDCORE structure.

**Parameter 1:**
PRECORDCORE — Pointer to RECORDCORE structure used as search criteria; ignored if CMA_FIRST or CMA_LAST is specified
**Parameter 2:**
USHORT — Search command

| Command | Meaning |
|---|---|
| CMA_FIRST | Retrieve first record in the container |
| CMA_FIRSTCHILD | Retrieve first child record of record specified in Parameter 1 |
| CMA_LAST | Retrieve last record in the container |
| CMA_LASTCHILD | Retrieve last child record of record specified in Parameter 1 |
| CMA_NEXT | Retrieve next record after record specified in Parameter 1 |
| CMA_PARENT | Retrieve parent of record specified in Parameter 1 |
| CMA_PREV | Retrieve record previous to record specified in Parameter 1 |

USHORT — CMA_ITEMORDER — records evaluated in item order
CMA_ZORDER — records evaluated in Z-order
**Reply:**
PRECORDCORE — Pointer to RECORDCORE structure of interest
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 and Reply are PMINIRECORDCOREs.

## CM_QUERYRECORDEMPHASIS

This message is sent to the container to find the record with a specified emphasis.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure of record to start searching with, or CMA_FIRST to start searching with the first record
**Parameter 2:**
USHORT Emphasis attribute to find: CRA_CURSORED, CRA_INUSE, or CRA_SELECTED
**Reply:**
PRECORDCORE Pointer to record satisfying search
*Note:* If the style CCS_MINIRECORD is specified, Parameter 1 and Reply are PMINIRECORDCOREs.

## CM_QUERYRECORDFROMRECT

This message is sent to the container to find the record that is contained in a specified rectangle.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure of record to start searching with, or CMA_FIRST to start searching with the first record
**Parameter 2:**
QUERYRECFROMRECT Pointer to QUERYRECFROMRECT structure

```
typedef struct _QUERYRECFROMRECT
{
   ULONG   cb;
   RECTL   rect;
   ULONG   fsSearch;
} QUERYRECFROMRECT;
typedef QUERYRECFROMRECT *PQUERYRECFROMRECT;
```

**Reply:**
PRECORDCORE Pointer to record satisfying search
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 and Reply are PMINIRECORDCOREs.

## CM_QUERYRECORDINFO

This message is sent to the container to update the specified records.

**Parameter 1:**
PVOID Pointer to array of RECORDCORE structures that the container will copy information into
**Parameter 2:**
USHORT Number of structures in parameter 1
**Reply:**
BOOL Successful? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a pointer to an array of PMINIRECORDCOREs.

## CM_QUERYRECORDRECT

This message is sent to the container to query the bounding rectangle of the specified record.

**Parameter 1:**
PRECTL Bounding rectangle coordinates
**Parameter 2:**

PQUERYRECORDRECT Pointer to QUERYRECORDRECT structure of record that is queried

```
typedef struct _QUERYRECORDRECT
{
   ULONG        cb;
   PRECORDCORE  pRecord;
   ULONG        fRightSplitWindow;
   ULONG        fsExtent;
} QUERYRECORDRECT;
typedef QUERYRECORDRECT *PQUERYRECORDRECT;
```

**Reply:**

BOOL Successful? TRUE: FALSE

## CM_QUERYVIEWPORTRECT

This message is sent to the container to query the rectangle that contains the container's entire coordinates.

**Parameter 1:**

PRECTL Pointer to RECTL that contains the container's coordinates

**Parameter 2:**

USHORT CMA_WINDOW — return the client window relative to the container window
CMA_WORKSPACE —return the client window relative to the desktop

BOOL TRUE — get size of right split window
FALSE — get size of left split window

**Reply:**

BOOL Successful? TRUE: FALSE

## CM_REMOVEDETAILFIELDINFO

This message is sent to the container to remove one or several FIELDINFO structures.

**Parameter 1:**

PVOID Pointer to array of PFIELDINFO that are to be removed

**Parameter 2:**

USHORT Number of structures in Parameter 1

USHORT CMA_FREE — structures are removed from container, and memory is freed
CMA_INVALIDATE — structures are removed from container and container is invalidated; no memory is freed

**Reply:**

SHORT Number of structures left in container

## CM_REMOVERECORD

This message is sent to the container to remove one or more RECORDCORE structures.

**Parameter 1:**

PVOID Pointer to array of PRECORDCORE that are to be removed

**Parameter 2:**

| | |
|---|---|
| USHORT | Number of structures in Parameter 1 |
| USHORT | CMA_FREE — structures are removed from container, and memory is freed<br>CMA_INVALIDATE — structures are removed from container and container is invalidated; no memory is freed |

**Reply:**

| | |
|---|---|
| SHORT | number of structures left in container |

*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE array.

## CM_SCROLLWINDOW

This message is sent to the container to scroll the container window.

**Parameter 1:**

| | |
|---|---|
| USHORT | Direction to scroll<br>CMA_VERTICAL — vertical scroll<br>CMA_HORIZONTAL — horizontal scroll |

**Parameter 2:**

| | |
|---|---|
| LONG | Amount to scroll; this can be a plus or minus value |

**Reply:**

| | |
|---|---|
| BOOL | Successful? TRUE: FALSE |

## CM_SEARCHSTRING

This message is sent to the container to find the record that contains a specified string.

**Parameter 1:**

PSEARCHSTRING Pointer to SEARCHSTRING structure

```
typedef struct _SEARCHSTRING
{
   ULONG  cb;
   PSZ    pszSearch;
   ULONG  fsPrefix;
   ULONG  fsCaseSensitive;
   ULONG  usView;
} SEARCHSTRING;
typedef SEARCHSTRING *PSEARCHSTRING;
```

**Parameter 2:**

| | |
|---|---|
| PRECORDCORE | Pointer to RECORDCORE structure to start search with |

**Reply:**

| | |
|---|---|
| PRECORDCORE | Pointer to RECORDCORE that contains the string |

*Note:* If the style CCS_MINIRECORD is specified, parameter 2 and Reply are PMINIRECORDCORE.

## CM_SETCNRINFO

This message is sent to the container to change the data in the container.

**Parameter 1:**

| | |
|---|---|
| PCNRINFO | Pointer to CNRIFO |

**Parameter 2:**
ULONG Flags indicating which data is to be changed

| Flag | Meaning |
|---|---|
| CMA_PSORTRECORD | pointer to sort function |
| CMA_PFIELDINFOLAST | pointer to last column in details view left window |
| CMA_PFIELDINFOOBJECT | pointer to object column in details view |
| CMA_CNRTITLE | pointer to container text title |
| CMA_FLWINDOWATTR | container attributes |
| CMA_PTLORIGIN | position of container window |
| CMA_DELTA | number of records used in delta operations |
| CMA_SLBITMAPORICON | size of bitmap or icon |
| CMA_SLTREEBITMAPORICON | size of expand/collapse bitmap or icon |
| CMA_TREEBITMAP | expand/collapse bitmaps in tree view |
| CMA_TREEICON | expand/collapse icons in tree view |
| CMA_LINESPACING | amount of space between records |
| CMA_CXTREEINDENT | amount of space between levels in tree view |
| CMA_CXTREELINE | width of lines in tree view |
| CMA_XVERTSPLITBAR | initial position of splitbar in con-tainer window in details view |

**Reply:**
BOOL Successful? TRUE: FALSE

## CM_SETRECORDEMPHASIS

This message is sent to change emphasis on the specified container record.

**Parameter 1:**
PRECORDCORE Pointer to RECORDCORE structure to change
**Parameter 2:**
USHORT TRUE — set emphasis attribute ON
FALSE — set emphasis attribute OFF
USHORT CRA_CURSORED, CRA_INUSE, CRA_SELECTED
**Reply:**
BOOL Successful? TRUE: FALSE
*Note:* If the style CCS_MINIRECORD is specified, parameter 1 is a PMINIRECORDCORE.

## CM_SORTRECORD

This message is sent to the container to sort the container records.

**Parameter 1:**
PFN Pointer to sort function
**Parameter 2:**
PVOID Application-defined space
**Reply:**
BOOL Sorted? TRUE: FALSE
*Note:* The sort function should be prototyped in the following manner:

```
BOOL PFN pSortFunction( PRECORDCORE pFirst,
                        PRECORDCORE pSecond,
                        PVOID pExtra );
```

If the style CCS_MINIRECORD is specified, *pRecord* is a PMINIRECORDCORE.

# Appendix B

# References


IBM [March 1991], Operating System/2™ Programming Tools and Information Version 1.3, *Programming Guide*. [91F9259]

IBM [March 1992], OS/2 2.0 Technical Library, *Control Program Programming Reference*. [10G6263]

IBM [March 1992], OS/2 2.0 Technical Library, *Presentation Manager Programming Reference Volume I.* [10G6264]

IBM [March 1992], OS/2 2.0 Technical Library, *Presentation Manager Programming Reference Volume II.* [10G6265]

IBM [March 1992], OS/2 2.0 Technical Library, *Presentation Manager Programming Reference Volume III.* [10G6272]

IBM [March 1992], OS/2 2.0 Technical Library, *Programming Guide Volume I.* [10G6261]

IBM [March 1992], OS/2 2.0 Technical Library, *Programming Guide Volume II.* [10G6494]

IBM [October 1991], Systems Application Architecture Library, *Common User Access Advanced Interface Design Reference*. [SC34-4290]

IBM [September 1991], Systems Application Architecture Library, *Common Programming Interface C Reference — Level 2*. [SC09-1308-02]

IBM [April 1992], C Set/2, *Migration Guide*. [10G4445]

IBM [April 1992], C Set/2, *User's Guide*. [10G4444

IBM [April 1992], Red Book, *OS/2 Version 2.0 Volume 1: Control Program*. [GG24-3730-00]

IBM [April 1992], Red Book, *OS/2 Version 2.0 Volume 4: Application Development*. [GG24-3774-00]

Paul Somerson [June 1988], *PC Magazine DOS Power Tools Techniques, Tricks and Utilities*, Bantam Books, Inc., New York, New York.

H. M Deitel, M.S. Kogan [1992], *The Design of OS/2*, Addison-Wesley Publishing Company, Inc., New York, New York.

Robert Orfali, Dan Harkey [1992], *Client/Server Programming with OS/2 2.0 2nd Ed.*, Van Nostrand Reinhold, New York, New York.

Reich, David, *Designing OS/2 Applications*, John Wiley & Sons, New York, New York.

Real World Programming for OS/2 2.1, Blain, Delimon, and English. Published by SAMS publishing.

Petzold, Charles, *Programming the OS/2 Presentation Manager*, Ziff-Davis Press.

EDM/2. Published by IQPac Inc. Available on the Internet at hobbes.nmsu.edu in the /os2/newsltr directory and on Compuserve in the OS2DF2 forum.

OS/2 Developer. Published by Miller Freeman Inc. Call (800) WANT-OS2 in the United States or (708) 647-5960 elsewhere for subscription information.

# Index

## —C—

## —D—

## —E—



## —F—

## —G—



## —H—

## —I—



## —K—

## —L—



## —M—

## —N—



## —O—



## —P—

## —Q—

## —R—



## —S—

## —T—



## —V—

## —W—

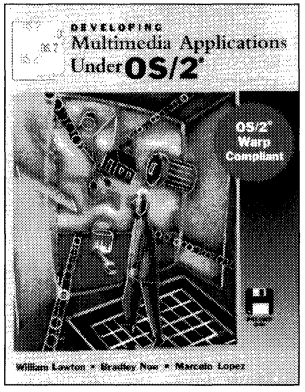
## —X—



## —Z—

# Other OS/2® Books

## FROM JOHN WILEY AND SONS

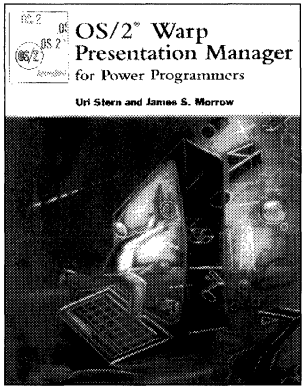



### Developing Multimedia Applications Under OS/2®

WILLIAM LAWTON,
BRADLEY NOE,
MARCELO LOPEZ

This book/disk set provides a comprehensive look at MMPM/2 and how to program powerful multimedia applications. *Developing Multimedia Applications Under OS/2* offers tips and techniques, complete with sample programs for integrating multimedia into your existing applications and developing new ones. The disk includes all the code from the book.

**ISBN# 0471-13168-7**
**$39.95 US/ $55.95 CAN**
**book/disk set**
**810pp. 1995**

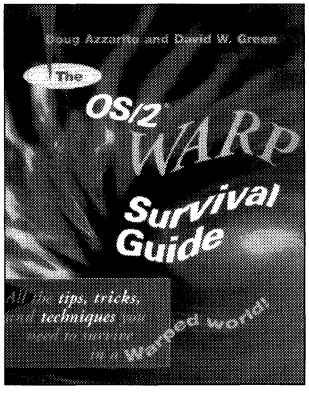



### OS/2® Warp Presentation Manager for Power Programmers

URI STERN AND
JAMES S. MORROW

Written by a pair of IBM insiders from Boca's OS/2 team, this book emphasizes powerful programming methods and stresses structured code, optimal resource management, and numerous other OS/2 performance benefits. You'll gain deep insight into the design and development of the complete Presentation Manager environment, including the new 32 bit Warp Window Manager.

**ISBN# 0471-05839-4**
**$34.95 US/$48.95 CAN**
**480pp. 1995**

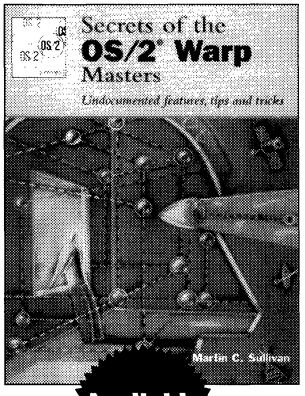



### The OS/2® Warp Survival Guide

DOUG AZZARITO AND
DAVID W. GREEN

This guide shows you how to use all the neat new tools and features IBM has packed into OS/2 Warp. The authors provide you with clear, guidance and a gold mine of tips, tricks, and undocumented information.

**ISBN# 0471-06083-6**
**$29.95 US/$41.95 CAN**
**608pp. 1995**




**Available Oct. '95**

### Secrets of the OS/2® Warp Masters

MARTIN C. SULLIVAN

Written by a member of the OS/2 development team, this book provides an insider's perspective on application development in OS/2 and Presentation Manager. You'll find dozens of OS/2 short cuts, secrets and tips and the disk includes all the code from the book.

**ISBN# 0471-13171-7**
**$39.95 US/$55.95 CAN**
**book/disk set**
**350pp. 1995**

---

WILEY
*Publishers Since 1807*

**Available at Bookstores Everywhere**
For more information **e-mail** - compbks@jwiley.com
or **Visit the Wiley Web site at** - http://www.wiley.com/CompBooks/CompBooks.html

CUSTOMER NOTE: IF THIS BOOK IS ACCOMPANIED BY SOFTWARE, PLEASE READ THE FOLLOWING BEFORE OPENING THE PACKAGE.

This software contains files to help you utilize the models described in the accompanying book. By opening the package, you are agreeing to be bound by the following agreement:



This software product is sold as is without warranty of any kind, either expressed or implied, including but not limited to the implied warranty of merchantability and fitness for a particular purpose. Neither Wiley nor its dealers or distributors assumes any liability of any alleged or actual damages arising from the use or the inability to use this software. (Some states do not allow the exclusion of implied warranties, so this exclusion may not apply to you.)

OS/2 Programming

$44.95 USA
$62.95 CAN

# System development was never easier. Your total hands-on guide to programming OS/2 Warp...

For serious OS/2 developers only! Here is a complete, A-to-Z guide to programming OS/2 Warp. Three leading IBM insiders provide step-by-step guidelines and a gold mine of programming tips and tricks that make developing in Warp's 32-bit OS/2 operating system easier than ever. Using dozens of helpful working example programs in C, the authors explore all the ins and outs of working with the base system, as well as programming using Presentation Manager. And throughout, they've peppered the text with special "gotcha" icons that alert you to common programming mistakes to avoid and tips on how to fix them once they've been made. Some of the crucial topics covered in this hands-on guide include:

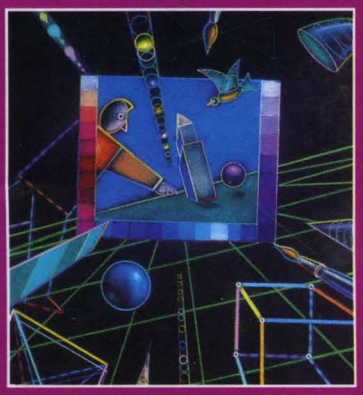
- **GUI design and development**
- **Memory management**
- **File I/O and extended attributes**
- **Multitasking**
- **Interprocess communication**
- **And much more**

**DISK INCLUDES MORE THAN 50 WORKING PROGRAMS**

**KATHLEEN PANOV** is a consultant to IBM specializing in OS/2.

**LARRY SALOMON, Jr.,** is the President of IQ Pac, Inc. An internationally known OS/2 guru, he publishes an extremely popular online OS/2 newsletter.

**ARTHUR PANOV** works for IBM on OS/2.



*Cover Design: Adrienne Weiss*
*Cover Illustration: Andrzej Dudinski*

**John Wiley & Sons, Inc.**
Professional, Reference and Trade Group
605 Third Avenue, New York, N.Y. 10158-0012
New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0-471-08633-9
54495
9 780471 086338