

书 名: Programming Windows 程式开发设计指南

出版日期: 2000/6/2

书 号: 957-8239-73-4

原 作 者: Charles Petzold

译 者: 余孟学

书本主旨:

本书是地球上最有名、最受推崇、最多人使用的程式设计用书。

内容简介:

“到 Petzold 的书中找找”——仍然是解决 Windows 程式开发各种疑难杂症时的灵丹妙药。在第五版的《Windows 程式开发设计指南》中，作者身为倍受敬重的 Windows Pioneer Award（Windows 开路先锋奖）得主，依据最新版本 Windows 作业系统，以可靠的取材资料校定这一本经典之作——再一次深入探索了 Win32 程式设计界面的根本重心。

本书特色:

基本入门: 输出、输入、对话方块

对 Unicode 的介绍

图形处理: 绘图、文字与字体、点阵图形与 metafile

系统核心与印表机

声音与音乐

动态连结程式库

多工与多执行绪

多重文件介面

网际网路与企业内网路程式设计

使用对象:

ALL

{ — ° ç Ô s < œ

ì {2! > h Ú t X+h t ÿìUì Y'qŁ \ ÿÔ;htô! l F'
r Ł,&ì·ÿp" Mäh ru ,ÿì pQ'lp » ÿqÓ{h2%81. X
+

š » öl Ó« .,ì ìÿ&ö÷ì O u÷Ñr:hØÒ[vhßO-.,[r:h{òVÆ
l 2ìP ßO-ÿ.,[!.3+/ ØÒ[v K [v=l <{ {2 2Cÿ{ lÓ!¢ 'ÿÕP
ì k .Àt!! » ÿqÓ{hì FOL @KJAP IB? R>ÿ"\\î À l ‡Ò ° ÖNâ ì
P» WìFOL @KJAPÀ ĩfflÿ_ w íý t »

r. ÿ ÆH {2 š \t r ! t b <š h 1ü+\$! t b š {Òh
ÿwì'i=OL JAP * ì "y Ü! JAP F AA LDL š h Úh4¶N_ ðl h r
ÿe vy Yu .ìÿOMHOANRAN ?OO =F=T v ß[> hHtß ìW Ò ø
HtßW"ª l .y "»+r Wrš " 'æ? ì' æq ? Úæ
l ? ì! ')h š Jh l ') Ps ¥ÿxŁ ðÓÓ b r kktà! e
Æ Úh« JAPìP ~ hÆF AAÿs7 hÆ=LE{2 hØÒ[v % ‡[y4 ì
ÿ¢l rŁ,&ì·ÿMä l » ÿqÓ{! .À

" JAP F=R=ÿN_?t Øð!|{ ØÒ[v l ° tÕPl » !f. »ÿh Óh
ý"t ý !"h'IB? JAPp<Ü¶Ö... š kæP hfÿÚh° »k Ü t h! '
ràkàt! l » P t bĩÿVÆ t b ÿV 'ì JAPÿ Æ9ÀÆØe Ü¶
÷t÷ð ¥ü r ð t b š ltâð Ó° N_ Ót bĩÿVÆ Ók Ü t
h°ýß E÷ðÿht"¥ N_ĩ< h°Ó' ÜĩÿVÆyWð ÀfÿÚ

Ó{ ¾ÿ- wJÓ ßÿ» hß>÷Ñr: O ìÿtPhx 2Cÿ{ J~¾8 xJ
<SD W`uUÿ÷Ñr: ß>t ÜØÒ[vhxJ Ül N_ 'pÿVÆ ĩ·k » ØÒ[v
ÿ t Ü|{h{òVÆhx b2CE~ Ó. Ó. Øh t { °òÿ-

ß> !ìP» ÖNâ Wì JAPhF=R=l #by ÀÚh#s+hĩ· [ÿ',—, h wt°
' ph q hIR? KNII A:hÓhÀð t ý š ðh ÚlpÀh ÚìP Ó«
<{ h4¶'2hÆß^RI » ìP W l k‡ÖtbEpb

. tÓ.pxæ p', P ,#2 6+*II',/°ß¥hk ß> ,ÿ» t Bh ì '
pfp~ ÖJhh ! &ht ! Kÿ"ÿ l p.ÿ',Û Ł,F ÿVÆhÓ f
ÿì ',H s Ch ßßÿ«» ĩìPh! l » hÓÕPe p', f

Nâ{2E÷òì!! [v=LE{2h ĩ· ØFF hì fl. Æ¥ZVÆ ìP ? !
{2 » h Wì IB? JAPI Às757h wf.ß äÆì 4 {2šÓ hì » 2
4!ÿ2CÆßšÓ > e ß>Óòì4 h 4ì y h tÀ hH >h ^C4hß>òì
y h 4ì4 h ŁÓhšÓ ' ìh"9,k

{ 2 Þsk - hß>ì JAP IB? R>lp,ÿøÞ 'p ° - %R h » ÿì - h q
%R - t ß tk ° ÿ f,¥Àà fi ´ ¯Æb § k !ôô h ¯ b
ôÿ, < 4 ¯ h /ß k - l h ô ¾1 o yNâk Õ= ÷ Õ âÿ l¢Uÿ
- h}À, ´æ¯Y.rk H\´Óÿ p °. - Ñ] H' yhn k ¯ k<ÿi
- hæ g k Õ¯Æb ´Ós ô ÿ » ÿh q ÓÿÖt { 2 Þs ° [hL» Þ !
Ö~ÀH Ôÿh w h#°h~Ò #8q l F ï + t !ð t °¢ÿt ìh
· m ‡ ÕÞÿ»

l ¥ÿ; Õhüÿt f¥Zh, k‡ l &ì·ÿp¯h Ý óy, Õ!À! ,À
ÿ ÿÚ, h «l &ì·ÿMäÿøB†Æ L@B , 's<s6 2% , !81 L@B
" > DPPL > > O PDAEPDKIA ?KI NA=@ DPI PE@ DPIH
ü l , " Ø, Ó Þ ßÿ» htÓh·R{ ÿ',h Þ» Wie ',
X. "ªh /ÿ_ À Þÿ

» l q ÓxJìÞïö l ÚhìÞï·E÷ Él » 2 [ÿÿ< , Ö pÿ § ìO
Ø { 2ç - , ö p, "o'+Ø‡q+ ÿ—-p l l pÿ § t¤ 81 · Þ» h w
·ÿ2C¥ï ï‡,ÿ_ h Ýà 6 ól §yh ·ÿ2C +À= t l ¯ l
§?tÞ ! î \ h¥ ÿ,ìðl \ôOÿ t, l Yâ{ÆØ , !4 ôÀ
! ^ !MÖ Ø‡h, æwÆØì l §h Àìÿ .ß Ô/ !% lJ
‡yÿì s Þÿ»

+Ø‡x§j > > O PDAEPDKIA ?KI

ßO - ì » t— ßÿ"\ ¯ ÿ{hÓ q -

ì Útpÿ §h ¯ E §hxJk -

ì ÝxÖ ÿ"ÿh ìÞhtÀe ^

q+ B ÆØh v hÀìÿ .ß Ô/

ÆØÿk pÿ §hqÖ+kÀâ ·a î q+ ÆØ f! ÿ

目 录

第一章 开始.....	1
WINDOWS 环境	1
Windows 简史	2
Windows 方面	3
动态连结	5
WINDOWS 程式设计选项	6
API 和记忆体模式	7
语言选项	8
程式开发环境	9
API 文件	10
编写第一个 WINDOWS 程式	10
文字模式 (Character-Mode) 模型	10
同样效果的 Windows 程式	11
表头档案	12
程式进入点	13
MessageBox 函式	14
编译、连结和执行	15
第二章 UNICODE 简介	17
字元集简史	17
美国标准	18
国际方面	19
扩展 ASCII	20
双位元组字元集	22
Unicode 解决方案	22
宽字元和 C	23
char 资料型态	24
宽字元	24
宽字元程式库函式	25
维护单一原始码	27
宽字元和 WINDOWS	28
Windows 表头档案类型	28
Windows 函式呼叫	30
Windows 的字串函式	31
在 Windows 中使用 printf	31
格式化讯息方块	33
本书与国际化	34
第三章 视窗和讯息	36
自己的视窗	36
总体结构	36
HELLOWIN 程式	38
通盘考量	41
Windows 函式呼叫	41

大写字母识别字.....	42
新的资料型态.....	43
代号简介.....	44
匈牙利表示法.....	44
注册视窗类别.....	46
建立视窗.....	51
显示视窗.....	53
讯息回圈.....	54
视窗讯息处理程式.....	55
处理讯息.....	56
播放音效档案.....	57
WM_PAINT 讯息.....	57
WM_DESTROY 讯息.....	59
WINDOWS 程式设计的难点	60
别呼叫我，我会呼叫您.....	60
伫列化讯息与非伫列化讯息.....	61
行动迅速.....	63
第四章 输出文字.....	64
绘制和更新	64
WM_PAINT 讯息.....	65
有效矩形和无效矩形.....	66
GDI 简介	66
装置内容.....	67
取得装置内容代号：方法一.....	67
绘图资讯结构.....	68
取得装置内容代号：方法二.....	71
TextOut：细节.....	71
系统字体.....	73
字元大小.....	74
文字大小：细节.....	75
格式化文字.....	76
综合使用.....	77
SYSMETSI.C 视窗讯息处理程式.....	84
空间不够.....	85
显示区域的大小.....	86
卷动列	87
卷动列的范围和位置.....	88
卷动列讯息.....	90
在 SYSMETSI 中加入卷动功能.....	92
绘图程式的组织.....	96
建立更好的滚动	97
卷动列资讯函式.....	97
卷动范围.....	99
新 SYSMETSI.....	100
不用滑鼠怎么办.....	106

第五章 图形基础	107
GDI 的结构	107
GDI 原理	107
GDI 函式呼叫	109
GDI 基本图形	110
其他部分	110
装置内容	111
取得装置内容代号	111
取得装置内容资讯	113
DEVCAPS1 程式	114
装置的大小	117
关于色彩	123
装置内容属性	124
保存装置内容	126
画点和线	127
设定图素	127
直线	128
边界框函式	133
贝塞尔曲线	139
使用现有画笔 (Stock Pens)	144
画笔的建立、选择和删除	145
填入空隙	148
绘图方式	148
绘制填入区域	150
Polygon 函式和多边形填入方式	151
用画刷填入内部	155
GDI 映射方式	157
装置座标和逻辑座标	159
装置座标系	159
视埠和视窗	160
处理 MM_TEXT	162
「度量」映射方式	164
「自行决定」的映射方式	168
MM_ISOTROPIC 映射方式	168
MM_ANISOTROPIC: 根据需要放缩图像	171
WHATSIZ 程式	172
矩形、区域和剪裁	176
矩形函式	176
随机矩形	177
建立和绘制剪裁区域	181
矩形与区域的剪裁	183
CLOVER 程式	184
第六章 键盘	189
键盘基础	189
忽略键盘	189

谁获得了焦点.....	190
伫列和同步.....	191
按键和字元.....	191
按键讯息.....	192
系统按键与非系统按键.....	192
虚拟键码.....	193
IParam 资讯.....	197
重复计数.....	197
OEM 扫描码.....	197
扩充键旗标.....	198
内容代码.....	198
键的先前状态.....	198
转换状态.....	198
位移状态.....	198
使用按键讯息.....	199
为 SYSMETS 加上键盘处理功能.....	200
字元讯息.....	208
四类字元讯息.....	208
讯息顺序.....	209
处理控制字元.....	211
死字元讯息.....	211
键盘讯息和字元集.....	212
KEYVIEW1 程式.....	212
外语键盘问题.....	218
字元集和字体.....	220
Unicode 怎么样?.....	231
TrueType 和大字体.....	232
插入符号 (不是游标).....	239
插入符号函式.....	239
TYPER 程式.....	240
第三章 滑鼠.....	248
滑鼠基础.....	248
一些简单的定义.....	249
滑鼠(Mouse)的复数.....	250
显示区域滑鼠讯息.....	250
简单的滑鼠处理: 一个例子.....	252
处理 Shift 键.....	256
双击滑鼠按键.....	257
非显示区域滑鼠讯息.....	258
命中测试讯息.....	259
从讯息产生讯息.....	260
程式中的命中测试.....	261
一个假想的例子.....	261
范例程式.....	262
使用键盘模拟滑鼠.....	265

在 CHECKER 中加入键盘介面	266
将子视窗用於命中测试	271
CHECKER 中的子视窗	271
子视窗和键盘	276
拦截滑鼠	282
设计矩形	282
拦截的解决方案	286
BLOKOUT2 程式	287
滑鼠滑轮	290
下面还有	299
第八章 计时器	300
计时器入门	300
系统和计时器	301
计时器讯息不是非同步的	301
计时器的使用：三种方法	302
方法一	302
方法二	306
方法三	309
计时器用於时钟	310
建立数位时钟	310
取得目前时间	315
显示数字和冒号	315
国际化	316
建立类比时钟	317
以计时器进行状态报告	323
第九章 子视窗控制项	327
按钮类别	328
建立子视窗	332
子视窗向父视窗发讯息	334
父视窗向子视窗发送讯息	335
按键	336
核取方块	337
单选按钮	338
分组方块	338
改变按钮文字	338
可见的和启用的按钮	339
按钮和输入焦点	339
控制项与颜色	340
系统颜色	341
按钮颜色	342
WM_CTLCOLORBTN 讯息	343
拥有者绘制按钮	344
静态类别	350
卷动列类别	352
COLORS1 程式	353

自动键盘介面.....	360
视窗子类别化 (Window Subclassing)	360
给背景著色.....	361
给卷动列和静态文字著色.....	362
编辑类别	363
编辑类别样式.....	365
编辑控制项通知.....	366
使用编辑控制项.....	367
发送给编辑控制项的讯息.....	367
清单方块类别	368
清单方块样式.....	369
将字串放入清单方块.....	369
选择和取得项.....	370
接收来自清单方块的讯息.....	372
一个简单的清单方块应用程式.....	372
档案列表.....	376
使用档案属性码.....	377
档案列表的排序.....	377
Windows 的 head 程式.....	378
第十章 功能表及其他资源	384
图示、游标、字串和自订资源	384
将图示添加到程式.....	385
取得图示代号.....	390
在程式中使用图示.....	392
使用自订游标.....	393
字串资源.....	394
自订的资源.....	396
功能表	404
功能表概念.....	404
功能表结构.....	405
定义功能表.....	405
在程式中引用功能表.....	406
功能表和讯息.....	407
范例程式.....	409
功能表设计规范.....	415
较难的一种功能表定义方法.....	415
浮动突现式功能表.....	417
使用系统功能表.....	422
改变功能表.....	425
其他功能表命令.....	425
建立功能表的非正统方法.....	427
键盘加速键	431
为什么要使用加速键.....	431
安排加速键的几条规则.....	432
加速键表.....	433

加速键表的载入.....	433
键盘代码转换.....	433
接收加速键讯息.....	434
功能表与加速键应用程式 POPPAD.....	435
启用功能表项.....	442
处理功能表项.....	442
第十一章 对话方块	445
模态对话方块	445
建立「About」对话方块.....	446
对话方块及其模板.....	450
对话方块程序.....	453
启动对话方块.....	454
不同的主题.....	455
更复杂的对话方块.....	458
使用对话方块控制项.....	465
「OK」和「Cancel」按钮.....	468
避免使用整体变数.....	469
Tab 停留和分组.....	470
在对话方块上画图.....	472
将其他函式用於对话方块.....	473
定义自己的控制项.....	473
非模态对话方块	480
模态对话方块与非模态对话方块的区别.....	481
新的 COLORS 程式.....	483
HEXCALC: 视窗还是对话方块?	488
通用对话方块	496
增强 POPPAD.....	496
Unicode 档案 I/O.....	518
改变字体.....	519
搜寻与替换.....	520
只呼叫一个函式的 Windows 程式	520
第十二章 剪贴簿	523
剪贴簿的简单使用	523
标准剪贴簿资料格式.....	523
记忆体配置.....	525
将文字传送到剪贴簿.....	527
从剪贴簿上取得文字.....	528
打开和关闭剪贴簿.....	529
剪贴簿和 Unicode.....	529
复杂的剪贴簿用法	535
利用多个资料项目.....	536
延迟提出.....	537
自订资料格式.....	538
实作剪贴簿浏览器	541
剪贴簿浏览器链.....	541

剪贴簿浏览器的函式和讯息.....	541
一个简单的剪贴簿浏览器.....	544
第十三章 使用印表机	548
列印入门	548
列印和背景处理.....	549
印表机装置内容.....	553
修改後的 DEVCAPS 程式	555
PrinterProperties 呼叫.....	566
检查 BitBlt 支援.....	567
最简单的列印程式.....	568
列印图形和文字	569
列印的基本程序.....	573
使用放弃程序来取消列印.....	574
Windows 如何使用 AbortProc.....	575
实作放弃程序.....	576
增加列印对话方块.....	578
为 POPPAD 增加列印功能.....	583
第十四章 点阵图和 BITBLT	590
点阵图入门	590
点阵图的来源.....	591
点阵图尺寸	591
颜色和点阵图.....	592
实际的设备.....	593
GDI 支援的点阵图.....	596
位元块传输	597
简单的 BitBlt.....	597
拉伸点阵图.....	601
StretchBlt 模式	605
位元映射操作.....	605
图案 Blt.....	608
GDI 点阵图物件	610
建立 DDB	610
点阵图位元.....	613
记忆体装置内容.....	614
载入点阵图资源.....	615
单色点阵图格式.....	619
点阵图中的画刷.....	622
绘制点阵图.....	624
阴影点阵图.....	629
在功能表中使用点阵图.....	634
非矩形点阵图图像.....	648
简单的动画.....	653
视窗外的点阵图.....	657
第十五章 与装置无关的点阵图.....	669

DIB 档案格式	669
OS/2 样式的 DIB.....	670
由下而上.....	673
DIB 图素位元.....	673
扩展的 Windows DIB.....	674
真实检查.....	677
DIB 压缩.....	679
颜色遮罩 (COLOR MASKING)	681
第 4 版本的 Header	684
第 5 版的 Header	688
显示 DIB 资讯.....	690
显示和列印	698
了解 DIB.....	698
点对点图素显示.....	701
DIB 的颠倒世界.....	710
循序显示.....	719
缩放到合适尺寸.....	727
色彩转换、调色盘和显示效能.....	738
DIB 和 DDB 的结合.....	739
从 DIB 建立 DDB	739
从 DDB 到 DIB	747
DIB 区块.....	748
DIB 区块的其他区别.....	756
档案映射选项.....	757
总结.....	758
第十六章 调色盘管理器	759
使用调色盘	759
视频硬體.....	759
显示灰阶.....	760
调色盘资讯.....	768
调色盘索引方法.....	769
查询调色盘支援.....	773
系统调色盘.....	774
其他调色盘函式.....	774
位元映射操作问题.....	775
查看系统调色盘.....	776
调色盘动画	786
跳动的球.....	787
一个项目的调色盘动画.....	795
工程應用程式.....	800
调色盘和真实世界图像	805
调色盘和 packed DIB	805
「通用」调色盘.....	816
中间色调色盘.....	823
索引调色盘颜色.....	828

调色盘和点阵图物件.....	834
调色盘和 DIB 区块.....	840
DIB 处理程式库.....	845
DIBSTRUCT 结构.....	847
资讯函式.....	848
读、写图素.....	856
建立和转换.....	860
DIBHELP 表头档案和巨集.....	873
DIBBLE 程式.....	876
档案载入和储存.....	900
显示、卷动和列印.....	900
剪贴簿.....	901
翻转和旋转.....	901
简单调色盘：最佳化调色盘.....	902
均匀分布.....	916
「Popularity」演算法.....	916
「Median Cut」演算法.....	917
转换格式.....	917
第十七章 文字和字体.....	922
简单的文字输出.....	922
文字输出函式.....	922
文字的装置内容属性.....	925
使用备用字体.....	926
字体的背景.....	927
字体型态.....	927
TrueType 字体.....	928
属性或样式.....	929
点值.....	929
间隔和间距.....	929
逻辑英寸问题.....	930
逻辑字体.....	931
逻辑字体的建立和选择.....	931
PICKFONT 程式.....	932
逻辑字体结构.....	950
字体映射演算法.....	954
取得字体资讯.....	955
字元集和 Unicode.....	957
EZFONT 系统.....	958
字体的旋转.....	967
字体列举.....	969
列举函式.....	969
「ChooseFont」对话方块.....	970
段落格式.....	979
简单文字格式.....	980
使用段落.....	981

列印输出预览.....	990
有趣的东西	1002
GDI 绘图路径.....	1003
扩展画笔.....	1004
四个范例程式.....	1008
第十八章 METAFILE	1016
旧的 METAFILE 格式.....	1016
记忆体 metafile 的简单利用.....	1016
将 metafile 储存在磁碟上.....	1020
老式 metafile 与剪贴簿	1021
增强型 METAFILE	1025
基本程序.....	1025
揭开内幕.....	1030
metafile 与 GDI 物件	1037
metafile 和点阵图.....	1042
列举 metafile 内容	1046
嵌入图像.....	1053
增强型 metafile 阅览器和印表机.....	1057
显示精确的 metafile 图像.....	1068
缩放比例和纵横比.....	1078
metafile 中的映射方式.....	1079
映射与显示.....	1082
第十九章 多重文件介面	1087
MDI 概念	1087
MDI 的组成.....	1087
MDI 支援.....	1088
MDI 的范例程式	1090
三个功能表.....	1102
程式初始化.....	1103
建立子视窗.....	1104
关于框架视窗的讯息处理.....	1105
子文件视窗.....	1106
结束处理.....	1108
第二十章 多工和多执行绪	1109
多工的各种模式	1109
DOS 下的多工.....	1109
非优先权式的多工.....	1110
Presentation Manager 和序列化的讯息佇列.....	1111
多执行绪解决方案.....	1112
多执行绪架构.....	1113
执行绪间的「争吵」	1113
Windows 的好处	1114
新改良过的！支援多执行绪了！	1115
WINDOWS 的多执行绪处理	1115

再论随机矩形.....	1116
程式设计竞赛的问题.....	1119
多执行绪解决方案.....	1127
有问题吗?	1136
Sleep 的好处.....	1137
执行绪同步	1137
临界区域.....	1137
事件信号.....	1139
BIGJOB1 程式.....	1140
事件物件.....	1145
执行绪区域储存空间 (TLS)	1149
第二十一章 动态连结程式库.....	1152
动态连结程式库的基本知识	1152
程式库：一词多义.....	1153
一个简单的 DLL.....	1154
程式库入口 / 出口点.....	1157
测试程式.....	1158
在 DLL 中共用记忆体.....	1161
STRPROG 程式.....	1166
在 STRPROG 执行实体之间共用资料.....	1172
各式各样的 DLL 讨论	1173
不用输入引用资讯的动态连结.....	1173
纯资源程式库.....	1174
第二十二章 声音与音乐	1179
WINDOWS 和多媒体	1179
多媒体硬体.....	1179
API 概述	1180
用 TESTMCI 研究 MCI.....	1181
MCITEXT 和 CD 声音	1186
波形声音	1190
声音与波形.....	1190
脉冲编码调制 (Pulse Code Modulation)	1191
取样频率.....	1192
样本大小.....	1192
在软体中产生正弦波.....	1193
数位录音机.....	1204
另一种 MCI 介面	1216
MCI 命令字串的方法	1224
波形声音档案格式.....	1228
叠加合成实验.....	1230
起床号波形声音.....	1238
MIDI 和音乐	1247
使用 MIDI.....	1247
程式更改.....	1249
MIDI 通道.....	1249

MIDI 讯息.....	1251
MIDI 编曲简介.....	1253
通过键盘演奏 MIDI 合成器.....	1259
MIDI 击鼓器.....	1276
多媒体 time 函式	1298
RIFF 档案 I/O	1301
第二十三章 领略 INTERNET	1304
WINDOWS SOCKETS	1304
Sockets 和 TCP/IP	1304
网路时间服务.....	1305
NETTIME 程式	1306
WININET 和 FTP.....	1319
FTP API 概况	1320
更新展示程式.....	1321
第二十四章 附录	1334

第一章 开始

本书介绍了在 Microsoft Windows 98、Microsoft Windows NT 4.0 和 Windows NT 5.0 下程式写作的方法。这些程式用 C 语言编写并使用原始的 Windows Application Programming Interface (API)。如在本章稍後所讨论的,这不是写作 Windows 程式的唯一方法。然而,无论最终您使用什么方式写作程式,了解 Windows API 都是非常重要的。

正如您可能知道的,Windows 98 已成为使用 Intel 32 位元微处理器(例如 486 和 Pentium)的 IBM 相容型个人电脑环境上最新的图形作业系统之代表。Windows NT 是 IBM PC 相容机种以及一些 RISC(精简指令集电脑)工作站上使用的 Windows 工业增强型版本。

使用本书有三个先决条件。首先,您应该从使用者的角度熟悉 Windows 98。不要期望可以在不了解 Windows 使用者介面的情形下开发其应用程式。因此,我建议您在开发程式(或在进行其他工作)时使用执行 Windows 的机器来跑 Windows 应用程式。

第二,您应了解 C 语言。如果要写 Windows 程式,一开始却不想了解 C 语言,那不是一个好主意。我建议您在文字控制台环境中,例如在 Windows 98 MS-DOS 命令提示视窗下提供的环境中学习 C 语言。Windows 程式设计有时包括一些非文字模式程式设计的 C 语言部分;在这些情况下,我将针对这些问题提供讨论。但大多数情况下,您应非常熟悉该语言,特别是 C 语言的结构和指标。了解标准 C 语言执行期程式库的一些相关知识是有帮助的,但不是必要的。

第三,您应该在机器上安装一个适於进行 Windows 程式设计的 32 位元 C 语言编译器和开发环境。在本书中,假定您正在使用 Microsoft Visual C++ 6.0,该套装软体可独立购买,也可作为 Visual Studio 6.0 套装软体的一部分购买。

到此为止,我将不再假设您具有任何图形使用者介面(如 Windows)的程式写作经验。

WINDOWS 环境

Windows 几乎不需要介绍。然而人们很容易忘记 Windows 给办公室和家庭桌上型电脑所带来的重大改变。Windows 在其早期曾经走过一段坎坷的道路,征服桌上型电脑市场的前途一度相当渺茫。

Windows 简史

在 1981 年秋天 IBM PC 推出之後不久, MS-DOS 就已经很明显成为 PC 上的主流作业系统。MS-DOS 代表 Microsoft Disk Operating System (磁碟作业系统)。MS-DOS 是一个小型的作业系统。MS-DOS 提供给用户一种命令列介面, 提供如 DIR 和 TYPE 的命令, 也可以将应用程式载入记忆体执行。对于应用程式写作者, 它提供了一组函式呼叫, 进行档案的输入输出 (I/O)。对于其他的周边处理——尤其是将文字或图形写到显示器上——应用程式可以直接存取 PC 的硬体。

由于记忆体和硬体的限制, 成熟的图形环境缓慢地才到来。当苹果电脑公司不幸的 Lisa 电脑在 1983 年 1 月发表时, 它提供了不同于文字模式环境的另一种选择, 并在 1984 年 1 月成为 Macintosh 上图形环境的一种标准。尽管 Macintosh 的市场占有率在下降, 但是它仍然被认为是衡量所有其他图形环境的标准。包括 Macintosh 和 Windows 的所有图形环境, 其实都要归功于 Xerox Palo Alto Research Center (PARC) 在 70 年代中期所作的开拓性研究工作。

Windows 是由微软在 1983 年 11 月 (在 Lisa 之後, Macintosh 之前) 宣布, 并在两年後 (1985 年 11 月) 发行。在此後的两年中, 紧隨著 Microsoft Windows 早期版本 1.0 之後, 又推出了几种改进版本, 以支援国际商业市场, 并提供新型视讯显示器和印表机的驱动程序。

Windows 版本 2.0 是在 1987 年 11 月正式在市场上推出的。该版本对使用者介面做了一些改进。这些改进中最有效的是使用了可重叠式视窗, 而 Windows 1.0 中使用的是并排式视窗。Windows 2.0 还增强了键盘和滑鼠介面, 特别是加入了功能表和对话方块。

至此, Windows 还只要求 Intel 8086 或者 8088 等级的微处理器, 以「实际模式」执行, 只能存取位址在 1MB 以下的记忆体。Windows/386 (在 Windows 2.0 之後不久发行的) 使用 Intel 386 微处理器的「虚拟 8086」模式, 实现将直接存取硬体的多个 MS-DOS 程式视窗化和多工化。为了统一起见, Windows 版本 2.1 被更名为 Windows/286。

Windows 3.0 是在 1990 年 5 月 22 日发表的。它将 Windows/286 和 Windows/386 结合到同一种产品中。Windows 3.0 有了一个很大的改变, 这就是对 Intel 的 286、386 和 486 微处理器保护模式的支援。这能使 Windows 和 Windows 应用程式能存取高达 16MB 的记忆体。Windows 用于执行程式和维护档案的「外壳」程式得到了全面的改进。Windows 3.0 是第一个在家用和办公室市场上取得立足点的版本。

任何 Windows 的历史介绍都必须包括一些 OS/2 的说明, OS/2 是对 DOS 和 Windows 的另一种选择, 最初是由 Microsoft 和 IBM 合作开发的。OS/2 版本 1.0

(只有文字模式) 在 Intel 286 (或者后来的) 微处理器上运行, 在 1987 年末发布。在 1988 年 10 月的 OS/2 版本 1.1 中出现了管理图形使用者界面的 PM (Presentation Manager)。PM 最初的设计构想是成为 Windows 的一种保护模式版本, 但是图形 API 改变程度太大, 致使软体生产厂商很难提供对这两种平台的支援。

到 1990 年 9 月, IBM 和 Microsoft 之间的冲突达到了高峰, 导致这两个公司最後分道扬镳。IBM 接管了 OS/2, 而 Microsoft 明确表示 Windows 将是他们作业系统策略的中心。虽然 OS/2 仍然拥有一些狂热的崇拜者, 但是它远不及 Windows 这样的普及程度。

Microsoft Windows 版本 3.1 是 1992 年 4 月发布的, 其中包括的几个重要特性是 TrueType 字体技术 (给 Windows 带来可缩放的轮廓字体)、多媒体 (声音和音乐)、物件连结和嵌入 (OLE: Object Linking and Embedding) 和通用对话方块。跟 OS/2 一样, Windows 3.1 只能在保护模式下运作, 并且要求至少配置了 1MB 记忆体的 286 或 386 处理器。

在 1993 年 7 月发表的 Windows NT 是第一个支援 Intel 386、486 和 Pentium 微处理器 32 位元保护模式的 Windows 版本。Windows NT 提供 32 位元平坦定址, 并使用 32 位元的指令集。(本章後面我会谈到一些定址空间的问题)。Windows NT 还可以移植到非 Intel 处理器上, 并在几种使用 RISC 晶片的工作站上执行。

Windows 95 是在 1995 年 8 月发布的。和 Windows NT 一样, Windows 95 也支援 Intel 386 或更高等级处理器的 32 位元保护模式。虽然它缺少 Windows NT 中的某些功能, 诸如高安全性和对 RISC 机器的可携性等, 但是 Windows 95 具有需要较少硬体资源的优点。

Windows 98 在 1998 年 6 月发布, 具有许多加强功能, 包括执行效能的提高、更好的硬体支援以及与网际网路和全球资讯网 (WWW) 更紧密的结合。

Windows 方面

Windows 98 和 Windows NT 都是支援 32 位元优先权式多工 (preemptive multitasking) 及多执行绪的图形作业系统。Windows 拥有图形使用者介面 (GUI), 这种使用者介面也称作「视觉化介面」或「图形视窗环境」。有关 GUI 的概念可追溯至 70 年代中期, 在 Alto 和 Star 等机器上以及 SmallTalk 等环境中由 Xerox PARC 所作的研究工作。该项研究的成果后来被 Apple Computer 和 Microsoft 引入主流并流行起来。虽然有一些争议, 但现在已非常清楚, GUI 是 (Microsoft 的 Charles Simonyi 的说法) 一个在个人电脑工业史上集各方面技术大成於一体的最重要产物。

所有 GUI 都在点矩阵对应的视讯显示器上处理图形。图形提供了使用萤幕的最佳方式、传递资讯的视觉化丰富多彩环境，以及能够 WYSIWYG (what you see is what you get: 所见即所得) 的图形视讯显示和为书面文件准备好格式化文字输出内容。

在早期，视讯显示器仅用於回应使用者通过键盘输入的文字。在图形使用者介面中，视讯显示器自身成为使用者输入的一个来源。视讯显示器以图示和输入设备（例如按钮和卷轴）的形式显示多种图形物件。使用者可以使用键盘（或者更直接地使用滑鼠等指向装置）直接在萤幕上操纵这些物件，拖动图形物件、按下滑鼠按钮以及滚动卷轴。

因此，使用者与程式的交流变得更为亲密。这不再是一种从键盘到程式，再到视讯显示器的单向资讯流动，使用者已经能够与显示器上的物件直接交互作用了。

使用者不再需要花费长时间学习如何使用电脑或掌握新程式了。Windows 让这一切成真，因为所有应用程式都有相同的基本外观和感觉。程式占据一个视窗——萤幕上的一块矩形区域。每个视窗由一个标题列标识。大多数程式功能由程式的功能表开始。用户可使用卷轴观察那些无法在一个萤幕中装下的资讯。某些功能表项目触发对话方块，用户可在其中输入额外的资讯。几乎在每个大的 Windows 程式中都有一个用於开启档案的特殊对话方块。该对话方块在所有这些 Windows 程式中看起来都一样（或接近相同），而且几乎总是从同一功能表选项中启动。

一旦您了解使用一个 Windows 程式的方法，您就非常容易学习其他的 Windows 程式。功能表和对话方块允许用户试验一个新程式并探究它的功能。大多数 Windows 程式同时具有键盘介面和滑鼠介面。虽然 Windows 程式的大多数功能可通过键盘控制，但使用滑鼠要容易得多。

从程式作者的角度看，一致的使用者介面来自於 Windows 建构功能表和对话方块的内置程式。所有功能表都有同样的键盘和滑鼠介面，因为这项工作是由 Windows 处理，而不是由应用程式处理。

为便於多个程式的使用，以及这些程式间资讯的交换，Windows 支援多工。在同一时刻能有多个 Windows 程式显示并运行。每个程式在萤幕上占据一个视窗。用户可在萤幕上移动视窗，改变它们的大小，在不同程式间切换，并从一个程式向另一个程式传送资料。因为这些视窗看起来有些像桌面上的纸（当然，这是电脑还未占据办公桌之前的年代），Windows 有时被称作：一个显示多个程式的「具象化桌面」。

Windows 的早期版本使用一种「非优先权式 (non-preemptive)」的多工系

统。这意味著 Windows 不使用系统计时器将处理时间分配给系统中运行的多个應用程式，程式必须自愿放弃控制以便其他程式运行。在 Windows NT 和 Windows 98 中，多工是优先权式的，而且程式自身可分割成近乎同时执行的多个执行绪。

作业系统不对记忆体进行管理便无法实现多工。当新程式启动、旧程式终止时，记忆体会出现碎裂空间。系统必须能够将闲置的记忆体空间组织在一起，因此系统必须能够移动记忆体中的程式码和资料块。

即使是在 8088 微处理器上跑的 Windows 1.0 也能进行这类记忆体管理。在实际模式限制下，这种能力被认为是软体工程一个令人惊讶的成就。在 Windows 1.0 中，PC 硬体结构的 640KB 记忆体限制，在不要求任何额外记忆体的情况下被有效地扩展了。但 Microsoft 并未就此停步：Windows 2.0 允许 Windows 應用程式存取延伸记忆体 (EMS)；Windows 3.0 在保护模式下，允许 Windows 應用程式存取高达 16MB 的扩展记忆体。Windows NT 和 Windows 98 通过成熟的 32 位元作业系统及平坦定址空间，摆脱了这些旧的限制。

Windows 上执行的程式可共用在称为「动态连结程式库」的档案中的常式。Windows 包括一个机制，能够在执行时连结使用动态连结程式库中常式的程式。Windows 自身基本上就是一个动态连结程式库的集合。

Windows 是一个图形介面，Windows 程式能够在视讯显示器和印表机上充分利用图形和格式化文字。图形介面不仅在外观上更有吸引力，而且还能够让使用者传递高层次的资讯。

Windows 應用程式不能直接存取萤幕和印表机等图形显示设备硬体。相反，Windows 提供一种图形程式语言（称作图形装置介面，或者 GDI），使显示图形和格式化文字更容易。Windows 虚拟化显示硬体，使为 Windows 编写的程式可使用任何具有 Windows 装置驱动程式的视频卡或印表机，而程式无需确定系统相连的装置类型。

对 Windows 开发者来说，将与装置无关的图形介面输出到 IBM PC 上不是件轻松的事。PC 的设计是基於开放式架构的原则，鼓励第三方硬体制造商为 PC 开发周边设备，而且开发了大量这样的设备。虽然出现了多种标准，PC 上的传统 MS-DOS 程式仍不得不各自支援许多不同的硬体设备。这对 MS-DOS 文字处理软体来说非常普遍，它们连同 1 到 2 张有许多小档案的磁片一同销售，每个档案支援一种特定的印表机。Windows 程式不要求每个應用程式都自行开发这些驱动程式，因为这种支援是 Windows 的一部分。

动态连结

Windows 运作机制的核心是一个称作「动态连结」的概念。Windows 提供了

应用程式丰富的可呼叫函式，大多数用於实作其使用者介面和在视讯显示器上显示文字和图形。这些函式采用动态连结程式库 (Dynamic Linking Library, DLL) 的方式撰写。这些动态连结程式库是些具有.DLL 或者有时是.EXE 副档名的档案，在 Windows 98 中通常位於\WINDOWS\SYSTEM 子目录中，在 Windows NT 中通常位於 \WINNT\SYSTEM 和\WINNT\SYSTEM32 子目录中。

在早期，Windows 的主要部分仅通过三个动态连结程式库实作。这代表了 Windows 的三个主要子系统，它们被称作 Kernel、User 和 GDI。当子系统的数目在 Windows 最近版本中增多时，大多数典型的 Windows 程式产生的函式呼叫仍对应到这三个模组之一。Kernel (日前由 16 位元的 KRNL386.EXE 和 32 位元的 KERNEL32.DLL 实现) 处理所有在传统上由作业系统核心处理的事务——记忆体管理、档案 I/O 和多工管理。User (由 16 位的 USER.EXE 和 32 位的 USER32.DLL 实作) 指使用者介面，实作所有视窗运作机制。GDI (由 16 位的 GDI.EXE 和 32 位的 GDI32.DLL 实作) 是一个图形装置介面，允许程式在萤幕和印表机上显示文字和图形。

Windows 98 支援应用程式可使用的上千种函式呼叫。每个函数都有一个描述名称，例如 CreateWindow。该函数 (如您所猜想的) 为程式建立新视窗。所有应用程式可以使用的 Windows 函式都在表头档案里预先宣告过。

在 Windows 程式中，使用 Windows 函式的方式通常与使用如 strlen 等 C 语言程式库函式的方式相同。主要的区别在於 C 语言程式库函式的机械码连结到您的程式码中，而 Windows 函式的程式码在您程式执行档外的 DLL 中。

当您执行 Windows 程式时，它通过一个称作「动态连结」的过程与 Windows 相接。一个 Windows 的 .EXE 档案中有使用到的不同动态连结程式库的参考资料，所使用的函式即在那些动态连结程式库中。当 Windows 程式被载入到记忆体中时，程式中的呼叫被指向 DLL 函式的入口。如果该 DLL 不在记忆体中，就把它载入到记忆体中。

当您连结 Windows 程式以产生一个可执行档案时，您必须连结程式开发环境提供的特定「引用程式库 (import library)」。这些引用程式库包含了动态连结程式库名称和所有 Windows 函式呼叫的引用资讯。连结程式使用该资讯在 .EXE 档案中建立一个表格，在载入程式时，Windows 使用它将呼叫转换为 Windows 函式。

WINDOWS 程式设计选项

为说明 Windows 程式设计的多种技术，本书提供了许多范例程式。这些程式使用 C 语言撰写并原原本本的使用 Windows API 来开发程式。我将这种方法

称作「古典」Windows 程式设计。这是我们在 1985 年为 Windows 1.0 写程式的方法，它今天仍是写作 Windows 程式的有效方法。

API 和记忆体模式

对于程式写作者来说，作业系统是由本身的 API 定义的。API 包含了所有应用程式能够使用的作业系统函式呼叫，同时包含了相关的资料型态和结构。在 Windows 中，API 还意味著一个特殊的程式架构，我们将在每章的开头进行研究。

一般而言，Windows API 自 Windows 1.0 以来一直保持一致，没什么重大改变。具有 Windows 98 程式写作经验的 Windows 程式写作者会对 Windows 1.0 程式的原始码感觉非常熟悉。API 改变的一种方式是在进行增强。Windows 1.0 支援不到 450 个函式呼叫，现在已有了上千种函式呼叫。

Windows API 和它的语法的最大变化来自于从 16 位元架构向 32 位元架构转化的过程中。Windows 从版本 1.0 到版本 3.1 使用 16 位元 Intel 8086、8088、和 286 微处理器上所谓的分段记忆体模式，由于相容性的原因，从 386 开始的 32 位元 Intel 微处理器也支援该模式。在这种模式下，微处理器暂存器的大小为 16 位元，因此 C 的 int 资料型态也是 16 位元宽。在分段记忆体模式下，记忆体位址由两个部分组成——一个 16 位元段 (segment) 指标和一个 16 位偏移量 (offset) 指标。从程式写作者的角度看，这非常凌乱并带来了 long 或 far 指标 (包括段位址和偏移量位址) 和 short 或 near 指标 (包括带有假定段位址的偏移量位址) 的区别。

从 Windows NT 和 Windows 95 开始，Windows 支援使用 Intel 386、486 和 Pentium 处理器 32 位元模式下的 32 位元平坦定址记忆体模式。C 语言的 int 资料型态也扩展为 32 位元的值。为 32 位元版本 Windows 编写的程式使用简单的平坦线性空间定址的 32 位元指标值。

用于 16 位元版本 Windows 的 API (Windows 1.0 到 Windows 3.1) 现在称作 Win16。用于 32 位元版本 Windows 的 API (Windows 95、Windows 98 和所有版本的 Windows NT) 现在称作 Win32。许多函式呼叫在从 Win16 到 Win32 的转变中保持相同，但有些需要增强。例如，图像座标点由 Win16 中的 16 位元值变为 Win32 中的 32 位元值。此外，某些 Win16 函式呼叫返回一个包含在 32 位元整数值中的二维座标点。这在 Win32 中不可能，因此增加的新函式呼叫以不同方式运作。

所有 32 位元版本的 Windows 都支援 Win16 API (以确保和旧有应用程式相容) 和 Win32 API (以运行新应用程式)。非常有趣的是，Windows NT 与 Windows 95 及 Windows 98 的工作方式不同。在 Windows NT 中，Win16 函式呼叫通过一

个转换层被转化为 Win32 函式呼叫，然後被作业系统处理。在 Windows 95 和 Windows 98 中，该操作正相反：Win32 函式呼叫通过转换层转换为 Win16 函式呼叫，再由作业系统处理。

在同一时刻有两个不同的 Windows API 集（至少名称不同）。Win32s（「s」代表「subset（子集）」）是一个 API，允许程式写作者编写在 Windows 3.1 上执行的 32 位元應用程式。该 API 仅支援已被 Win16 支援的 32 位元函式版本。此外，Windows 95 API 一度被称作 Win32c（「c」代表「compatibility（相容性）」），但该术语已被抛弃了。

现在，Windows NT 和 Windows 98 都被认为能够支援 Win32 API。然而，每个作业系统依然都支援某些不被别的作业系统支援的某些功能特性。因为它们的相同之处是相当可观的，所以有可能编写在两个作业系统下都可执行的程式。而且，人们普遍认为这两个产品最终会合而为一。

语言选项

使用 C 语言和原始的 API 不是编写 Windows 98 程式的唯一方法。然而，这种方法却提供给您最佳的性能、最强大的功能和在发掘 Windows 特性方面最大的灵活性。可执行档案相对较小且运行时不要求外部程式库（自然，Windows DLL 自身除外）。最重要的是，不管您最终以什么方式开发 Windows 應用程式，熟悉 API 会使您对 Windows 内部有更深入的了解。

虽然我认为学习古典的 Windows 程式设计对任何 Windows 程式写作者都是重要的，我没有必要建议使用 C 和 API 编写每个 Windows 應用程式。许多程式写作者，特别是那些为公司内部开发程式或在家编写娱乐程式的程式写作者喜欢轻松的开发环境，例如 Microsoft Visual Basic 或者 Borland Delphi（它结合了物件导向的 Pascal 版本）。这些环境使程式写作者将精力集中於應用程式的使用者介面和相关使用者介面物件的程式码上。要学习 Visual Basic，您也许需要参考 Microsoft Press 的一些其他图书，例如 Michael Halvorson 1996 年著的《Learn Visual Basic Now》。

在专业程式写作者中——特别是那些开发商业應用程式的程式写作者——Microsoft Visual C++ 和 Microsoft Foundation Class Library (MFC) 是近年来流行的选择。MFC 在一组 C++ 物件类别中封装了许多 Windows 程式设计中的琐碎细节。Jeff Prosise 的《Programming Windows with MFC, 第二版》(Microsoft Press, 1999 年) 提供了 MFC 程式的写作指南。

最近，Internet 和 World Wide Web 的流行大力推广著 Sun Microsystems 的 Java，这是一个受 C++ 启发却与微处理器无关的程式设计语言，而且结合了

可在几个作业系统平台上执行的图形应用程序开发工具组。Microsoft Press 有一本关于 Microsoft J++ (Microsoft 的 Java) 开发工具的好书,《Programming Visual J++ 6.0》(1998 年),由 Stephen R. Davis 著。

显然,很难说哪种方法更有利于开发 Windows 应用程序。更主要的是,也许是应用程序自身的特性决定了所使用的工具。不管您最后实际上使用什么工具写作程式,学习 Windows API 将使您更深入地了解 Windows 工作的方式。Windows 是一个复杂的系统,在 API 上增加一个程式写作层并未减少它的复杂性,仅仅是掩盖了它,早晚您会碰到它。了解 API 会给您更好的补救机会。

在原始的 Windows API 之上的任何软体层都必定将您限制在全部功能的一个子集内。您也许发现,例如,使用 Visual Basic 编写应用程序非常理想,然而它不允许您做一个或两个很简单的基本工作。在这种情况下,您将不得不使用原始的 API 呼叫。API 定义了作为 Windows 程式写作者所需的一切。没有什么方法比直接使用 API 更万能的了。

MFC 尤其问题百出。虽然它大幅简化了某些工作(例如 OLE),我却经常发现要让它们按我所想的去工作时,会在其他特性(例如 Document/View 架构)上碰壁。MFC 还不是 Windows 程式设计者所追求的灵丹妙药,很少有人认为它是一个好的物件导向设计的模型。MFC 程式写作者从他们使用的物件类别定义如何工作中受益颇深,并会发现他们经常参考 MFC 原始码,搞懂这些原始码是学习 Windows API 的好处之一。

程式开发环境

在本书中,假定您正使用 Microsoft Visual C++ 6.0,标准版、专业版和企业版都可以。经济的标准版足以应付本书中的程式设计需求。Visual C++ 还是 Visual Studio 6.0 中的一部分。

Microsoft Visual C++ 套装软体中包括 C 编译器和其他编译及连结 Windows 程式所需的档案和工具等。它还包括 Visual C++ Developer Studio,一个可编辑原始码、以交谈方式建立资源(如图示和对话方块)以及编辑、编译、执行和测试程式的环境。

如果您正使用 Visual C++ 5.0,则需要为 Windows 98 和 Windows NT 5.0 更新表头档案和引用程式库,这些东西可从 Microsoft 的网站上得到。在 <http://www.microsoft.com/msdn/>, 选择「**Downloads**」,然后选择「**Platform SDK**」(软体开发套件),您就能在选择的目录中下载和安装更新档案。要让 Microsoft Developer Studio 浏览这些目录,可以从「**Tool**」功能表项选择「**Options**」然后按下「**Directories**」标签。

Microsoft 网站上的 msdn 部分代表「Microsoft Developer Network (Microsoft 软体开发者网路)」。这是一个向程式写作者提供了经常更新的 CD-ROM 的计划, 这些 CD-ROM 中包含了程式写作者在 Windows 开发中所需的最新东西。您也可以订阅 MSDN, 这样就避免经常得从 Microsoft 的网站下载档案。

API 文件

本书不是 Windows API 权威的正式文件的替代品。那组文件不再以印刷形式出版, 它仅能从 CD-ROM 或 Internet 上取得。

当您安装 Visual C++ 6.0 时, 您将得到一个包括 API 文件的线上求助系统。您可通过订阅 MSDN 或使用 Microsoft 网站上的线上求助系统更新该文件。连接到 <http://www.microsoft.com/msdn/>, 并选择「MSDN Library Online」。

在 Visual C++ 6.0 中, 从「Help」功能表项选择「Contents」项目开启 MSDN 视窗。API 文件按树形结构组织, 寻找标有「Platform SDK」的部分, 所有在本书中引用的文件都来自於该部分。我将向您介绍如何从「Platform SDK」开始寻找以斜线分层分门别类的文件的位置。(我知道「Platform SDK」是整个 MSDN 知识库中较为晦涩的部分, 但我敢保证那是 Windows 程式设计的基本核心。) 例如, 对于如何在 Windows 程式中使用滑鼠的文件, 您可参考/ Platform SDK / User Interface Services / User Input / Mouse Input。

我在前面提到 Windows 大致分为 Kernel、User 和 GDI 子系统。kernel 介面在/ Platform SDK / Windows Base Services 中, User 介面函式在 / Platform SDK / User Interface Services 中, GDI 位於 / Platform SDK / Graphics and Multimedia Services / GDI 中。

编写第一个 WINDOWS 程式

现在是开始写些程式的时候了。为了便於对比, 让我们以一个非常短的 Windows 程式和一个简短的文字模式程式开始。这会帮助我们找到使用开发环境并感受建立和编译程式机制的正确方向。

文字模式 (Character-Mode) 模型

程式写作者们喜爱的一本书是《The C Programming Language》(Prentice Hall, 1978 年和 1988 年), 由 Brian W. Kernighan 和 Dennis M. Ritchie (亲切地称为 K&R) 编著。该书的第一章以一个显示「hello, world」的 C 语言程式开始。

这里是在《The C Programming Language》第一版第 6 页中出现的程式:

```
main ()
{
    printf ("hello, world\n") ;
}
```

以前 C 程式写作者在使用 printf 等 C 执行期程式库函式时，无需先宣告它们。但这是 90 年代，我们愿意给编译器一个在我们的程式中标出错误的机会。这里是在 K&R 第二版中修正的程式：

```
#include <stdio.h>
main ()
{
    printf ("hello, world\n") ;
}
```

该程式仍然是那么短。但它可通过编译并执行得很好，但当今许多程式写作者更愿意清楚地说明 main 函式的返回值，在这种情况下 ANSI C 规定该函式必须返回一个值：

```
#include <stdio.h>
int main ()
{
    printf ("hello, world\n") ;
    return 0 ;
}
```

我们还可以包括 main 的参数，把程式弄得更长一些，但让我们暂且这样就好了——包括一个 include 宣告、程式的进入点、一个对执行期程式库函式的呼叫和一个 return 语句。

同样效果的 Windows 程式

Windows 关于「hello, world」程式的等价程式有和文字模式版本完全相同的元件。它有一个 include 宣告、一个程式进入点、一个函式呼叫和一个 return 语句。下面便是该程式：

```
/*-----
HelloMsg.c -- Displays "Hello, Windows 98!" in a message box
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
{
    MessageBox (NULL, TEXT ("Hello, Windows 98!"), TEXT ("HelloMsg"), 0);
    return 0 ;
}
```

在剖析该程式之前，让我们看一下在 Visual C++ Developer Studio 中建

立新程式的方式。

首先,从 **File** 功能表中选 **New**。在 **New** 对话方块中,单击 **Projects** 页面标签,选择 **Win32 Application**。在 **Location** 栏中,选择一个子目录,在 **Project Name** 栏中,输入该专案的名称,此时该名称是 **HelloMsg**,这便是在 **Location** 栏中显示的目录的子目录。**Create New Workspace** 核取方块应该勾起来, **Platforms** 部分应该显示 **Win32**, 选择 **OK**。

将会出现一个标题为 **Win32 Application - Step 1 Of 1** 的对话方块,指出要建立一个 **Empty Project**, 并按下 **Finish** 按钮。

从 **File** 功能表中再次选择 **New**。在 **New** 对话方块中,选择 **Files** 页面标签,选择 **C++ Source File**。 **Add To Project** 核取方块应被选中,并应显示 **HelloMsg**。在 **File Name** 栏中输入 **HelloMsg.c**, 选中 **OK**。

现在您可输入上面所示的 **HELLOMSG.C** 档案,您也可以选择 **Insert** 功能表和 **File As Text** 选项从本书附带的 CD-ROM 上复制 **HELLOMSG.C** 的内容。

从结构上说, **HELLOMSG.C** 与 K&R 的「hello,world」程式是相同的。表头档案 **STDIO.H** 已被 **WINDOWS.H** 所代替,进入点 **main** 被 **WinMain** 所代替,而且 C 语言执行时期程式库函式 **printf** 被 Windows API 函式 **MessageBox** 所代替。然而,在程式中有许多新东西,包括几个陌生的大写识别字。

让我们从头开始。

表头档案

HELLOMSG.C 以一个前置处理器指示命令开始,实际上在每个用 C 编写的 Windows 程式的开头都可看到:

```
#include <windows.h>
```

WINDOWS.H 是主要的含入档案,它包含了其他 Windows 表头档案,这些表头档案的某些也包含了其他表头档案。这些表头档案中最重要和最基本的是:

WINDEF.H 基本型态定义。

WINNT.H 支援 Unicode 的型态定义。

WINBASE.H Kernel 函式。

WINUSER.H 使用者介面函式。

WINGDI.H 图形装置介面函式。

这些表头档案定义了 Windows 的所有资料型态、函式呼叫、资料结构和常数识别字,它们是 Windows 文件中的一个重要部分。使用 Visual C++ Developer Studio 的 **Edit** 功能表中的 **Find in Files** 搜索这些表头档案非常方便。您还可以在 Developer Studio 中打开这些表头档案并直接阅读它们。

程式进入点

正如在 C 程式中的进入点是函数 main 一样, Windows 程式的进入点是 WinMain, 总是像这样出现:

```
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
```

该进入点在 / Platform SDK / User Interface Services / Windowing / Windows / Window Reference / Window Functions 中有说明。它在 WINBASE.H 中宣告如下:

```
int
WINAPI
WinMain (
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nShowCmd
);
```

您会注意到我在 HELLOMSG.C 中做了许多小改动。第三个参数在 WINBASE.H 中定义为 LPSTR, 我将它改为 PSTR。这两种资料型态都定义在 WINNT.H 中, 作为指向字串的指标。LP 字首代表「长指标」, 这是 16 位元 Windows 下的产物。

我还在 WinMain 宣告中改变了两个参数的名称。许多 Windows 程式中的变数名使用一种称作「匈牙利表示法」的命名系统, 该系统在变数名称前面增加了表示变数资料型态的短字首, 我将在第三章更详细地讨论这个概念。现在仅需记住字首 i 表示 int、sz 表示「以零结束的字串」。

WinMain 函式宣告为返回一个 int 值。WINAPI 识别字在 WINDEF.H 定义, 语句如下:

```
#define WINAPI __stdcall
```

该语句指定了一个呼叫约定, 包括如何生产机械码以在堆叠中放置函式呼叫的参数。许多 Windows 函式呼叫宣告为 WINAPI。

WinMain 的第一个参数被称作「执行实体代号」。在 Windows 程式设计中, 代号仅是一个应用程式用来识别某些东西的数字。在这种情况下, 该代号唯一地标识该程式, 还需要它在其他 Windows 函式呼叫中作为参数。在 Windows 的早期版本中, 当同时运行同一程式多次时, 您便创建了该程式的「多个执行实体 (multiple instances)」。同一应用程式的所有执行实体共用程式和唯读的记忆体 (通常是例如功能表和对话方块模板的资源)。程式通过检查 hPrevInstance 参数就能够确定自身的其他执行实体是否正在运行。然後它可以略过一些繁杂的工作并从前面的执行实体将某些资料移到自己的资料区域。

在 32 位元 Windows 版本中, 该概念已被抛弃。传给 WinMain 的第二个参数

总是 NULL (定义为 0)。

WinMain 的第三个参数是用於执行程式的命令列。某些 Windows 应用程式利用它在程式启动时将档案载入记忆体。WinMain 的第四个参数指出程式最初显示的方式，可以是正常的或者是最大化地充满整个画面，或者是最小化显示在工作列中。我们将在第三章中介绍使用该参数的方法。

MessageBox 函式

MessageBox 函式用於显示短资讯。虽然，MessageBox 显示的小视窗不具有什么功能，实际上它被认为是一个对话方块。

MessageBox 的第一个参数通常是视窗代号，我们将在第三章介绍其含义。第二个参数是在讯息方块主体中显示的字串，第三个参数是出现在讯息方块标题列上的字串。在 HELLOMSG.C 中，这些文字字串的每一个都被封装在一个 TEXT 巨集中。通常您不必将所有字串都封装在 TEXT 巨集中，但如果想将您的程式转换为 Unicode 字元集，这确是一个好主意。我将在第二章详细讨论该问题。

MessageBox 的第四个参数可以是在 WINUSER.H 中定义的一组以字首 MB_ 开始的常数的组合。您可从第一组中选择一个常数指出希望在对话方块中显示的按钮：

#define	MB_OK	0x00000000L
#define	MB_OKCANCEL	0x00000001L
#define	MB_ABORTRETRYIGNORE	0x00000002L
#define	MB_YESNOCANCEL	0x00000003L
#define	MB_YESNO	0x00000004L
#define	MB_RETRYCANCEL	0x00000005L

如果在 HELLOMSG 中将第四个参数设置为 0，则仅显示「 **OK** 」按钮。可以使用 C 语言的 OR (|) 操作符号将上面显示的一个常数与代表内定按钮的常数组合：

#define	MB_DEFBUTTON1	0x00000000L
#define	MB_DEFBUTTON2	0x00000100L
#define	MB_DEFBUTTON3	0x00000200L
#define	MB_DEFBUTTON4	0x00000300L

还可以使用一个常数指出讯息方块中图示的外观：

#define	MB_ICONHAND	0x00000010L
#define	MB_ICONQUESTION	0x00000020L
#define	MB_ICONEXCLAMATION	0x00000030L
#define	MB_ICONASTERISK	0x00000040L

这些图示中的某些有替代名称：

#define	MB_ICONWARNING	MB_ICONEXCLAMATION
#define	MB_ICONERROR	MB_ICONHAND
#define	MB_ICONINFORMATION	MB_ICONASTERISK

```
#define MB_ICONSTOP MB_ICONHAND
```

虽然只有少数其他 MB_常数,但您可以自己参考表头档案或 / Platform SDK / User Interface Services / Windowing / Dialog Boxes / Dialog Box Reference / Dialog Box Functions 里的档案。

在本程式中, MessageBox 返回数值 1, 但更严格地说它返回 IDOK, IDOK 在 WINUSER.H 中定义, 等於 1。根据在讯息方块中显示的其他按钮, MessageBox 函式还可返回 IDYES、IDNO、IDCANCEL、IDABORT、IDRETRY 或 IDIGNORE。

这个小的 Windows 程式真的与 K&R 的「hello, world」程式有著同等效果吗? 您也许认为不是, 因为 MessageBox 函式并没有「hello, world」中 printf 函数所具有的潜在格式化文字能力。但我们将在下一章中看到编写类似 printf 的 MessageBox 版本的方法。

编译、连结和执行

当您准备编译 HELLOMSG 时, 您可从「Build」功能表中选择「Build Hellomsg.exe」, 或者按 **F7**, 或者在「Build」工具列中选择「Build」图示。(该图示的外观显示在「Build」功能表中。如果当前没有显示「Build」工具列, 您可从「Tools」功能表中选择「Customize」并选择「Toolbars」页面标签, 选中「Build」或者「Build MiniBar」。))

另一种方法, 您可从「Build」功能表中选择「Execute Hellomsg.exe」, 或者按「**Ctrl+F5**」, 或者在「Build」工具列单击「Execute Program」图示(该图示看上去像一个红的感叹号), 就会弹出一个讯息方块询问是否编译该程式。

正常情况下, 在编译阶段, 编译器从 C 原始码档案产生一个.OBJ(目标)档案。在连结阶段, 连结程式结合.OBJ 档案和.LIB(库)档案以建立.EXE(可执行)档案。通过在「Project」页面标签上选择「Settings」并单击「Link」页面标签可以查看这些库档案的列表。特别地, 您会注意到 KERNEL32.LIB、USER32.LIB 和 GDI32.LIB。这些是三个主要 Windows 子系统的「引用程式库」。它们包含了动态连结程式库的名称以及放进.EXE 档案的引用资讯。Windows 使用该资讯处理程式对 KERNEL32.DLL、USER32.DLL、GDI32.DLL 动态连结程式库中函数的呼叫。

在 Visual C++ Developer Studio 中, 您可用不同的设定编译和连结程式。内定情况下, 它们是「Debug」和「Release」。可执行档案被存放在以这些名称命名的子目录下。在 Debug 设定下, 资讯被附加到 .EXE 档案中, 这些资讯有助於测试程式和追踪原始码。

如果您喜欢在命令列下工作，附上的 CD-ROM 包含所有范例程式的 .MAK (make) 档案。(可通过「 [Tools](#) 」功能表选择「 [Options](#) 」，再选择「 [Build](#) 」页面标签，来告诉 Developer Studio 生成 make 档案。这里有一个核取方块需要勾选)。您需要执行在 Developer Studio 的 BIN 子目录下的 VCVARS32.BAT 来设置环境变数。要从命令列执行 make 档案，可以转到 HELLOMSG 目录并执行：

```
NMAKE /f HelloMsg.mak CFG="HelloMsg - Win32 Debug"
```

或者

```
NMAKE /f HelloMsg.mak CFG="HelloMsg - Win32 Release"
```

然後您可通过输入：

```
DEBUG\HELLOMSG
```

或者

```
RELEASE\HELLOMSG
```

从命令列执行 .EXE 档案。

我已经在本书附上的 CD-ROM 中对专案档案中的内定 Debug 设定做了一个改动。在「 [Project Settings](#) 」对话方块中，选择「 [C/C++](#) 」页面标签後，在「 [Preprocessor Definitions](#) 」栏中，我已定义了识别字 UNICODE。我将在下章中对此有更多的解释。

第二章 Unicode 简介

在第一章中，我已经预告，C 语言中在 Microsoft Windows 程式设计中扮演著重要角色的任何部分都会讲述到，您也许在传统文字模式程式设计中还尚未遇到过这些问题。宽字元集和 Unicode 差不多就是这样的问题。

简单地说，Unicode 扩展自 ASCII 字元集。在严格的 ASCII 中，每个字元用 7 位元表示，或者电脑上普遍使用的每字元有 8 位元宽；而 Unicode 使用全 16 位元字元集。这使得 Unicode 能够表示世界上所有的书写语言中可能用於电脑通讯的字元、象形文字和其他符号。Unicode 最初打算作为 ASCII 的补充，可能的话，最终将代替它。考虑到 ASCII 是电脑中最具支配地位的标准，所以这的确是一个很高的目标。

Unicode 影响到了电脑工业的每个部分，但也许会对作业系统和程式设计语言的影响最大。从这方面来看，我们已经上路了。Windows NT 从底层支援 Unicode（不幸的是，Windows 98 只是小部分支援 Unicode）。先天即被 ANSI 束缚的 C 程式设计语言通过对宽字元集的支援来支援 Unicode。下面将详细讨论这些内容。

自然，作为程式写作者，我们通常会面对许多繁重的工作。我已试图透过使本书中的所有程式「Unicode 化」来减轻负担。其含义会随著本章对 Unicode 的讨论而清晰起来。

字元集简史

虽然不能确定人类开始讲话的时间，但书写已有大约 6000 年的历史了。实际上，早期书写的内容是象形文字。每个字元都对应於发声的字母表则出现於大约 3000 年前。虽然人们过去使用的多种书写语言都用得好好的，但 19 世纪的几个发明者还是看到了更多的需求。Samuel F. B. Morse 在 1838 年到 1854 年间发明了电报，当时他还发明了一种电报上使用的代码。字母表中的每个字元对应於一系列短的和长的脉冲（点和破折号）。虽然其中大小写字母之间没有区别，但数字和标点符号都有了自己的代码。

Morse 代码并不是以其他图画的或印刷的象形文字来代表书写语言的第一个例子。1821 年到 1824 年之间，年轻的 Louis Braille 受到在夜间读写资讯的军用系统的启发，发明了一种代码，它用纸上突起的点作为代码来帮助盲人阅读。Braille 代码实际上是一种 6 位元代码，它把字元、常用字母组合、常用单字和标点进行编码。一个特殊的 escape 代码表示後续的字元代码应解释为大写。

一个特殊的 shift 代码允许后续代码被解释为数字。

Telex 代码，包括 Baudot（以一个法国工程师命名，该工程师死于 1903 年）以及一种被称为 CCITT #2 的代码（1931 年被标准化），都是包括字元和数字的 5 位元代码。

美国标准

早期电脑的字元码是从 Hollerith 卡片（号称不能被折叠、卷曲或毁伤）发展而来的，该卡片由 Herman Hollerith 发明并首次在 1890 年的美国人口普查中使用。6 位元字元码系统 BCDIC (Binary-Coded Decimal Interchange Code: 二进位编码十进位交换编码) 源自 Hollerith 代码，在 60 年代逐步扩展为 8 位元 EBCDIC，并一直是 IBM 大型主机的标准，但没使用在其他地方。

美国资讯交换标准码 (ASCII: American Standard Code for Information Interchange) 起始於 50 年代後期，最後完成於 1967 年。开发 ASCII 的过程中，在字元长度是 6 位元、7 位元还是 8 位元的问题上产生了很大的争议。从可靠性的观点来看不应使用替换字元，因此 ASCII 不能是 6 位元编码，但由於费用的原因也排除了 8 位元版本的方案（当时每位元的储存空间成本仍很昂贵）。这样，最终的字元码就有 26 个小写字母、26 个大写字母、10 个数字、32 个符号、33 个代号和一个空格，总共 128 个字元码。ASCII 现在记录在 ANSI X3.4-1986 字元集——用於资讯交换的 7 位元美国国家标准码 (7-Bit ASCII: 7-Bit American National Standard Code for Information Interchange)，由美国国家标准协会 (American National Standards Institute) 发布。图 2-1 中所示的 ASCII 字元码与 ANSI 文件中的格式相似。

ASCII 有许多优点。例如，26 个字母代码是连续的（在 EBCDIC 代码中就不是这样的）；大写字母和小写字母可通过改变一位元资料而相互转化；10 个数位的代码可从数值本身方便地得到（在 BCDIC 代码中，字元「0」的编码在字元「9」的後面！）

最棒的是，ASCII 是一个非常可靠的标准。在键盘、视讯显示卡、系统硬体、印表机、字体档案、作业系统和 Internet 上，其他标准都不如 ASCII 码流行而且根深蒂固。

	0-	1-	2-	3-	4-	5-	6-	7-
-0	NUL	DLE	SP	0	@	P	`	p
-1	SOH	DC1	!	1	A	Q	a	q
-2	STX	DC2	"	2	B	R	b	r
-3	ETX	DC3	#	3	C	S	c	s

-4	EOT	DC4	\$	4	D	T	d	t
-5	ENQ	NAK	%	5	E	U	e	u
-6	ACK	SYN	&	6	F	V	f	v
-7	BEL	ETB	'	7	G	W	g	w
-8	BS	CAN	(8	H	X	h	x
-9	HT	EM)	9	I	Y	I	y
-A	LF	SUB	*	:	J	Z	j	z
-B	VT	ESC	+	;	K	[k	{
-C	FF	FS	,	<	L	\	l	
-D	CR	GS	-	=	M]	m	}
-E	SO	RS	.	>	N	^	n	~
-F	SI	US	/	?	O	_	o	DEL

图 2-1 ASCII 字元集

国际方面

ASCII 的最大问题就是该缩写的第一个字母。ASCII 是一个真正的美国标准，所以它不能良好满足其他讲英语国家的需要。例如英国的英镑符号 (£) 在哪里？

英语使用拉丁（或罗马）字母表。在使用拉丁语字母表的书写语言中，英语中的单词通常很少需要重音符号（或读音符号）。即使那些传统惯例加上读音符号也无不当的英语单字，例如 **cöoperate** 或者 **résumé**，拼写中没有读音符号也会被完全接受。

但在美国以南、以北，以及大西洋地区的许多国家，在语言中使用读音符号很普遍。这些重音符号最初是为使拉丁字母表适合这些语言读音不同的需要。在远东或西欧的南部旅游，您会遇到根本不使用拉丁字母的语言，例如希腊语、希伯来语、阿拉伯语和俄语（使用斯拉夫字母表）。如果您向东走得更远，就会发现中国象形汉字，日本和朝鲜也采用汉字系统。

ASCII 的历史开始於 1967 年，此後它主要致力於克服其自身限制以更适合於非美国英语的其他语言。例如，1967 年，国际标准化组织 (ISO: International Standards Organization) 推荐一个 ASCII 的变种，代码 0x40、0x5B、0x5C、0x5D、0x7B、0x7C 和 0x7D「为国家使用保留」，而代码 0x5E、0x60 和 0x7E 标为「当国内要求的特殊字元需要 8、9 或 10 个空间位置时，可用於其他图形符号」。这显然不是一个最佳的国际解决方案，因为这并不能保证一致性。但这却显示了人们如何想尽办法为不同的语言来编码的。

扩展 ASCII

在小型电脑开发的初期，就已经严格地建立了 8 位元位元组。因此，如果使用一个位元组来保存字元，则需要 128 个附加的字元来补充 ASCII。1981 年，当最初的 IBM PC 推出时，视讯卡的 ROM 中烧有一个提供 256 个字元的字元集，这也成为 IBM 标准的一个重要组成部分。

最初的 IBM 扩展字元集包括某些带重音的字元和一个小写希腊字母表（在数学符号中非常有用），还包括一些块型和线状图形字元。附加的字元也被添加到 ASCII 控制字元的编码位置，这是因为大多数控制字元都不是拿来显示用的。

该 IBM 扩展字元集被烧进无数显示卡和印表机的 ROM 中，并被许多应用程序用於修饰其文字模式的显示方式。不过，该字元集并没有为所有使用拉丁字母表的西欧语言提供足够多的带重音字元，而且也不适用於 Windows。Windows 不需要图形字元，因为它有一个完全图形化的系统。

在 Windows 1.0（1985 年 11 月发行）中，Microsoft 没有完全放弃 IBM 扩展字元集，但它已退居第二重要位置。因为遵循了 ANSI 草案和 ISO 标准，纯 Windows 字元集被称作「ANSI 字元集」。ANSI 草案和 ISO 标准最终成为 ANSI/ISO 8859-1-1987，即「American National Standard for Information Processing-8-Bit Single-Byte Coded Graphic Character Sets-Part 1: Latin Alphabet No 1」，通常也简称为「Latin 1」。

在 Windows 1.0 的《Programmer's Reference》中印出了 ANSI 字元集的最初版本，如图 2-2 所示。

0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-	
-0	*	*	0	@	P	`	p	*	*	°	À	Ð	à	ð		
-1	*	*	!	1	A	Q	a	q	*	*	¡	±	Á	Ñ	á	ñ
-2	*	*	"	2	B	R	b	r	*	*	¢	²	Â	ò	â	ò
-3	*	*	#	3	C	S	c	s	*	*	£	³	Ã	ó	ã	ó
-4	*	*	\$	4	D	T	d	t	*	*	¤	´	Ä	ô	ä	ô
-5	*	*	%	5	E	U	e	u	*	*	¥	µ	Å	õ	å	õ
-6	*	*	&	6	F	V	f	v	*	*	¦	¶	Æ	ö	æ	ö
-7	*	*	'	7	G	W	g	w	*	*	§	•	Ç	*	ç	*
-8	*	*	(8	H	*	h	*	*	*	¨	,	È	ø	è	ø
-9	*	*)	9	I	Y	I	y	*	*	©	¹	É	Ù	é	ù
-A	*	*	*	:	J	Z	j	z	*	*	ª	º	Ê	Ú	ê	ú
-B	*	*	+	;	K	[k	{	*	*	«	»	Ë	Û	ë	û

-C	*	*	,	<	L	\	l		*	*	¬	¼	İ	Ü	ì	ü
-D	*	*	-	=	M]	m	}	*	*		½	Í	Ý	í	ý
-E	*	*	.	>	N	^	n	~	*	*	®	¾	Î	Ð	î	ð
-F	*	*	/	?	*	_	o	del	*	*	—	¿	Ï	ß	ï	ÿ

* - not applicable

图 2-2 Windows ANSI 字元集 (基於 ANSI/ISO 8859-1)

空方框表示该位置未定义字元。这与 ANSI/ISO 8859-1 的最终定义一致。ANSI/ISO 8859-1 仅显示了图形字元，而没有控制字元，因此没有定义 DEL。此外，代码 0xA0 定义为一个非断开的空格（这意味著在编排格式时，该字元不用於断开一行），代码 0xAD 是一个软连字元（表示除非在行尾断开单词时使用，否则不显示）。此外，ANSI/ISO 8859-1 将代码 0xD7 定义为乘号 (*)，0xF7 为除号 (/)。Windows 中的某些字体也定义了从 0x80 到 0x9F 的某些字元，但这些不是 ANSI/ISO 8859-1 标准的一部分。

MS-DOS 3.3 (1987 年 4 月发行) 向 IBM PC 用户引进了内码表 (code page) 的概念，Windows 也使用此概念。内码表定义了字元的映射代码。最初的 IBM 字元集被称作内码表 437，或者「MS-DOS Latin US」。内码表 850 就是「MS-DOS Latin 1」，它用附加的带重音字母（但 **不是** 图 2-2 所示的 Latin 1 ISO/ANSI 标准）代替了一些线形字元。其他内码表被其他语言定义。最低的 128 个代码总是相同的；较高的 128 个代码取决於定义内码表的语言。

在 MS-DOS 中，如果用户为 PC 的键盘、显示卡和印表机指定了一个内码表，然後在 PC 上创建、编辑和列印文件，一切都很正常，每件事都会保持一致。然而，如果用户试图与使用不同内码表的用户交换档案，或者在机器上改变内码表，就会产生问题。字元码与错误的字元相关联。应用程式能够将内码表资讯与文件一起保存来试图减少问题的产生，但该策略包括了某些在内码表间转换的工作。

虽然内码表最初仅提供了不包括带重音符号字母的附加拉丁字元集，但最终内码表的较高的 128 个字元还是包括了完整的非拉丁字母，例如希伯来语、希腊语和斯拉夫语。自然，如此多样会导致内码表变得混乱；如果少数带重音的字母未正确显示，那么整个文字便会混乱不堪而不可阅读。

内码表的扩展正是基於所有这些原因，但是还不够。斯拉夫语的 MS-DOS 内码表 855 与斯拉夫语的 Windows 内码表 1251 以及斯拉夫语的 Macintosh 内码表 10007 不同。每个环境下的内码表都是对该环境所作的标准字元集修正。IBM OS/2 也支援多种 EBCDIC 内码表。

但等一下，你会发现事情变得更糟糕。

双位元组字元集

迄今为止，我们已经看到了 256 个字元的字元集。但中国、日本和韩国的象形文字符号有大约 21,000 个。如何容纳这些语言而仍保持和 ASCII 的某种相容性呢？

解决方案（如果这个说法正确的话）是双位元组字元集（DBCS: double-byte character set）。DBCS 从 256 代码开始，就像 ASCII 一样。与任何行为良好的内码表一样，最初的 128 个代码是 ASCII。然而，较高的 128 个代码中的某些总是跟随著第二个位元组。这两个位元组一起（称作首位元组和跟随位元组）定义一个字元，通常是一个复杂的象形文字。

虽然中文、日文和韩文共用一些相同的象形文字，但显然这三种语言是不同的，而且经常是同一个象形文字在三种不同的语言中代表三件不同的事。Windows 支援四个不同的双位元组字元集：内码表 932（日文）、936（简体中文）、949（韩语）和 950（繁体汉字）。只有为这些国家（地区）生产的 Windows 版本才支援 DBCS。

双字元集问题并不是说字元由两个位元组代表。问题在於一些字元（特别是 ASCII 字元）由 1 个位元组表示。这会引起附加的程式设计问题。例如，字串中的字元数不能由字串的位元组数决定。必须剖析字串来决定其长度，而且必须检查每个位元组以确定它是否为双位元组字元的首位元组。如果有一个指向 DBCS 字串中间的指标，那么该字串前一个字元的位址是什么呢？惯用的解决方案是从开始的指标分析该字串！

Unicode 解决方案

我们面临的基本问题是世界上的书写语言不能简单地用 256 个 8 位元代码表示。以前的解决方案包括内码表和 DBCS 已被证明是不能满足需要的，而且也是笨拙的。那什么才是真正的解决方案呢？

身为程式写作者，我们经历过这类问题。如果事情太多，用 8 位元数值已经不能表示，那么我们就试更宽的值，例如 16 位元值。而且这很有趣的，正是 Unicode 被制定的原因。与混乱的 256 个字元代码映射，以及含有一些 1 位元组代码和一些 2 位元组代码的双位元组字元集不同，Unicode 是统一的 16 位元系统，这样就允许表示 65,536 个字元。这对表示所有字元及世界上使用象形文字的语言，包括一系列的数学、符号和货币单位符号的集合来说是充裕的。

明白 Unicode 和 DBCS 之间的区别很重要。Unicode 使用（特别在 C 程式设计语言环境里）「宽字元集」。「Unicode 中的每个字元都是 16 位元宽而不是

8 位元宽。」在 Unicode 中，没有单单使用 8 位元数值的意义存在。相比之下，在双位元组字元集中我们仍然处理 8 位元数值。有些位元组自身定义字元，而某些位元组则显示需要和另一个位元组共同定义一个字元。

处理 DBCS 字串非常杂乱，但是处理 Unicode 文字则像处理有秩序的文字。您也许会高兴地知道前 128 个 Unicode 字元（16 位元代码从 0x0000 到 0x007F）就是 ASCII 字元，而接下来的 128 个 Unicode 字元（代码从 0x0080 到 0x00FF）是 ISO 8859-1 对 ASCII 的扩展。Unicode 中不同部分的字元都同样基於现有的标准。这是为了便於转换。希腊字母表使用从 0x0370 到 0x03FF 的代码，斯拉夫语使用从 0x0400 到 0x04FF 的代码，美国使用从 0x0530 到 0x058F 的代码，希伯来语使用从 0x0590 到 0x05FF 的代码。中国、日本和韩国的象形文字（总称为 CJK）占用了从 0x3000 到 0x9FFF 的代码。

Unicode 的最大好处是这里只有一个字元集，没有一点含糊。Unicode 实际上是个人电脑行业中几乎每个重要公司共同合作的结果，并且它与 ISO 10646-1 标准中的代码是一一对应的。Unicode 的重要参考文献是《The Unicode Standard, Version 2.0》（Addison-Wesley 出版社，1996 年）。这是一本特别的书，它以其他文件少有的方式显示了世界上书写语言的丰富性和多样性。此外，该书还提供了开发 Unicode 的基本原理和细节。

Unicode 有缺点吗？当然有。Unicode 字串占用的记忆体是 ASCII 字串的两倍。（然而压缩档案有助於极大地减少档案所占的磁碟空间。）但也许最糟的缺点是：人们相对来说还不习惯使用 Unicode。身为程式写作者，这就是我们的工作。

宽字元和 C

对 C 程式写作者来说，16 位元字元的想法的确让人扫兴。一个 char 和一个位元组同宽是最不能确定的事情之一。没几个程式写作者清楚 ANSI/ISO 9899-1990，这是「美国国家标准程式设计语言——C」（也称作「ANSI C」）通过一个称作「宽字元」的概念来支援用多个位元组代表一字元的字元集。这些宽字元与常用的字元完美地共存。

ANSI C 也支援多位元组字元集，例如中文、日文和韩文版本 Windows 支援的字元集。然而，这些多位元组字元集被当成单位元组构成的字串看待，只不过其中一些字元改变了後续字元的含义而已。多位元组字元集主要影响 C 语言程式执行时期程式库函式。相比之下，宽字元比正常字元宽，而且会引起一些编译问题。

宽字元不需要是 Unicode。Unicode 是一种可能的宽字元集。然而，因为本

书的焦点是 Windows 而不是 C 执行的理论，所以我将把宽字元和 Unicode 作为同义语。

char 资料型态

假定我们都非常熟悉在 C 程式中使用 char 资料型态来定义和储存字元跟字串。但为了便於理解 C 如何处理宽字元，让我们先回顾一下可能在 Win32 程式中出现的标准字元定义。

下面的语句定义并初始化了一个只包含一个字元的变数：

```
char c = 'A' ;
```

变数 c 需要 1 个位元组来保存，并将用十六进位数 0x41 初始化，这是字母 A 的 ASCII 代码。

您可以像这样定义一个指向字串的指标：

```
char * p ;
```

因为 Windows 是一个 32 位元作业系统，所以指标变数 p 需要用 4 个位元组保存。您还可初始化一个指向字串的指标：

```
char * p = "Hello!" ;
```

像前面一样，变数 p 也需要用 4 个位元组保存。该字串保存在静态记忆体中并占用 7 个位元组——6 个位元组保存字串，另 1 个位元组保存终止符号 0。

您还可以像这样定义字元阵列：

```
char a[10] ;
```

在这种情况下，编译器为该阵列保留了 10 个位元组的储存空间。运算式 sizeof (a) 将返回 10。如果阵列是整体变数（即在所有函式外定义），您可使用像下面的语句来初始化一个字元阵列：

```
char a[] = "Hello!" ;
```

如果您将该阵列定义为一个函式的区域变数，则必须将它定义为一个 static 变数，如下：

```
static char a[] = "Hello!" ;
```

无论哪种情况，字串都储存在静态程式记忆体中，并在末尾添加 0，这样就需要 7 个位元组的储存空间。

宽字元

Unicode 或者宽字元都没有改变 char 资料型态在 C 中的含义。char 继续表示 1 个位元组的储存空间， sizeof (char) 继续返回 1。理论上，C 中 1 个位元组可比 8 位元长，但对我们大多数人来说，1 个位元组（也就是 1 个 char）是 8 位元宽。

C 中的宽字元基於 `wchar_t` 资料型态, 它在几个表头档案包括 `WCHAR.H` 中都有定义, 像这样:

```
typedef unsigned short wchar_t ;
```

因此, `wchar_t` 资料型态与无符号短整数型态相同, 都是 16 位元宽。

要定义包含一个宽字元的变数, 可使用下面的语句:

```
wchar_t c = 'A' ;
```

变数 `c` 是一个双位元组值 `0x0041`, 是 Unicode 表示的字母 A。(然而, 因为 Intel 微处理器从最小的位元组开始储存多位元组数值, 该位元组实际上是以 `0x41`、`0x00` 的顺序保存在记忆体中。如果检查 Unicode 文字的电脑储存应注意这一点。)

您还可定义指向宽字串的指标:

```
wchar_t * p = L"Hello!" ;
```

注意紧接在第一个引号前面的大写字母 `L` (代表「long」)。这将告诉编译器该字串按宽字元保存——即每个字元占用 2 个位元组。通常, 指标变数 `p` 要占用 4 个位元组, 而字串变数需要 14 个位元组——每个字元需要 2 个位元组, 末尾的 0 还需要 2 个位元组。

同样, 您还可以用下面的语句定义宽字元阵列:

```
static wchar_t a[] = L"Hello!" ;
```

该字串也需要 14 个位元组的储存空间, `sizeof (a)` 将返回 14。索引阵列 `a` 可得到单独的字元。`a[1]` 的值是宽字元「e」, 或者 `0x0065`。

虽然看上去更像一个印刷符号, 但第一个引号前面的 `L` 非常重要, 并且在两个符号之间必须没有空格。只有带有 `L`, 编译器才知道您需要将字串存为每个字元 2 位元组。稍後, 当我们看到使用宽字串而不是变数定义时, 您还会遇到第一个引号前面的 `L`。幸运的是, 如果忘记了包含 `L`, C 编译器通常会给提出警告或错误资讯。

您还可在单个字元文字前面使用 `L` 字首, 来表示它们应解释为宽字元。如下所示:

```
wchar_t c = L'A' ;
```

但通常这是不必要的, C 编译器会对该字元进行扩充, 使它成为宽字元。

宽字元程式库函式

我们都知道如何获得字串的长度。例如, 如果我们已经像下面这样定义了一个字串指标:

```
char * pc = "Hello!" ;
```

我们可以呼叫

```
int iLength = strlen (pc) ;
```

这时变数 `iLength` 将等於 6，也就是字串中的字元数。

太好了！现在让我们试著定义一个指向宽字元的指标：

```
wchar_t * pw = L"Hello!" ;
```

再次呼叫 `strlen`：

```
iLength = strlen (pw) ;
```

现在麻烦来了。首先，C 编译器会显示一条警告消息，可能是这样的内容：

```
'function' : incompatible types - from 'unsigned short *' to 'const char *'
```

这条消息的意思是：宣告 `strlen` 函式时，该函式应接收 `char` 类型的指标，但它现在却接收了一个 `unsigned short` 类型的指标。您仍然可编译并执行该程式，但您会发现 `iLength` 等於 1。为什么？

字串「Hello!」中的 6 个字元占用 16 位元：

```
0x0048 0x0065 0x006C 0x006C 0x006F 0x0021
```

Intel 处理器在记忆体中将其存为：

```
48 00 65 00 6C 00 6C 00 6F 00 21 00
```

假定 `strlen` 函式正试图得到一个字串的长度，并把第 1 个位元组作为字元开始计数，但接著假定如果下一个位元组是 0，则表示字串结束。

这个小练习清楚地说明了 C 语言本身和执行时期程式库函式之间的区别。编译器将字串 `L"Hello!"` 解释为一组 16 位元短整数型态资料，并将其保存在 `wchar_t` 阵列中。编译器还处理阵列索引和 `sizeof` 操作符，因此这些都能正常工作，但在连结时才添加执行时期程式库函式，例如 `strlen`。这些函式认为字串由单位元组字元组成。遇到宽字串时，函式就不像我们所希望那样执行了。

您可能要说：「噢，太麻烦了！」现在每个 C 语言程式库函式都必须重写以接受宽字元。但事实上并不是每个 C 语言程式库函式都需要重写，只是那些有字串参数的函式才需要重写，而且也不用由您来完成。它们已经重写完了。

`strlen` 函式的宽字元版是 `wcslen` (wide-character string length: 宽字串长度)，并且在 `STRING.H` (其中也说明了 `strlen`) 和 `WCHAR.H` 中均有说明。`strlen` 函式说明如下：

```
size_t __cdecl strlen (const char *) ;
```

而 `wcslen` 函式则说明如下：

```
size_t __cdecl wcslen (const wchar_t *) ;
```

这时我们知道，要得到宽字串的长度可以呼叫

```
iLength = wcslen (pw) ;
```

函式将返回字串中的字元数 6。请记住，改成宽位元组後，字串的字元长度不改变，只是位元组长度改变了。

您熟悉的所有带有字串参数的 C 执行时期程式库函式都有宽字元版。例如，`wprintf` 是 `printf` 的宽字元版。这些函式在 `WCHAR.H` 和含有标准函式说明的表

头档案中说明。

维护单一原始码

当然，使用 Unicode 也有缺点。第一点也是最主要的一点是，程式中的每个字串都将占用两倍的储存空间。此外，您将发现宽字元执行时期程式库中的函式比常规的函式大。出於这个原因，您也许想建立两个版本的程式——一个处理 ASCII 字串，另一个处理 Unicode 字串。最好的解决办法是维护既能按 ASCII 编译又能按 Unicode 编译的单一原始码档案。

虽然只是一小段程式，但由於执行时期程式库函式有不同的名称，您也要定义不同的字元，这将在处理前面有 L 的字串文字时遇到麻烦。

一个办法是使用 Microsoft Visual C++ 包含的 TCHAR.H 表头档案。该表头档案不是 ANSI C 标准的一部分，因此那里定义的每个函式和巨集定义的前面都有一条底线。TCHAR.H 为需要字串参数的标准执行时期程式库函式提供了一系列的替代名称（例如，_tprintf 和 _tcslen）。有时这些名称也称为「通用」函式名称，因为它们既可以指向函式的 Unicode 版也可以指向非 Unicode 版。

如果定义了名为 _UNICODE 的识别字，并且程式中包含了 TCHAR.H 表头档案，那么 _tcslen 就定义为 wcslen：

```
#define _tcslen wcslen
```

如果没有定义 UNICODE，则 _tcslen 定义为 strlen：

```
#define _tcslen strlen
```

等等。TCHAR.H 还用一个新的资料型态 TCHAR 来解决两种字元资料型态的问题。如果定义了 _UNICODE 识别字，那么 TCHAR 就是 wchar_t：

```
typedef wchar_t TCHAR ;
```

否则，TCHAR 就是 char：

```
typedef char TCHAR ;
```

现在开始讨论字串文字中的 L 问题。如果定义了 _UNICODE 识别字，那么一个称作 __T 的巨集就定义如下：

```
#define __T(x) L##x
```

这是相当晦涩的语法，但合乎 ANSI C 标准的前置处理器规范。那一对井字号称为「粘贴符号 (token paste)」，它将字母 L 添加到巨集引数上。因此，如果巨集引数是 "Hello!"，则 L##x 就是 L"Hello!"。

如果没有定义 _UNICODE 识别字，则 __T 巨集只简单地定义如下：

```
#define __T(x) x
```

此外，还有两个巨集与 __T 定义相同：

```
#define _T(x) __T(x)
```

```
#define _TEXT(x) __T(x)
```

在 Win32 console 程式中使用哪个巨集，取决於您喜欢简洁还是详细。基本本地，必须按下述方法在 `_T` 或 `_TEXT` 巨集内定义字串文字：

```
_TEXT ("Hello!")
```

这样做的话，如果定义了 `_UNICODE`，那么该串将解释为宽字元的组合，否则解释为 8 位元的字元字串。

宽字元和 WINDOWS

Windows NT 从底层支援 Unicode。这意味著 Windows NT 内部使用由 16 位元字元组成的字串。因为世界上其他许多地方还不使用 16 位元字串，所以 Windows NT 必须经常将字串在作业系统内转换。Windows NT 可执行为 ASCII、Unicode 或者 ASCII 和 Unicode 混合编写的程式。即，Windows NT 支援不同的 API 函式呼叫，这些函式接受 8 位元或 16 位元的字串（我们将马上看到这是如何动作的。）

相对於 Windows NT，Windows 98 对 Unicode 的支援要少得多。只有很少的 Windows 98 函式呼叫支援宽字串（这些函式列在《Microsoft Knowledge Base article Q125671》中；它们包括 `MessageBox`）。如果要发行的程式中只有一个 .EXE 档案要求在 Windows NT 和 Windows 98 下都能执行，那么就不应该使用 Unicode，否则就不能在 Windows 98 下执行；尤其程式不能呼叫 Unicode 版的 Windows 函式。这样，将来发行 Unicode 版的程式时会处於更有利的位置，您应试著编写既为 ASCII 又为 Unicode 编译的原始码。这就是本书中所有程式的编写方式。

Windows 表头档案类型

正如您在第一章所看到的那样，一个 Windows 程式包括表头档案 `WINDOWS.H`。该档案包括许多其他表头档案，包括 `WINDEF.H`，该档案中有许多在 Windows 中使用的基本型态定义，而且它本身也包括 `WINNT.H`。`WINNT.H` 处理基本的 Unicode 支援。

`WINNT.H` 的前面包含 C 的表头档案 `CTYPE.H`，这是 C 的众多表头档案之一，包括 `wchar_t` 的定义。`WINNT.H` 定义了新的资料型态，称作 `CHAR` 和 `WCHAR`：

```
typedef char CHAR ;
typedef wchar_t WCHAR ;      // wc
```

当您需要定义 8 位元字元或者 16 位元字元时，推荐您在 Windows 程式中使用的资料型态是 `CHAR` 和 `WCHAR`。`WCHAR` 定义後面的注释是匈牙利标记法的建议：一个基於 `WCHAR` 资料型态的变数可在前面附加上字母 `wc` 以说明一个宽字元。

`WINNT.H` 表头档案进而定义了可用做 8 位元字串指标的六种资料型态和四个可用做 `const` 8 位元字串指标的资料型态。这里精选了表头档案中一些实用的

说明资料型态语句:

```
typedef CHAR * PCHAR, * LPCH, * PCH, * NPSTR, * LPSTR, * PSTR ;
typedef CONST CHAR * LPCCH, * PCCH, * LPCSTR, * PCSTR ;
```

字首 N 和 L 表示「near」和「long」, 指的是 16 位元 Windows 中两种大小不同的指标。在 Win32 中 near 和 long 指标没有区别。

类似地, WINNT.H 定义了六种可作为 16 位元字串指标的资料型态和四种可作为 const 16 位元字串指标的资料型态:

```
typedef WCHAR * PWCHAR, * LPWCH, * PWCH, * NWPSTR, * LPWSTR, * PWSTR ;
typedef CONST WCHAR * LPCWCH, * PCWCH, * LPCWSTR, * PCWSTR ;
```

至此, 我们有了资料型态 CHAR (一个 8 位的 char) 和 WCHAR (一个 16 位的 wchar_t), 以及指向 CHAR 和 WCHAR 的指标。与 TCHAR.H 一样, WINNT.H 将 TCHAR 定义为一般的字元类型。如果定义了识别字 UNICODE (没有底线), 则 TCHAR 和指向 TCHAR 的指标就分别定义为 WCHAR 和指向 WCHAR 的指标; 如果没有定义识别字 UNICODE, 则 TCHAR 和指向 TCHAR 的指标就分别定义为 char 和指向 char 的指标:

```
#ifdef UNICODE
typedef WCHAR TCHAR, * PTCHAR ;
typedef LPWSTR LPTCH, PTCH, PTSTR, LPTSTR ;
typedef LPCWSTR LPCTSTR ;
#else
typedef char TCHAR, * PTCHAR ;
typedef LPSTR LPTCH, PTCH, PTSTR, LPTSTR ;
typedef LPCSTR LPCTSTR ;
#endif
```

如果已经在某个表头档案或者其他表头档案中定义了 TCHAR 资料型态, 那么 WINNT.H 和 WCHAR.H 表头档案都能防止其重复定义。不过, 无论何时在程式中使用其他表头档案时, 都应在所有其他表头档案之前包含 WINDOWS.H。

WINNT.H 表头档案还定义了一个巨集, 该巨集将 L 添加到字串的第一个引号前。如果定义了 UNICODE 识别字, 则一个称作 __TEXT 的巨集定义如下:

```
#define __TEXT(quote) L##quote
```

如果没有定义识别字 UNICODE, 则像这样定义 __TEXT 巨集:

```
#define __TEXT(quote) quote
```

此外, TEXT 巨集可这样定义:

```
#define TEXT(quote) __TEXT(quote)
```

这与 TCHAR.H 中定义 __TEXT 巨集的方法一样, 只是不必操心底线。我将在本书中使用这个巨集的 TEXT 版本。

这些定义可使您在同一程式中混合使用 ASCII 和 Unicode 字串, 或者编写一个可被 ASCII 或 Unicode 编译的程式。如果您希望明确定义 8 位元字元变数和字串, 请使用 CHAR、PCHAR (或者其他), 以及带引号的字串。为明确地使用

16 位元字元变数和字串, 请使用 WCHAR、PWCHAR, 并将 L 添加到引号前面。对於是 8 位还是 16 位取决於 UNICODE 识别字的定义的变数或字串, 要使用 TCHAR、PTCHAR 和 TEXT 巨集。

Windows 函式呼叫

从 Windows 1.0 到 Windows 3.1 的 16 位元 Windows 中, MessageBox 函式位於动态连结程式库 USER.EXE。在 Windows 3.1 软体开发套件的 WINDOWS.H 中, MessageBox 函式定义如下:

```
int WINAPI MessageBox (HWND, LPCSTR, LPCSTR, UINT) ;
```

注意, 函式的第二个、第三个参数是指向常数字串的指标。当编译连结一个 Win16 程式时, Windows 并不处理 MessageBox 呼叫。程式.EXE 档案中的表格, 允许 Windows 将该程式的呼叫与 USER 中的 MessageBox 函式动态连结起来。

32 位的 Windows (即所有版本的 Windows NT, 以及 Windows 95 和 Windows 98) 除了含有与 16 位相容的 USER.EXE 以外, 还含有一个称为 USER32.DLL 的动态连结程式库, 该动态连结程式库含有 32 位元使用者介面函式的进入点, 包括 32 位元的 MessageBox。

这就是 Windows 支援 Unicode 的关键: 在 USER32.DLL 中, 没有 32 位元 MessageBox 函式的进入点。实际上, 有两个进入点, 一个名为 MessageBoxA(ASCII 版), 另一个名为 MessageBoxW(宽字元版)。用字串作参数的每个 Win32 函式都在作业系统中有两个进入点! 幸运的是, 您通常不必关心这个问题, 程式中只需使用 MessageBox。与 TCHAR 表头档案一样, 每个 Windows 表头档案都有我们需要的技巧。

下面是 MessageBoxA 在 WINUSER.H 中定义的方法。这与 MessageBox 早期的定义很相似:

```
WINUSERAPI int WINAPI MessageBoxA (      HWND hWnd, LPCSTR lpText,  
                                     LPCSTR lpCaption, UINT uType) ;
```

下面是 MessageBoxW:

```
WINUSERAPI int WINAPI MessageBoxW (HWND hWnd, LPCWSTR lpText,  
                                   LPCWSTR lpCaption, UINT uType) ;
```

注意, MessageBoxW 函式的第二个和第三个参数是指向宽字元的指标。

如果需要同时使用并分别匹配 ASCII 和宽字元函式呼叫, 那么您可在 Windows 程式中明确地使用 MessageBoxA 和 MessageBoxW 函式。但大多数程式写作者将继续使用 MessageBox。根据是否定义了 UNICODE, MessageBox 将与 MessageBoxA 或 MessageBoxW 一样。在 WINUSER.H 中完成这一技巧时, 程式相当琐碎:

```
#ifndef UNICODE
#define MessageBox MessageBoxW
#else
#define MessageBox MessageBoxA
#endif
```

这样，如果定义了 UNICODE 识别字，那么程式中所有的 MessageBox 函式呼叫实际上就是 MessageBoxW 函式；否则，就是 MessageBoxA 函式。

执行该程式时，Windows 将程式中不同的函式呼叫与不同的 Windows 动态连结程式库的进入点连结。虽然只有少数例外，但是，在 Windows 98 中不能执行 Unicode 版的 Windows 函式。虽然这些函式有进入点，但通常返回错误代码。应用程式注意这些返回的错误并采取一些合理的动作。

Windows 的字串函式

正如前面谈到的，Microsoft C 包括宽字元和需要字串参数的 C 语言执行时期程式库函式的所有普通版本。不过，Windows 复制了其中一部分。例如，下面是 Windows 定义的一组字串函式，这些函式用来计算字串长度、复制字串、连接字串和比较字串：

```
lLength = lstrlen (pString) ;
pString = lstrcpy (pString1, pString2) ;
pString = lstrcpyn (pString1, pString2, iCount) ;
pString = lstrcat (pString1, pString2) ;
iComp = lstrcmp (pString1, pString2) ;
iComp = lstrcmpi (pString1, pString2) ;
```

这些函式与 C 程式库中对应的函式功能相同。如果定义了 UNICODE 识别字，那么这些函式将接受宽字串，否则只接受常规字串。宽字串版的 lstrlenW 函式可在 Windows 98 中执行。

在 Windows 中使用 printf

有文字模式、命令列 C 语言程式写作历史的程式写作者往往特别喜欢 printf 函式。即使可以使用更简单的命令（例如 puts），但 printf 出现在 Kernighan 和 Ritchie 的「hello, world」程式中一点也不会令人惊奇。我们知道，增强後的「hello, world」最终还是需要 printf 的格式化输出，因此我们最好从头开始就使用它。

但有个坏消息：在 Windows 程式中不能使用 printf。虽然 Windows 程式中可以使用大多数 C 的执行时期程式库——实际上，许多程式写作者更愿意使用 C 记忆体管理和档案 I/O 函式而不是 Windows 中等效的函式——Windows 对标准输入和标准输出没有概念。在 Windows 程式中可使用 fprintf，而不是 printf。

还有一个好消息，那就是仍然可以使用 sprintf 及 sprintf 系列中的其他函式来显示文字。这些函式除了将内容格式化输出到函式第一个参数所提供的字串缓冲区以外，其功能与 printf 相同。然後便可对该字串进行操作（例如将其传给 MessageBox）。

如果您从未使用过 sprintf（我第一次开始写 Windows 程式时也没用过此函式），这里有一个简短的执行实体，printf 函式说明如下：

```
int printf (const char * szFormat, ...) ;
```

第一个参数是一个格式字串，後面是与格式字串中的代码相对应的不同类型多个参数。

sprintf 函式定义如下：

```
int sprintf (char * szBuffer, const char * szFormat, ...) ;
```

第一个参数是字元缓冲区；後面是一个格式字串。Sprintf 不是将格式化结果标准输出，而是将其存入 szBuffer。该函式返回该字串的长度。在文字模式程式设计中，

```
printf ("The sum of %i and %i is %i", 5, 3, 5+3) ;
```

的功能相同於

```
char szBuffer [100] ;
sprintf (szBuffer, "The sum of %i and %i is %i", 5, 3, 5+3) ;
puts (szBuffer) ;
```

在 Windows 中，使用 MessageBox 显示结果优於 puts。

几乎每个人都经历过，当格式字串与被格式化的变数不合时，可能使 printf 执行错误并可能造成程式当掉。使用 sprintf 时，您不但要担心这些，而且还有一个新的负担：您定义的字串缓冲区必须足够大以存放结果。Microsoft 专用函式_sprintf 解决了这一问题，此函式引进了另一个参数，表示以字元计算的缓冲区大小。

vsprintf 是 sprintf 的一个变形，它只有三个参数。vsprintf 用於执行有多个参数的自订函式，类似 printf 格式。vsprintf 的前两个参数与 sprintf 相同：一个用於保存结果的字元缓冲区和一个格式字串。第三个参数是指向格式化参数阵列的指标。实际上，该指标指向在堆叠中供函式呼叫的变数。va_list、va_start 和 va_end 巨集（在 STDARG.H 中定义）帮助我们处理堆叠指标。本章最後的 SCRNSIZE 程式展示了使用这些巨集的方法。使用 vsprintf 函式，sprintf 函式可以这样编写：

```
int sprintf (char * szBuffer, const char * szFormat, ...)
{
    int    iReturn ;
    va_list pArgs ;
    va_start (pArgs, szFormat) ;
    iReturn = vsprintf (szBuffer, szFormat, pArgs) ;
}
```

```
va_end (pArgs) ;  
return iReturn ;  
}
```

va_start 巨集将 pArg 设置为指向一个堆叠变数，该变数位址在堆叠参数 szFormat 的上面。

由於许多 Windows 早期程式使用了 sprintf 和 vsprintf，最终导致 Microsoft 向 Windows API 中增添了两个相似的函式。Windows 的 wsprintf 和 wvsprintf 函式在功能上与 sprintf 和 vsprintf 相同，但它们不能处理浮点格式。

当然，随著宽字元的发表，sprintf 类型的函式增加许多，使得函式名称变得极为混乱。表 2-1 列出了 Microsoft 的 C 执行时期程式库和 Windows 支援的所有 sprintf 函式。

表 2-1

	ASCII	宽字元	常规
参数的变数个数			
标准版	sprintf	swprintf	_stprintf
最大长度版	_snprintf	_snwprintf	_sntprintf
Windows 版	wsprintfA	wsprintfW	wsprintf
参数阵列的指标			
标准版	vsprintf	vswprintf	_vstprintf
最大长度版	_vsnprintf	_vsnwprintf	_vsntprintf
Windows 版	wvsprintfA	wvsprintfW	wvsprintf

在宽字元版的 sprintf 函式中，将字串缓冲区定义为宽字串。在宽字元版的所有这些函式中，格式字串必须是宽字串。不过，您必须确保传递给这些函式的其他字串也必须由宽字元组成。

格式化讯息方块

程式 2-1 所示的 SCRNSIZE 程式展示了如何实作 MessageBoxPrintf 函式，该函式有许多参数并能像 printf 那样编排它们的格式。

程式 2-1 SCRNSIZE

```
SCRNSIZE.C  
/*-----  
-  
SCRNSIZE.C --      Displays screen size in a message box  
                  (c) Charles Petzold, 1998
```

```

-----
*/
#include <windows.h>
#include <tchar.h>
#include <stdio.h>

int CDECL MessageBoxPrintf (TCHAR * szCaption, TCHAR * szFormat, ...)
{
    TCHAR    szBuffer [1024] ;
    va_list pArgList ;

    // The va_start macro (defined in STDARG.H) is usually equivalent to:
    // pArgList = (char *) &szFormat + sizeof (szFormat) ;

    va_start (pArgList, szFormat) ;

    // The last argument to wvsprintf points to the arguments

    _vsntprintf (    szBuffer, sizeof (szBuffer) / sizeof (TCHAR),
                    szFormat, pArgList) ;

    // The va_end macro just zeroes out pArgList for no good reason
    va_end (pArgList) ;
    return MessageBox (NULL, szBuffer, szCaption, 0) ;
}

int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
{
    int cxScreen, cyScreen ;
    cxScreen = GetSystemMetrics (SM_CXSCREEN) ;
    cyScreen = GetSystemMetrics (SM_CYSCREEN) ;

    MessageBoxPrintf (    TEXT ("ScrnSize"),
                        TEXT ("The screen is %i pixels wide by %i pixels high."),
                        cxScreen, cyScreen) ;

    return 0 ;
}

```

经由从 GetSystemMetrics 函式得到的资讯，该程式以图素为单位显示了视讯显示的宽度和高度。GetSystemMetrics 是一个能用来获得 Windows 中不同物件的尺寸资讯的函式。事实上，我将在第四章用 GetSystemMetrics 函式向您展示如何在一个 Windows 视窗中显示和滚动多行文字。

本书与国际化

为国际市场准备的 Windows 程式不光要使用 Unicode。国际化超出了本书的范围，但在 Nadine Kano 所写的《Developing International Software for

Windows 95 and Windows NT》(Microsoft Press, 1995 年)一书中涉猎了许多。

本书中的程式写作时被限制成既可使用也可不使用定义的 UNICODE 识别字来编译。这包括对所有字元和字串定义使用 TCHAR, 对字串文字使用 TEXT 巨集, 以及注意不要混淆位元组和字元。例如, 注意 SCRNSIZE 中的 `_vsntprintf` 呼叫。第二个参数是缓冲区的字元大小。通常, 您使用 `sizeof (szBuffer)`。但如果缓冲区中有宽字元, 则返回的不是缓冲区的字元长度, 而是缓冲区的位元组大小。您必须用 `sizeof (TCHAR)` 将其分开。

通常, 在 Visual C++ Developer Studio 中, 可使用两种不同的设定来编译程式: Debug 和 Release。为简便起见, 对本书的范例程式, 我已修改了 Debug 设定, 以便於定义 UNICODE 识别字。如果程式使用了需要字串作参数的 C 程式库函式, 那么 UNICODE 识别字也在 Debug 设定中定义 (要了解这是在哪里完成的, 请从「Project」功能表中选择「Settings」, 然後单击「C/C++」标签)。使用这种方式, 这些程式就可以方便地被重新编译和连结以供测试。

本书中所有程式——无论是否为 Unicode 编译——都可以在 Windows NT 下执行。只有极少数情况例外。本书中按 Unicode 编译的程式不能在 Windows 98 中执行, 而非 Unicode 版则可以。本章和第一章的程式就是两个特例。MessageBoxW 是 Windows 98 支援的少数宽字元 Windows 函式之一。在 SCRNSIZE.C 中, 如果用 Windows 函式 `wprintf` 代替了 `_vsntprintf` (您还必须删除该函式的第二个参数), 那么 SCRNSIZE.C 的 Unicode 版将不能在 Windows 98 下执行, 这是因为 Windows 98 不支援 `wprintfW`。

在本书的後面 (特别在第六章, 介绍键盘的使用时), 我们将看到, 编写能处理远东版 Windows 双字元集的 Windows 程式不是一件容易的事情。本书没有说明如何去做, 并且基於这个原因, 本书中的某些非 Unicode 版本的程式在远东版的 Windows 下不能正常执行。这也是 Unicode 对将来的程式设计如此重要的一条理由。Unicode 允许程式更容易地跨越国界。

第三章 视窗和讯息

在前两章，程式使用了同一个函式 `MessageBox` 来向使用者输出文字。`MessageBox` 函式会建立一个「视窗」。在 Windows 中，「视窗」一词有确切的含义。一个视窗就是萤幕上的一个矩形区域，它接收使用者的输入并以文字或图形的格式显示输出内容。

`MessageBox` 函式建立一个视窗，但这只是一个功能有限的特殊视窗。讯息视窗有一个带关闭按钮的标题列、一个选项图示、一行或多行文字，以及最多四个按钮。当然，必须选择 Windows 提供给您的图示与按钮。

`MessageBox` 函式非常有用，但下面不会过多地使用它。我们不能在讯息方块中显示图形，而且也不能在讯息方块中添加功能表。要添加这些物件，就需要建立自己的视窗，现在就开始。

自己的视窗

建立视窗很简单，只需呼叫 `CreateWindow` 函式即可。

好啦，虽然建立视窗的函式的确名为 `CreateWindow`，而且您也能在 `/Platform SDK/User Interface Services/Windowing/Windows/Window Reference/Window Functions` 找到此文件，但您将发现 `CreateWindow` 的第一个参数就是所谓的「视窗类别名称」，并且该视窗类别连接所谓的「视窗讯息处理程式」。在我们呼叫 `CreateWindow` 之前，有一点背景知识会对您大有帮助。

总体结构

进行 Windows 程式设计，实际上是在进行一种物件导向的程式设计 (OOP)。这一点在 Windows 中使用得最多的物件上表现最为明显。这种物件正是 Windows 之所以命名为「Windows」的原因，它具有人格化的特徵，甚至可能会在您的梦中出现，这就是那个叫做「视窗」的东西。

桌面上最明显的视窗就是應用程式视窗。这些视窗含有显示程式名称的标题列、功能表甚至可能还有工具列和卷动列。另一类视窗是对话方块，它可以有标题列也可以没有标题列。

装饰对话方块表面的还有各式各样的按键、单选按钮、核取方块、清单方块、卷动列和文字输入区域。其中每一个小的视觉物件都是一个视窗。更确切地说，这些都称为「子视窗」或「控制项视窗」或「子视窗控制项」。

作为物件，使用者会在萤幕上看到这些视窗，并通过键盘和滑鼠直接与它

们进行交互操作。更有趣的是，程式作者的观点与使用者的观点极其类似。视窗以「讯息」的形式接收视窗的输入，视窗也用讯息与其他视窗通讯。对讯息的理解将是学习如何写作 Windows 程式所必须越过的障碍之一。

这有一个 Windows 的讯息范例：我们知道，大多数的 Windows 程式都有大小合适的應用程式视窗。也就是说，您能够通过滑鼠拖动视窗的边框来改变视窗的大小。通常，程式将通过改变视窗中的内容来回应这种大小的变化。您可能会猜测（并且您也是正确的），是 Windows 本身而不是應用程式在处理与使用者重新调整视窗大小相关的全部杂乱程式。由於應用程式能改变其显示的样子，所以它也「知道」视窗大小改变了。

應用程式是如何知道使用者改变了视窗的大小的呢？由於程式写作者习惯了往常的文字模式程式，作业系统没有设置将此类讯息通知给使用者的机制。问题的关键在於理解 Windows 所使用的架构。当使用者改变视窗的大小时，Window 给程式发送一个讯息指出新视窗的大小。然後程式就可以调整视窗中的内容，以回应大小的变化。

「Windows 给程式发送讯息。」我们希望读者不要对这句话视而不见。它到底表达了什么意思呢？我们在这里讨论的是程式码，而不是一个电子邮件系统。作业系统怎么给程式发送讯息呢？

其实，所谓「Windows 给程式发送讯息」，是指 Windows 呼叫程式中的一个函式，该函式的参数描述了这个特定讯息。这种位於 Windows 程式中的函式称为「视窗讯息处理程式」。

无疑，读者对程式呼叫作业系统的做法是很熟悉的。例如，程式在打开磁片档案时就要使用有关的系统呼叫。读者所不习惯的，可能是作业系统呼叫程式，而这正是 Windows 物件导向架构的基础。

程式建立的每一个视窗都有相关的视窗讯息处理程式。这个视窗讯息处理程式是一个函式，既可以在程式中，也可以在动态连结程式库中。Windows 通过呼叫视窗讯息处理程式来给视窗发送讯息。视窗讯息处理程式根据此讯息进行处理，然後将控制传回给 Windows。

更确切地说，视窗通常是在「视窗类别」的基础上建立的。视窗类别标识了处理视窗讯息的视窗讯息处理程式。使用视窗类别使多个视窗能够属於同一个视窗类别，并使用同一个视窗讯息处理程式。例如，所有 Windows 程式中的所有按钮均依据同一个视窗类别。这个视窗类别与一个处理所有按钮讯息的视窗讯息处理程式（位於 Windows 的动态连结程式库中）联结。

在物件导向的程式设计中，物件是程式与资料的组合。视窗是一种物件，其程式是视窗讯息处理程式。资料是视窗讯息处理程式保存的资讯和 Windows

为每个视窗以及系统中那个视窗类别保存的资讯。

视窗讯息处理程式处理给视窗发送讯息。这些讯息经常是告知视窗，使用者正使用键盘或者滑鼠进行输入。这正是按键视窗知道它被「按下」的奥妙所在。在视窗大小改变，或者视窗表面需要重画时，由其他讯息通知视窗。

Windows 程式开始执行後，Windows 为该程式建立一个「讯息佇列」。这个讯息佇列用来存放该程式可能建立的各种不同视窗的讯息。程式中有一小段程式码，叫做「讯息回圈」，用来从佇列中取出讯息，并且将它们发送给相应的视窗讯息处理程式。有些讯息直接发送给视窗讯息处理程式，不用放入讯息佇列中。

如果您对这段 Windows 架构过於简略的描述将信将疑，就让我们去看看在实际的程式中，视窗、视窗类别、视窗讯息处理程式、讯息佇列、讯息回圈和视窗讯息是如何相互配合的。这或许会对您有些帮助。

HELLOWIN 程式

建立一个视窗首先需要注册一个视窗类别，那需要一个视窗讯息处理程式来处理视窗讯息。处理视窗讯息对每个 Windows 程式都带来了些负担。程式 3-1 所示的 HELLOWIN 程式中整个做的事情差不多就是料理这些事情。

程式 3-1 HELLOWIN

```

HELLOWIN.C
/*-----
    HELLOWIN.C -- Displays "Hello, Windows 98!" in client area
    (c) Charles Petzold, 1998
    -----*/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("HelloWin") ;
    HWND  hwnd ;
    MSG   msg ;
    WNDCLAS  wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;

```

```
wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName    = NULL ;
wndclass.lpszClassName  = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow( szAppName,    // window class name
                    TEXT ("The Hello Program"), // window caption
                    WS_OVERLAPPEDWINDOW, // window style
                    CW_USEDEFAULT,  // initial x position
                    CW_USEDEFAULT,  // initial y position
                    CW_USEDEFAULT,  // initial x size
                    CW_USEDEFAULT,  // initial y size
                    NULL,            // parent window handle
                    NULL,           // window menu handle
                    hInstance,      // program instance handle
                    NULL) ;        // creation parameters

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC          hdc ;
    PAINTSTRUCT ps ;
    RECT         rect ;

    switch (message)
    {
    case WM_CREATE:
        PlaySound (TEXT ("hellowin.wav"), NULL, SND_FILENAME | SND_ASYNC) ;
        return 0 ;

    case WM_PAINT:
```

```
hdc = BeginPaint (hwnd, &ps) ;

GetClientRect (hwnd, &rect) ;

DrawText (hdc, TEXT ("Hello, Windows 98!"), -1, &rect,
          DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
EndPaint (hwnd, &ps) ;
return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

程式建立一个普通的应用程式视窗，如图 3-1 所示。在视窗显示区域的中央显示「Hello, Windows 98!」。如果安装了音效卡，那么您还可以听到相应的朗读声音。

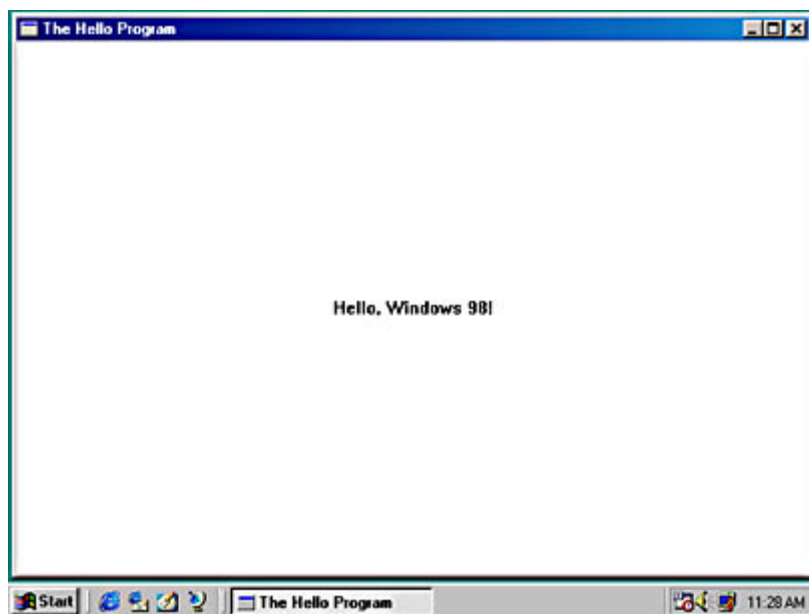


图 3-1 HELLOWIN 视窗

提醒您注意：如果您使用 Microsoft Visual C++ 为此程式建立新专案，那么您得加上连结程式所需的程式库档案。从 **Project** 功能表选择 **Setting** 选项，然後选取 **Link** 页面标签。从 **Category** 清单方块中选择 **General**，然後在 **Object/Library Modules** 文字方块添加 **WINMM.LIB**（**Windows multimedia** —— **Windows 多媒体**）。您这样做是因为 HELLOWIN 将使用多媒体功能呼叫，而内定的专案中又不包括多媒体程式库档案。不然连结程式报告了错误资讯，表明 PlaySound 函式不可用。

HELLOWIN 将存取档案 HELLOWIN.WAV，该档案在本书所附光碟的 HELLOWIN 目录中。执行 HELLOWIN.EXE 时，内定的目录必须是 HELLOWIN。在 Visual C++

中执行此程式时,虽然执行档会产生在 HELLOWIN 的 RELEASE 或 DEBUG 子目录中,但执行程式的目录还是必须在 HELLOWIN 中。

通盘考量

实际上,每一个 Windows 程式码中都包括 HELLOWIN.C 程式的大部分。没人能真正记住此程式的全部写法;通常,Windows 程式写作者在开始写一个新程式时总是会复制一个现有的程式,然後再做相应的修改。您可以按此习惯自由使用本书附带光碟中的程式。

上面提到,HELLOWIN 将在其视窗的中央显示字串。这种说法不是完全正确的。文字实际显示在程式显示区域的中央,它在图 3-1 中是标题列和边界范围内的大片白色区域。这区别对我们来说很重要;显示区域就是程式自由绘图并且向使用者显示输出结果的视窗区域。

如果您认真思考一下,将会发现虽然只有 80 行程式码,这个视窗却令人惊讶地具有许多功能。您可以用滑鼠按住标题列,在萤幕上移动视窗;可以按住大小边框,改变视窗的大小。在视窗大小改变时,程式自动地将「Hello, Windows 98!」字串重新定位在显示区域的中央。您可以按最大化按钮,放大 HELLOWIN 以充满整个萤幕;也可以按最小化按钮,将程式缩小成一个图示。您可以在系统功能表中执行所有选项(就是按下在标题列最左端的小图示);也可以从系统功能表中选择 **Close** 选项,或者单击标题列最右端的关闭按钮,或者双击标题列最左端的图示,来关闭视窗以终止程式的执行。

我们将在本章的余下部分对此程式作一详细的检查。当然,我们首先要从整体上看一下。

与前两章中的范例程式一样,HELLOWIN.C 也有一个 WinMain 函式,但它还有另外一个函式,名为 WndProc。这就是视窗讯息处理程式。注意,在 HELLOWIN.C 中没有呼叫 WndProc 的程式码。当然,在 WinMain 中有对 WndProc 的参考,而这就是该函式要在程式开头附近宣告的原因。

Windows 函式呼叫

HELLOWIN 至少呼叫了 18 个 Windows 函式。下面以它们在 HELLOWIN 中出现的次序列出这些函式以及各自的简明描述:

LoadIcon	载入图示供程式使用。
LoadCursor	载入滑鼠游标供程式使用。
GetStockObject	取得一个图形物件(在这个例子中,是取得绘制视窗背景的画刷物件)。

RegisterClass	为程式视窗注册视窗类别。
MessageBox	显示讯息方块。
CreateWindow	根据视窗类别建立一个视窗。
ShowWindow	在萤幕上显示视窗。
UpdateWindow	指示视窗自我更新。
GetMessage	从讯息伫列中取得讯息。
TranslateMessage	转译某些键盘讯息。
DispatchMessage	将讯息发送给视窗讯息处理程式。
PlaySound	播放一个音效档案。
BeginPaint	开始绘制视窗。
GetClientRect	取得视窗显示区域的大小。
DrawText	显示字串。
EndPaint	结束绘制视窗。
PostQuitMessage	在讯息伫列中插入一个「退出程式」讯息。
DefWindowProc	执行内定的讯息处理。

这些函式均在 Platform SDK 文件中说明，并在不同的表头档案中宣告，其中绝大多数宣告在 WINUSER.H 中。

大写字母识别字

读者可能注意到，HELLOWIN.C 中有几个大写的识别字，这些识别字是在 Windows 表头档案中定义的。有些识别字含有两个字母或者三个字母的字首，这些字首後头接著一个底线：

CS_HREDRAW	DT_VCENTER	SND_FILENAME
CS_VREDRAW	IDC_ARROW	WM_CREATE
CW_USEDEFAULT	IDI_APPLICATION	WM_DESTROY
DT_CENTER	MB_ICONERROR	WM_PAINT
DT_SINGLELINE	SND_ASYNC	WS_OVERLAPPEDWINDOW

这些是简单的数值常数。字首指示该常数所属的类别，如表 3-1 所示。

表 3-1

字首	类别
CS	视窗类别样式

CW	建立视窗
DT	绘制文字
IDI	图示 ID
IDC	游标 ID
MB	讯息方块
SND	声音
WM	视窗讯息
WS	视窗样式

奉劝程式写作者不要费力气去记忆 Windows 程式设计中的数值常数。实际上，Windows 中使用的每个数值常数在表头档案中均有相应的识别字定义。

新的资料型态

HELLOWIN.C 中的其他识别字是新的资料型态，也在 Windows 表头档案中使用 typedef 叙述或者#define 叙述加以定义了。最初是为了便於将 Windows 程式从原来的 16 位元系统上移植到未来的使用 32 位元(或者其他)技术的作业系统上。这种作法并不如当时每个人想像的那样顺利，但是这种概念基本上是正确的。

有时这些新的资料型态只是为了方便缩写。例如，用於 WndProc 的第二个参数的 UINT 资料型态只是一个 unsigned int（无正负号整数），在 Windows 98 中，这是一个 32 位元的值。用於 WinMain 的第三个参数的 PSTR 资料型态是指向一个字串的指标，即是一个 char *。

其他资料型态的含义不太明显。例如，WndProc 的第三和第四个参数分别被定义为 WPARAM 和 LPARAM，这些名字的来源有点历史背景：当 Windows 还是 16 位元系统时，WndProc 的第三个参数被定义为一个 WORD，这是一个 16 位元的 **无正负号短**（unsigned short）整数，而第四个参数被定义为一个 LONG，这是一个 32 位元有正负号长整数，从而导致了文字「PARAM」前面加上了前置字首「W」和「L」。当然，在 32 位元的 Windows 中，WPARAM 被定义为一个 UINT，而 LPARAM 被定义为一个 LONG（这就是 C 中的 long 整数型态），因此视窗讯息处理程式的这两个参数都是 32 位元的值。这也许有点奇怪，因为 WORD 资料型态在 Windows98 中仍然被定义为一种 16 位元的 **无正负号** 整数，因此「PARAM」前的「W」就有点误用了。

WndProc 函式传回一个型态为 LRESULT 的值，该值简单地被定义为一个 LONG。WinMain 函式被指定了一个 WINAPI 型态（在表头档案中定义的所有

Windows 函式都被指定这种型态)，而 WndProc 函式被指定一个 CALLBACK 型态。这两个识别字都被定义为_stdcall，表示在 Windows 本身和使用者的應用程式之间发生的函式呼叫的呼叫参数传递方式。

HELLOWIN 还使用了 Windows 表头档案中定义的四种类结构（我们将在本章稍後加以讨论）。这些资料结构如表 3-2 所示。

表 3-2

结构	含义
MSG	讯息结构
WNDCLASS	视窗类别结构
PAINTSTRUCT	绘图结构
RECT	矩形结构

前面两个资料结构在 WinMain 中使用，分别定义了两个名为 msg 和 wndclass 的结构，後面两个资料结构在 WndProc 中使用，分别定义了 ps 和 rect 结构。

代号简介

最後，还有三个大写识别字（见表 3-3），用於不同型态的「代号」：

表 3-3

识别字	含义
HINSTANCE	执行实体（程式自身）代号
HWND	视窗代号
HDC	装置内容代号

代号在 Windows 中使用非常频繁。在本章结束之前，我们将遇到 HICON（图示代号）、HCURSOR（滑鼠游标代号）和 HBRUSH（画刷代号）。

代号是一个（通常为 32 位元的）整数，它代表一个物件。Windows 中的代号类似传统 C 或者 MS-DOS 程式设计中使用的档案代号。程式几乎总是通过呼叫 Windows 函式取得代号。程式在其他 Windows 函式中使用这个代号，以使用它代表的物件。代号的实际值对程式来说是无关紧要的。但是，向您的程式提供代号的 Windows 模组知道如何利用它来使用相对应的物件。

匈牙利表示法

读者可能注意到，HELLOWIN.C 中有一些变数的名字显得很古怪。如 szCmdLine，它是传递给 WinMain 的参数。

许多 Windows 程式写作者使用一种叫做「匈牙利表示法」的变数命名通则。这是为了纪念传奇性的 Microsoft 程式写作者 Charles Simonyi。非常简单，变数名以一个或者多个小写字母开始，这些字母表示变数的资料型态。例如，szCmdLine 中的 sz 代表「以 0 结尾的字串」。在 hInstance 和 hPrevInstance 中的 h 字首表示「代号」；在 iCmdShow 中的 i 字首表示「整数」。WndProc 的後两个参数也使用匈牙利表示法。正如我在前面已经解释过的，尽管 wParam 应该更适当地被命名为 uiParam（代表「无正负号整数」），但是因为这两个参数是使用资料型态 WPARAM 和 LPARAM 定义的，因此保留它们传统的名字。

在命名结构变数时，可以用结构名（或者结构名的一种缩写）的小写作为变数名的字首，或者用作整个变数名。例如，在 HELLOWIN. C 的 WinMain 函式中，msg 变数是 MSG 型态的结构；wndclass 是 WNDCLASSEX 型态的一个结构。在 WndPmc 函式中，ps 是一个 PAINTSTRUCT 结构，rect 是一个 RECT 结构。

匈牙利表示法能够帮助程式写作者及早发现并避免程式中的错误。由於变数名既描述了变数的作用，又描述了其资料型态，就比较容易避免产生资料型态不合的错误。

表 3-4 列出了在本书中经常用到的变数字首。

表 3-4

字首	资料型态
c	char 或 WCHAR 或 TCHAR
by	BYTE （无正负号字元）
n	short
i	int
x, y	int 分别用作 x 座标和 y 座标
cx, cy	int 分别用作 x 长度和 y 长度；C 代表「计数器」
b 或 f	BOOL (int)；f 代表「旗标」
w	WORD （无正负号短整数）
l	LONG （长整数）
dw	DWORD （无正负号长整数）
fn	function（函式）
s	string（字串）
sz	以位元组值 0 结尾的字串
h	代号
p	指标

注册视窗类别

视窗依照某一视窗类别建立，视窗类别用以标识处理视窗讯息的视窗讯息处理程式。

不同视窗可以依照同一种视窗类别建立。例如，Windows 中的所有按钮视窗——包括按键、核取方块，以及单选按钮——都是依据同一种视窗类别建立的。视窗类别定义了视窗讯息处理程式和依据此类别建立的视窗的其他特徵。在建立视窗时，要定义一些该视窗所独有的特徵。

在为程式建立视窗之前，必须首先呼叫 RegisterClass 注册一个视窗类别。该函式只需要一个参数，即一个指向型态为 WNDCLASS 的结构指标。此结构包括两个指向字串的栏位，因此结构在 WINUSER.H 表头档案中定义了两种不同的方式，第一个是 ASCII 版的 WNDCLASSA：

```
typedef struct tagWNDCLASSA
{
    UINT            style ;
    WNDPROC         lpfnWndProc ;
    int             cbClsExtra ;
    int             cbWndExtra ;
    HINSTANCE       hInstance ;
    HICON           hIcon ;
    HCURSOR         hCursor ;
    HBRUSH          hbrBackground ;
    LPCSTR          lpszMenuName ;
    LPCSTR          lpszClassName ;
}
WNDCLASSA, * PWNDCLASSA, NEAR * NPWNDCLASSA, FAR * LPWNDCLASSA ;
```

在这里提示一下资料型态和匈牙利表示法：其中的 lpfn 字首代表「指向函式的长指标」。（在 Win32 API 中，长指标和短指标（或者近程指标）没有区别。这只是 16 位元 Windows 的遗物。）cb 字首代表「位元组数」而且通常作为一个常数来表示一个位元组的大小。h 字首是一个代号，而 hbr 字首代表「一个画刷的代号」。lpsz 字首代表「指向以 0 结尾字串的指标」。

Unicode 版的结构定义如下：

```
typedef struct tagWNDCLASSW
{
    UINT            style ;
    WNDPROC         lpfnWndProc ;
    int             cbClsExtra ;
    int             cbWndExtra ;
    HINSTANCE       hInstance ;
    HICON           hIcon ;
    HCURSOR         hCursor ;
```

```

        HBRUSH      hbrBackground ;
        LPCWSTR     lpszMenuName ;
        LPCWSTR     lpszClassName ;
    }
    WNDCLASSW, * PWNDCLASSW, NEAR * NPWNDCLASSW, FAR * LPWNDCLASSW ;

```

与前者唯一的区别在於最後两个栏位定义为指向宽字符串常数，而不是指向 ASCII 字符串常数。

WINUSER.H 定义了 WNDCLASSA 和 WNDCLASSW 结构（以及指向结构的指标）以後，表头档案依据对 UNICODE 识别字的解释，定义了 WNDCLASS 和指向 WNDCLASS 的指标（包括一些向後相容的程式码）：

```

#ifdef UNICODE
typedef      WNDCLASSW      WNDCLASS ;
typedef      PWNDCLASSW     PWNDCLASS ;
typedef      NPWNDCLASSW    NPWNDCLASS ;
typedef      LPWNDCLASSW    LPWNDCLASS ;
#else
typedef      WNDCLASSA      WNDCLASS ;
typedef      PWNDCLASSA     PWNDCLASS ;
typedef      NPWNDCLASSA    NPWNDCLASS ;
typedef      LPWNDCLASSA    LPWNDCLASS ;
#endif

```

本书後面列出结构时，将只列出功用相同的结构定义，对 WNDCLASS 就像这样：

```

typedef struct
{
    UINT          style ;
    WNDPROC        lpfnWndProc ;
    int            cbClsExtra ;
    int            cbWndExtra ;
    HINSTANCE      hInstance ;
    HICON          hIcon ;
    HCURSOR        hCursor ;
    HBRUSH         hbrBackground ;
    LPCTSTR        lpszMenuName ;
    LPCTSTR        lpszClassName ;
}
WNDCLASS, * PWNDCLASS ;

```

我也不再著重说明指标的定义。一个程式写作者的程式不应该因为使用以 LP 或 NP 为字首的不同指标型态而被搅乱。

在 WinMain 中为 WNDCLASS 定义一个结构，通常像这样：

```
WNDCLASS wndclass ;
```

然後，你就可以初始化该结构的 10 个栏位，并呼叫 RegisterClass。

在 WNDCLASS 结构中最重要的是第二个和最後一个，第二个栏位

(lpfnWndProc) 是依据这个类别来建立的所有视窗所使用的视窗讯息处理程式的位址。在 HELLOWIN.C 中, 这个是 WndProc 函式。最後一个栏位是视窗类别的文字名称。程式写作者可以随意定义其名称。在只建立一个视窗的程式中, 视窗类别名称通常设定为程式名称。

其他栏位依照下面的方法描述了视窗类别的一些特徵。让我们依次看看 WNDCLASS 结构中的每个栏位。

叙述

```
wndclass.style = CS_HREDRAW | CS_VREDRAW ;
```

使用 C 的位元「或」运算符结合了两个「视窗类别样式」识别字。在表头档案 WINUSER.H 中, 已定义了一整组以 CS 为字首的识别字:

```
#define CS_VREDRAW          0x0001
#define CS_HREDRAW          0x0002
#define CS_KEYCVTWINDOW    0x0004
#define CS_DBLCLKS          0x0008
#define CS_OWNDC             0x0020
#define CS_CLASSDC          0x0040
#define CS_PARENTDC         0x0080
#define CS_NOKEYCVT         0x0100
#define CS_NOCLOSE          0x0200
#define CS_SAVEBITS         0x0800
#define CS_BYTEALIGNCLIENT  0x1000
#define CS_BYTEALIGNWINDOW  0x2000
#define CS_GLOBALCLASS      0x4000
#define CS_IME               0x00010000
```

由於每个识别字都可以在一个复合值中设置一个位元的值, 所以按这种方式定义的识别字通常称为「位元旗标」。通常我们只使用少数的视窗类别样式。HELLOWIN 中用到的这两个识别字表示, 所有依据此类别建立的视窗, 每当视窗的水平方向大小 (CS_HREDRAW) 或者垂直方向大小 (CS_VREDRAW) 改变之後, 视窗要完全重画。改变 HELLOWIN 的视窗大小, 可以看到字串仍然显示在视窗的中央, 这两个识别字确保了这一点。不久我们就将看到视窗讯息处理程式是如何得知这种视窗大小的变化的。

WNDCLASS 结构的第二个栏位由以下叙述进行初始化:

```
wndclass.lpfnWndProc = WndProc ;
```

这条叙述将这个视窗类别的视窗讯息处理程式设定为 WndProc, 即 HELLOWIN.C 中的第二个函式。这个过程将处理依据这个视窗类别建立的所有视窗的全部讯息。在 C 语言中, 像这样在结构中使用函式名时, 真正提供的是指向函式的指标。

下面两个栏位用於在视窗类别结构和 Windows 内部保存的视窗结构中预留一些额外空间:

```
wndclass.cbClsExtra = 0 ;
wndclass.cbWndExtra = 0 ;
```

程式可以根据需要来使用预留的空间。HELLOWIN 没有使用它们，所以设定值为 0。否则，和匈牙利表示法所指示的一样，这个栏位将被当成「预留的位元组数」。（在第七章的程式 CHECKER3 将使用 cbWndExtra 栏位。）

下一个栏位就是程式的执行实体代号（它也是 WinMain 的参数之一）：

```
wndclass.hInstance = hInstance ;
```

叙述

```
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
```

为所有依据这个视窗类别建立的视窗设置一个图示。图示是一个小的点阵图图像，它对使用者代表程式，将出现在 Windows 工作列中和视窗的标题列的左端。在本书的後面，您将学习如何为您的 Windows 程式自订图示。现在，为了方便起见，我们将使用预先定义的图示。

要取得预先定义图示的代号，可以将第一个参数设定为 NULL 来呼叫 LoadIcon。在载入程式写作者自订的图示时（图示应该存放在磁片上的.EXE 程式档案中），这个参数应该被设定为程式的执行实体代号 hInstance。第二个参数代表图示。对于预先定义图示，此参数是以 IDI 开始的识别字（「ID 代表图示」），识别字在 WINUSER.H 中定义。IDI_APPLICATION 图示是一个简单的视窗小图形。LoadIcon 函式传回该图示的代号。我们并不关心这个代号的实际值，它只用於设置 hIcon 栏位元的值。该栏位在 WNDCLASS 结构中定义为 HICON 型态，此型态名的含义为「handle to an icon（图示代号）」。

叙述

```
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
```

与前一条叙述非常相似。LoadCursor 函式载入一个预先定义的滑鼠游标（命名为 IDC_ARROW），并传回该游标的代号。该代号被设定给 WNDCLASS 结构的 hCursor 栏位。当滑鼠游标在依据这个类别建立的视窗的显示区域上出现时，它变成一个小箭头。

下一个栏位指定依据这个类别建立的视窗背景颜色。hbrBackground 栏位名称中的 hbr 字首代表「handle to a brush（画刷代号）」。画刷是个绘图词汇，指用来填充一个区域的著色样式。Windows 有几个标准画刷，也称为「备用 (stock)」画刷。这里所示的 GetStockObject 呼叫将传回一个白色画刷的代号：

```
wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
```

这意味著视窗显示区域的背景完全为白色，这是一种极其普遍的做法。

下一个栏位指定视窗类别功能表。HELLOWIN 没有应用程式功能表，所以该栏位被设定为 NULL：

```
wndclass.lpszMenuName = NULL ;
```

最後，必须给出一个类别名称。對於小程序，类别名称可以与程式名相同，即存放在 szAppName 变数中的「HelloWin」字串。

```
wndclass.lpszClassName = szAppName ;
```

至於该字串由 ASCII 字元组成或由 Unicode 字元组成，取决於是否定义了 UNICODE 识别字。

在初始化该结构的 10 个栏位後，HELLOWIN 呼叫 RegisterClass 来注册这个视窗类别。该函式只有一个参数，即指向 WNDCLASS 结构的指标。实际上，RegisterClassA 函式将获得一个指向 WNDCLASSA 结构的指标，而 RegisterClassW 函式将获得一个指向 WNDCLASSW 结构的指标。程式要使用哪个函式来注册视窗类别，取决於发送给视窗的讯息包含 ASCII 文字还是 Unicode 文字。

现在有一个问题：如果用定义的 UNICODE 识别字编译了程式，程式将呼叫 RegisterClassW。该程式可以在 Microsoft Windows NT 中执行良好。但如果此程式在 Windows 98 上执行，RegisterClassW 函式并未真地被执行到。函式有一个进入点，但函式呼叫後只传回 0，表明错误。對於在 Windows 98 下执行的 Unicode 程式来说，这是一个通知使用者有问题并终止执行的好机会。这是本书中多数程式处理 RegisterClass 函式呼叫的方法：

```
if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;
    return 0 ;
}
```

由於 MessageBoxW 是可在 Windows 98 环境下执行的几个 Unicode 函式之一，所以其执行正常。

当然，这段程式假定 RegisterClass 不会因为其他原因而呼叫失败，诸如 WNDCLASS 结构中 lpfnWndProc 栏位被设定成 NULL 之类的错误。GetLastError 函式会帮助您确定在这样的情况下产生错误的原因。GetLastError 是 Windows 中常用的函式，它可以在函式呼叫失败时获得更多错误资讯。不同函式的文件将指出您是否能够用 GetLastError 来获得这些资讯。在 Windows 98 中呼叫 RegisterClassW 时，GetLastError 将传回 120。在 WINERROR.H 中您可以看到，值 120 与识别字 ERROR_CALL_NOT_IMPLEMENTED 相等。您也可以在/Platform SDK/Windows Base Services/Debugging and Error Handling/Error Codes/System Errors - Numerical Order 查看错误。

一些 Windows 程式写作者喜欢检查所有可能发生错误的函式呼叫的传回值。这么做确实有点道理，相信您也非常习惯在配置记忆体後检查错误。而许多

Windows 函式需要配置记忆体。例如，RegisterClass 需要配置记忆体，以保存视窗类别的资讯。如此一来，您就应该要检查这个函式的执行结果。另一方面说来，如果由於 RegisterClass 不能得到所需要的记忆体，它会宣告呼叫失败，而 Windows 大概也快当掉了。

在本书的范例程式中，我做了最少的错误检查。这不是因为我认为错误检查不是一个好方法，而是因为这会让我们在程式举例中分心。

最後，一个老经验是：在一些 Windows 范例程式中，您可能在 WinMain 中看到以下程式码：

```
if (!hPrevInstance)
{
    wndclass.cbStyle = CS_HREDRAW | CS_VREDRAW ;

    初始化其他 wndclass

    RegisterClass (&wndclass) ;
}
```

这是出於「旧习难改」的原因。在 16 位元的 Windows 中，如果您启动正在执行的程式的一个新执行实体，WinMain 的 hPrevInstance 参数将是前一个执行实体的执行实体代号。为节省记忆体，两个或多个执行实体就可能会共用相同的视窗类别。这样，视窗类别就只在 hPrevInstance 是 NULL 的时候才注册，这表明程式没有其他执行实体。

在 32 位元的 Windows 中，hPrevInstance 总是 NULL。此程式码会正常执行，而实际上也没必要检查 hPrevInstance。

建立视窗

视窗类别定义了视窗的一般特徵，因此可以使用同一视窗类别建立许多不同的视窗。实际呼叫 CreateWindow 建立视窗时，可能指定有关视窗的更详细的资讯。

Windows 程式设计新手有时会混淆视窗类别和视窗之间的区别，以及为什么一个视窗的所有特徵不能被一次设定好。实际上，以这种方式分开这些样式资讯是非常方便的。例如，所有的按钮视窗都可以依据同样的视窗类别来建立，与这个视窗类别相关的视窗讯息处理程式位於 Windows 内部。由视窗类别来负责处理按钮的键盘和滑鼠输入，并定义按钮在萤幕上的外观形象。从这一点看来，所有的按钮都是以同样的方式工作的。但是并非所有的按钮都是一样的。它们可以有不同的大小，不同的萤幕位置，以及不同的字串。後面的这样一些特徵是视窗定义的一部分，而不是视窗类别定义的。

传递给 RegisterClass 函式的资讯会在一个资料结构中设定好，而传递给 CreateWindow 函式的资讯会在函式单独的参数中设定好。下面是 HELLOWIN.C 中的 CreateWindows 呼叫，每一个栏位都做了完整的说明：

```
hwnd = CreateWindow (szAppName,      // window class name
    TEXT (        "The Hello Program"), // window caption
    WS_OVERLAPPEDWINDOW,      // window style
    CW_USEDEFAULT,            // initial x position
    CW_USEDEFAULT,            // initial y position
    CW_USEDEFAULT,            // initial x size
    CW_USEDEFAULT,            // initial y size
    NULL,                    // parent window handle
    NULL,                    // window menu handle
    hInstance,                // program instance handle
    NULL) ;                  // creation parameters
```

在这里，我不想提实际上有 CreateWindowA 函式和 CreateWindowW 函式，两个函式分别将前两个参数当成 ASCII 或者 Unicode 字串来处理。

标记为「window class name」的参数是 szAppName，它含有字串「HelloWin」——这是程式注册的视窗类别名称。这就是我们建立的视窗联结视窗类别的方式。

此程式建立的视窗是一个普通的重叠式视窗。它含有一个标题列，标题列左边有一个系统功能表按钮，标题列右边有缩小、放大和关闭图示，四周还有一个表示视窗大小的边框。这是标准样式的视窗，名为 WS_OVERLAPPEDWINDOW，出现在 CreateWindow 的「视窗样式」参数中。如果看一下 WINUSER.H，您将会发现此样式是几种位元旗标的组合：

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED | \
    WS_CAPTION                | \
    WS_SYSMENU                | \
    WS_THICKFRAME              | \
    WS_MINIMIZEBOX            | \
    WS_MAXIMIZEBOX)
```

「视窗标题」是显示在标题列中的文字。

注释著「initial x position」和「initial y position」的参数指定了视窗左上角相对於萤幕左上角的初始位置。由於这些参数使用 CW_USEDEFAULT 识别字，指示 Windows 使用重叠视窗的内定位置。（CW_USEDEFAULT 定义为 0x80000000。）内定情况下，Windows 依次对新建立的视窗定位，使各视窗左上角的垂直和水平距离在萤幕上按一定的大小递增。与此类似，注释著「initial x size」和「initial y size」的参数分别指定视窗的宽度和高度。同样使用了 CW_USEDEFAULT 识别字，表明希望 Windows 使用内定尺寸。

在建立一个「最上层」视窗，如應用程式视窗时，注释为「父视窗代号」

的参数设定为 NULL。通常，如果视窗之间存在有父子关系，则子视窗总是出现在父视窗的上面。應用程式视窗出现在桌面视窗的上面，但不必为呼叫 CreateWindow 而找出桌面视窗的代号。

因为视窗没有功能表，所以「视窗功能表代号」也设定为 NULL。「程式执行实体代号」设定为执行实体代号，它是作为 WinMain 的参数传递给这个程式的。最後，「建立参数」指标设定为 NULL，可以用这个参数存取稍後程式中可能引用到的资料。

CreateWindow 传回被建立的视窗的代号，该代号存放在变数 hwnd 中，後者被定义为 HWND 型态（「视窗代号型态」）。Windows 中的每个视窗都有一个代号，程式用代号来使用视窗。许多 Windows 函式需要使用 hwnd 作为参数，这样，Windows 才能知道函式是针对哪个视窗的。如果一个程式建立了许多视窗，则每个视窗均有一个代号。视窗代号是 Windows 程式所处理最重要的代号之一。

显示视窗

在 CreateWindow 呼叫传回之後，Windows 内部已经建立了这个视窗。这就是说，Windows 已经配置了一块记忆体，用来保存在 CreateWindow 呼叫中指定视窗的全部资讯跟一些其他资讯，而 Windows 稍後就是依据视窗代号找到这些资讯的。

然而，光是这样子，视窗并不会出现在视讯显示器上。您还需要两个函式呼叫，一个是：

```
ShowWindow (hwnd, iCmdShow) ;
```

第一个参数是刚刚用 CreateWindow 建立的视窗代号。第二个参数是作为参数传给 WinMain 的 iCmdShow。它确定最初如何在萤幕上显示视窗，是一般大小、最小化还是最大化。在开始功能表中安装程式时，使用者可能做出最佳选择。如果视窗按一般大小显示，那么 WinMain 接收到後传递给 ShowWindow 的就是 SW_SHOWNORMAL；如果视窗是最大化显示的，则为 SW_SHOWMAXIMIZED。而如果视窗只显示在工作列上，则是 SW_SHOWMINNOACTIVE。

ShowWindow 函式在显示器上显示视窗。如果 ShowWindow 的第二个参数是 SW_SHOWNORMAL，则视窗的显示区域就会被视窗类别中定义的背景画刷所覆盖。

函式呼叫

```
UpdateWindow (hwnd) ;
```

会重画显示区域。它经由发送给视窗讯息处理程式（即 HELLOWIN.C 中的 WndProc 函式）一个 WM_PAINT 讯息做到这一点。後面，我们将说明 WndProc 如何处理这个讯息。

讯息回圈

呼叫 UpdateWindow 之後，视窗就出现在视讯显示器上。程式现在必须准备读入使用者用键盘和滑鼠输入的资料。Windows 为当前执行的每个 Windows 程式维护一个「讯息伫列」。在发生输入事件之後，Windows 将事件转换为一个「讯息」并将讯息放入程式的讯息伫列中。

程式通过执行一块称之为「讯息回圈」的程式码从讯息伫列中取出讯息：

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
```

msg 变数是型态为 MSG 的结构，型态 MSG 在 WINUSER.H 中定义如下：

```
typedef struct tagMSG
{
    HWND    hwnd ;
    UINT    message ;
    WPARAM   wParam ;
    LPARAM   lParam ;
    DWORD    time ;
    POINT    pt ;
}
MSG, * PMSG ;
```

POINT 资料型态也是一个结构，它在 WINDEF.H 中定义如下：

```
typedef struct tagPOINT
{
    LONG    x ;
    LONG    y ;
}
POINT, * PPOINT;
```

讯息回圈以 GetMessage 呼叫开始，它从讯息伫列中取出一个讯息：

```
GetMessage (&msg, NULL, 0, 0)
```

这一呼叫传给 Windows 一个指标，指向名为 msg 的 MSG 结构。第二、第三和第四个参数设定为 NULL 或者 0，表示程式接收它自己建立的所有视窗的所有讯息。Windows 用从讯息伫列中取出的下一个讯息来填充讯息结构的各个栏位，结构的各个栏位包括：

hwnd 接收讯息的视窗代号。在 HELLOWIN 程式中，这一参数与 CreateWindow 传回的 hwnd 值相同，因为这是该程式拥有的唯一视窗。

message 讯息识别字。这是一个数值，用以标识讯息。对於每个讯息，均有一个对应的识别字，这些识别字定义於 Windows 表头档案（其中大多数在 WINUSER.H 中），以字首 WM（「window message」，视窗讯息）开头。例如，

使用者将滑鼠游标放在 HELLOWIN 显示区域之内，并按下滑鼠左按钮，Windows 就在讯息佇列中放入一个讯息，该讯息的 message 栏位等於 WM_LBUTTONDOWN。这是一个常数，其值为 0x0201。

wParam	一个 32 位元的「message parameter (讯息参数)」，其含义和数值根据讯息的不同而不同。
lParam	一个 32 位元的讯息参数，其值与讯息有关。
time	讯息放入讯息佇列中的时间。
pt	讯息放入讯息佇列时的滑鼠座标。

只要从讯息佇列中取出讯息的 message 栏位不为 WM_QUIT(其值为 0x0012)，GetMessage 就传回一个非零值。WM_QUIT 讯息将导致 GetMessage 传回 0。

叙述

```
TranslateMessage (&msg) ;
```

将 msg 结构传给 Windows，进行一些键盘转换。（关于这一点，我们将在第六章中深入讨论。）

叙述

```
DispatchMessage (&msg) ;
```

又将 msg 结构回传给 Windows。然後，Windows 将该讯息发送给适当的视窗讯息处理程式，让它进行处理。这也就是说，Windows 将呼叫视窗讯息处理程式。
在 HELLOWIN 中，这个视窗讯息处理程式就是 WndProc 函式。处理完讯息之後，
WndProc 传回到 Windows。此时，Windows 还停留在 DispatchMessage 呼叫中。
在结束 DispatchMessage 呼叫的处理之後，Windows 回到 HELLOWIN，并且接著
从下一个 GetMessage 呼叫开始讯息回圈。

视窗讯息处理程式

以上我们所讨论的都是必要的负担：注册视窗类别，建立视窗，然後在萤幕上显示视窗，程式进入讯息回圈，然後不断从讯息佇列中取出讯息来处理。

实际的动作发生在视窗讯息处理程式中。视窗讯息处理程式确定了在视窗的显示区域中显示些什么以及视窗怎样回应使用者输入。

在 HELLOWIN 中，视窗讯息处理程式是命名为 WndProc 的函式。视窗讯息处理程式可任意命名（只要求不和其他名字发生冲突）。一个 Windows 程式可以包含多个视窗讯息处理程式。一个视窗讯息处理程式总是与呼叫 RegisterClass 注册的特定视窗类别相关联。CreateWindow 函式根据特定视窗类别建立一个视窗。但依据一个视窗类别，可以建立多个视窗。

视窗讯息处理程式总是定义为如下形式：

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,
```

LPARAM lParam)

注意，视窗讯息处理程式的四个参数与 MSG 结构的前四个栏位是相同的。第一个参数 hwnd 是接收讯息的视窗的代号，它与 CreateWindow 函式的传回值相同。对于与 HELLOWIN 相似的程式（只建立一个视窗），这个参数是程式所知道的唯一视窗代号。如果程式是依据同一视窗类别（同时也是同一视窗讯息处理程式）建立多个视窗，则 hwnd 标识接收讯息的特定视窗。

第二个参数与 MSG 结构中的 message 栏位相同，它是标识讯息的数值。最后两个参数都是 32 位元的讯息参数，提供关于讯息的更多资讯。这些参数包含每个讯息型态的详细资讯。有时讯息参数是两个存放在一起的 16 位元值，而有时讯息参数又是一个指向字串或资料结构的指标。

程式通常不直接呼叫视窗讯息处理程式，视窗讯息处理程式通常由 Windows 本身呼叫。通过呼叫 SendMessage 函式，程式能够直接呼叫它自己的视窗讯息处理程式。我们将在后面的章节讨论 SendMessage 函式。

处理讯息

视窗讯息处理程式所接受的每个讯息均是使用一个数值来标识的，也就是传给视窗讯息处理程式的 message 参数。Windows 表头档案 WINUSER.H 为每个讯息参数定义以「WM」（视窗讯息）为字首开头的识别字。

一般来说，Windows 程式写作者使用 switch 和 case 结构来确定视窗讯息处理程式接收的是什么讯息，以及如何适当地处理它。视窗讯息处理程式在处理讯息时，必须传回 0。视窗讯息处理程式不予处理的所有讯息应该被传给名为 DefWindowProc 的 Windows 函式。从 DefWindowProc 传回的值必须由视窗讯息处理程式传回。

在 HELLOWIN 中，WndProc 只选择处理三种讯息：WM_CREATE、WM_PAINT 和 WM_DESTROY。视窗讯息处理程式的结构如下：

```
switch (iMsg)
{
case WM_CREATE :
    处理 WM_CREATE 讯息
    return 0 ;

case WM_PAINT :
    处理 WM_PAINT 讯息
    return 0 ;

case WM_DESTROY :
    处理 WM_DESTROY 讯息
    return 0 ;
```

```
}  
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
```

呼叫 DefWindowProc 来为视窗讯息处理程式不予处理的所有讯息提供内定处理，这是很重要的。不然一般动作，如终止程式，将不会正常执行。

播放音效档案

视窗讯息处理程式接收的第一个讯息——也是 WndProc 选择处理的第一个讯息——是 WM_CREATE。当 Windows 在 WinMain 中处理 CreateWindow 函式时，WndProc 接收这个讯息。就是说，在 HELLOWIN 呼叫 CreateWindow 时，Windows 将做一些它必须做的工作。在这些工作中，Windows 呼叫 WndProc，将第一个参数设定为视窗代号，第二个参数设定为 WM_CREATE。WndProc 处理 WM_CREATE 讯息并将控制传回给 Windows。Windows 然後可以从 CreateWindow 呼叫中传回到 HELLOWIN 中，继续在 WinMain 中进行下一步的处理。

通常，视窗讯息处理程式在 WM_CREATE 处理期间进行一次视窗初始化。HELLOWIN 对这个讯息的处理中播放一个名为 HELLOWIN.WAV 的音效档案。它使用简单的 PlaySound 函式来做到这一点。该函式说明在 /Platform SDK/Graphics and Multimedia Services/Multimedia Audio/Waveform Audio 中，而文件在 /Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Functions 中。

PlaySound 的第一个参数是音效档案的名称（它也可能是在 Control Panel 的 Sounds 中定义的一种声音的别名，或者是一个程式资源）。第二个参数只有当音效档案是一种资源时才被使用。第三个参数指定一些选项。在这个例子中，我指定第一个参数是一个档案名，并且非同步地播放声音，即 PlaySound 函式呼叫在音效档案开始播放时立即传回，而不会等待它的完成。在这种方法下，程式能够继续初始化。

WndProc 通过从视窗讯息处理程式中传回 0，结束了整个 WM_CREATE 的处理。

WM_PAINT 讯息

WndProc 处理的第二个讯息为 WM_PAINT。这个讯息在 Windows 程式设计中是很重要的。当视窗显示区域的一部分显示内容或者全部变为「无效」，以致於必须「更新画面」时，将由这个讯息通知程式。

显示区域的显示内容怎么会变得无效呢？在最初建立视窗的时候，整个显示区域都是无效的，因为程式还没有在视窗上画什么东西。第一条 WM_PAINT 讯息（通常发生在 WinMain 中呼叫 UpdateWindow 时）指示视窗讯息处理程式在显

示区域上画一些东西。

在使用者改变 HELLOWIN 视窗的大小後,显示区域的显示内容重新变得无效。读者应该还记得,HELLOWIN 中 wndclass 结构的 style 栏位设定为标志 CS_HREDRAW 和 CS_VREDRAW,这样的格式设定指示 Windows,在视窗大小改变後,就把整个视窗显示内容当成无效。然後,视窗讯息处理程式将收到一条 WM_PAINT 讯息。

当使用者将 HELLOWIN 最小化,然後再次将视窗恢复为以前的大小时,Windows 将不会保存显示区域的内容。在图形环境下,视窗显示区域涉及的资料量很大。因此,Windows 令视窗无效,视窗讯息处理程式接收一条 WM_PAINT 讯息,并自动恢复其视窗的内容。

在移动视窗以致其相互重叠时,Windows 不保存一个视窗中被另一个视窗所遮盖的内容。在这一部分不再被遮盖之後,它就被标志为无效。视窗讯息处理程式接收到一条 WM_PAINT 讯息,以更新视窗的内容。

对 WM_PAINT 的处理几乎总是从一个 BeginPaint 呼叫开始:

```
hdc = BeginPaint (hwnd, &ps) ;
```

而以一个 EndPaint 呼叫结束:

```
EndPaint (hwnd, &ps) ;
```

在这两个呼叫中,第一个参数都是程式的视窗代号,第二个参数是指向型态为 PAINTSTRUCT 的结构指标。PAINTSTRUCT 结构中包含一些视窗讯息处理程式,可以用来更新显示区域的内容。我们将在下一章中讨论该结构的各个栏位。现在我们只在 BeginPaint 和 EndPaint 函式中用到它。

在 BeginPaint 呼叫中,如果显示区域的背景还未被删除,则由 Windows 来删除。它使用注册视窗类别的 WNDCLASS 结构的 hbrBackground 栏位中指定的画刷来删除背景。在 HELLOWIN 中,这是一个白色备用画刷。这意味著,Windows 将通过把视窗背景设定为白色来删除视窗背景。BeginPaint 呼叫令整个显示区域有效,并传回一个「装置内容代号」。装置内容是指实体输出设备(如视讯显示器)及其装置驱动程序。在视窗的显示区域显示文字和图形需要装置内容代号。但是从 BeginPaint 传回的装置内容代号不能在显示区域之外绘图,读者可以试一试。EndPaint 释放装置内容代号,使之不再有效。

如果视窗讯息处理程式不处理 WM_PAINT 讯息(这是很罕见的),它们必须被传送给 DefWindowProc。DefWindowProc 只是依次呼叫 BeginPaint 和 EndPaint,以使显示区域有效。

呼叫完 BeginPaint 之後,WndProc 接著呼叫 GetClientRect:

```
GetClientRect (hwnd, &rect) ;
```

第一个参数是程式视窗的代号。第二个参数是一个指标,指向一个 RECT 型

态的 rectangle 结构。该结构有四个 LONG 栏位，分别为 left、top、right 和 bottom。GetClientRect 将这四个栏位设定为视窗显示区域的尺寸。left 和 top 栏位通常设定为 0，right 和 bottom 栏位设定为显示区域的宽度和高度（图元点数）。

WndProc 除了将该 RECT 结构指标作为 DrawText 的第四个参数传递外，不再对它做其他处理：

```
DrawText ( hdc, TEXT ("Hello, Windows 98!"), -1, &rect,
          DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
```

DrawText 可以输出文字（正如其名字所表明的一样）。由於该函式要输出文字，第一个参数是从 BeginPaint 传回的装置内容代号，第二个参数是要输出的文字，第三个参数是 -1，指示字串是以位元组 0 终结的。

DrawText 最後一个参数是一系列位元旗标，它们均在 WINUSER.H 中定义（虽然由於其显示输出的效果，使得 DrawText 像一个 GDI 函式呼叫，但它确实因为相当高级的画图功能而成为 User 模组的一部分。此函式在 /Platform SDK/Graphics and Multimedia Services/GDI/Fonts and Text 中说明）。旗标指示了文字必须显示在一行上，水平方向和垂直方向都位於第四个参数指定的矩形中央。因此，这个函式呼叫将让字串「Hello, Windows 98!」显示在显示区域的中央。

一旦显示区域变得无效（正如在改变大小时所发生的情况一样），WndProc 就接收到一个新的 WM_PAINT 讯息。WndProc 通过呼叫 GetClientRect 取得变化後的视窗大小，并在新视窗的中央显示文字。

WM_DESTROY 讯息

WM_DESTROY 讯息是另一个重要讯息。这一个讯息指示，Windows 正在根据使用者的指示关闭视窗。该讯息是使用者单击 Close 按钮或者在程式的系统功能表上选择 Close 时发生的（在本章的後面，我们将详细讨论 WM_DESTROY 讯息是如何生效的）。

HELLOWIN 通过呼叫 PostQuitMessage 以标准方式回应 WM_DESTROY 讯息：

```
PostQuitMessage (0) ;
```

该函式在程式的讯息佇列中插入一个 WM_QUIT 讯息。前面提到过，GetMessage 对於除了 WM_QUIT 之外的从讯息佇列中取出的所有讯息都传回非 0 值。而当 GetMessage 得到一个 WM_QUIT 讯息时，它传回 0。这将导致 WinMain 退出讯息回圈，并终止程式。然後程式执行下面的叙述：

```
return msg.wParam ;
```

结构的 wParam 栏位是传递给 PostQuitMessage 函式的值（通常是 0）。然

後 return 叙述将退出 WinMain 并终止程式。

WINDOWS 程式设计的难点

即使有了对 HELLOWIN 的说明,读者对程式的结构和原理可能仍然觉得神秘。在为传统环境编写简单的 C 程式时,整个程式可能包含在 main 函式中。而在 HELLOWIN 中,WinMain 只包含了注册视窗类别,建立视窗,从讯息佇列中取出讯息和发送讯息所必须的程式码。

程式的所有实际动作均在视窗讯息处理程式中发生。在 HELLOWIN 中,这些动作不多,WndProc 只是简单地播放了一个音效档案并在视窗中显示一个字符串。但是在後面的章节中,读者将发现,Windows 程式所作的一切,都是回应发送给视窗讯息处理程式的讯息。这是概念上的主要难点之一,在开始写作 Windows 程式之前,必须先搞清楚。

别呼叫我,我会呼叫您

前面我们提到过,程式写作者已经熟悉了使用作业系统呼叫的做法。例如,C 程式写作者使用 fopen 函式打开档案。fopen 函式最终通过呼叫作业系统来打开档案,这一点问题也没有。

但是 Windows 不同,尽管 Windows 有 1000 个以上的函式可供程式呼叫,但 Windows 也呼叫使用者程式,比如前面定义的视窗讯息处理程式 WndProc。视窗讯息处理程式与视窗类别相关,视窗类别是程式呼叫 RegisterClass 注册的。依据该类别建立的视窗使用这个视窗讯息处理程式来处理视窗的所有讯息。Windows 通过呼叫视窗讯息处理程式对视窗发送讯息。

在第一次建立视窗时,Windows 呼叫 WndProc。在视窗关闭时,Windows 也呼叫 WndProc。视窗改变大小、移动或者变成图示时,从功能表中选择某一项目、挪动卷动列、按下鼠标按钮或者从键盘输入字元时,以及视窗显示区域必须被更新时,Windows 都要呼叫 WndProc。

所有这些 WndProc 呼叫都以讯息的形式进行。在大多数 Windows 程式中,程式的主要部分都用来处理讯息。Windows 可以发送给视窗讯息处理程式的讯息通常都以 WM 开头的名字标识,并且都在 WINUSER.H 表头档案中定义。

实际上,从程式外呼叫程式内的常式这一种做法,在传统的程式设计中并非前所未闻。C 中的 signal 函式可以拦截 Ctrl-C 中断或作业系统的其他中断。为 MS-DOS 编写的老程式中经常有拦截硬体中断的程式码。

但在 Windows 中,这种概念扩展为包括一切事件。视窗中发生的一切都以讯息的形式传给视窗讯息处理程式。然後,视窗讯息处理程式以某种方式回应

这个讯息，或者将讯息传给 DefWindowProc，进行内定处理。

在 HELLOWIN 中，视窗讯息处理程式的 wParam 和 lParam 参数除了作为传递给 DefWindowProc 的参数外，不再有其他用处。这些参数给出了关于讯息的其它资讯，参数的含义与具体讯息相关。

让我们来看一个例子。一旦视窗的显示区域大小发生了改变，Windows 就呼叫视窗的视窗讯息处理程式。视窗讯息处理程式的 hwnd 参数是改变大小的视窗的代号（请记住，一个视窗讯息处理程式能处理依据同一个视窗类别建立的多个视窗的讯息。参数 hwnd 让视窗讯息处理程式知道是哪个视窗在接收讯息）。参数 message 是 WM_SIZE。讯息 WM_SIZE 的参数 wParam 的值是 SIZE_RESTORED、SIZE_MINIMIZED、SIZE_MAXIMIZED、SIZE_MAXSHOW 或 SIZE_MAXHIDE（在 WINUSER.H 表头档案中分别定义为数字 0 到 4）。也就是说，参数 wParam 表明视窗是非最小化还是非最大化，是最小化、最大化，还是隐藏。

lParam 参数包含了新视窗的大小，新宽度和新高度均为 16 位元值，合在一起成为 32 位元的 lParam。WINDEF.H 中提供了帮助程式写作者从 lParam 中取出这两个值的巨集，我们将在下一章说明这个巨集。

有时候，DefWindowProc 处理完讯息后会产生其它的讯息。例如，假设使用者执行 HELLOWIN，并且使用者最终单击了 **Close** 按钮，或者假设用键盘或滑鼠从系统功能表中选择了 **Close**，DefWindowProc 处理这一键盘或者滑鼠输入，在检测到使用者选择了 **Close** 选项之后，它给视窗讯息处理程式发送一条 WM_SYSCOMMAND 讯息。WndProc 将这个讯息传给 DefWindowProc。DefWindowProc 给视窗讯息处理程式发送一条 WM_CLOSE 讯息来回应之。WndProc 再次将它传给 DefWindowProc.DestroyWindow 呼叫 DestroyWindow 来回应这条 WM_CLOSE 讯息。DestroyWindow 导致 Windows 给视窗讯息处理程式发送一条 WM_DESTROY 讯息。WndProc 再呼叫 PostQuitMessage，将一条 WM_QUIT 讯息放入讯息伫列中，以此来回应此讯息。这个讯息导致 WinMain 中的讯息回圈终止，然后程式结束。

伫列化讯息与非伫列化讯息

我们已经谈到过，Windows 给视窗发送讯息，这意味著 Windows 呼叫视窗讯息处理程式。但是，Windows 程式也有一个讯息回圈，它呼叫 GetMessage 从讯息伫列中取出讯息，并且呼叫 DispatchMessage 将讯息发送给视窗讯息处理程式。

那么，Windows 程式是依次等待讯息（类似于普通程式中相同的键盘输入），然后将讯息送到某地方去的吗？或者，它是直接从程式外面接收讯息的吗？实际上，两种情况都存在。

讯息能够被分为「伫列化的」和「非伫列化的」。伫列化的讯息是由 Windows 放入程式讯息伫列中的。在程式的讯息回圈中，重新传回并分配给视窗讯息处理程式。非伫列化的讯息在 Windows 呼叫视窗时直接送给视窗讯息处理程式。也就是说，伫列化的讯息被「发送」给讯息伫列，而非伫列化的讯息则「发送」给视窗讯息处理程式。任何情况下，视窗讯息处理程式都将获得视窗所有的讯息—包括伫列化的和非伫列化的。视窗讯息处理程式是视窗的「讯息中心」。

伫列化讯息基本上是使用户输入的结果，以击键（如 WM_KEYDOWN 和 WM_KEYUP 讯息）、击键产生的字元（WM_CHAR）、滑鼠移动（WM_MOUSEMOVE）和滑鼠按钮（WM_LBUTTONDOWN）的形式给出。伫列化讯息还包含时钟讯息（WM_TIMER）、更新讯息（WM_PAINT）和退出讯息（WM_QUIT）。

非伫列化讯息则是其他讯息。在许多情况下，非伫列化讯息来自呼叫特定的 Windows 函式。例如，当 WinMain 呼叫 CreateWindow 时，Windows 将建立视窗并在处理中给视窗讯息处理程式发送一个 WM_CREATE 讯息。当 WinMain 呼叫 ShowWindow 时，Windows 将给视窗讯息处理程式发送 WM_SIZE 和 WM_SHOWWINDOW 讯息。当 WinMain 呼叫 UpdateWindow 时，Windows 将给视窗讯息处理程式发送 WM_PAINT 讯息。键盘或滑鼠输入时发出的伫列化讯息信号，也能在非伫列化讯息中出现。例如，用键盘或滑鼠选择了一个功能表项时，键盘或滑鼠讯息就是伫列化的，而说明功能表项已选中的 WM_COMMAND 讯息则可能就是非伫列化的。

这一过程显然很复杂，但幸运的是，其中的大部分是由 Windows 解决的，不关我们的程式的事。从视窗讯息处理程式的角度来看，这些讯息是以一种有序的、同步的方式进出的。视窗讯息处理程式可以处理它们，也可以不处理。

当我说讯息是以一种有序的同步的方式进出时，我是说首先讯息与硬体的中断不同。在一个视窗讯息处理程式中处理讯息时，程式不会被其他讯息突然中断。

虽然 Windows 程式可以多执行绪执行，但每个执行绪的讯息伫列只为视窗讯息处理程式在该执行绪中执行的视窗处理讯息。换句话说，讯息回圈和视窗讯息处理程式不是并发执行的。当一个讯息回圈从其讯息伫列中接收一个讯息，然後呼叫 DispatchMessage 将讯息发送给视窗讯息处理程式时，直到视窗讯息处理程式将控制传回给 Windows，DispatchMessage 才能结束执行。

当然，视窗讯息处理程式能呼叫给视窗讯息处理程式发送另一个讯息的函式。这时，视窗讯息处理程式必须在函式呼叫传回之前完成对第二个讯息的处理。那时视窗讯息处理程式将处理最初的讯息。例如，当视窗程序呼叫 UpdateWindow 时，Windows 将呼叫视窗讯息处理程式来处理 WM_PAINT 讯息。视窗讯息处理程式处理 WM_PAINT 讯息结束以後，UpdateWindow 呼叫将把控制传回

给视窗讯息处理程式。

这也就是说视窗讯息处理程式必须是可重入。在大多数情况下，这不会带来问题，但是程式写作者应该意识到这一点。例如，假设您在视窗讯息处理程式中处理一个讯息时设置了一个静态变数，然後呼叫了一个 Windows 函式。在这个函式传回时，您还能保证那个变数的值还是原来那个吗？难说——很可能您呼叫的 Windows 函式产生了另外一个讯息，并且视窗讯息处理程式在处理这个讯息时改变了该变数的值。这也是在编译 Windows 程式时，有些编译最佳化选项必须关闭的原因之一。

在许多情况下，视窗讯息处理程式必须保存它从讯息中取得的资讯，并在处理另一个讯息时使用这些资讯。这些资讯可以储存在视窗的静态 (static) 变数或整体变数中。

当然，读者将在下面几章对此有一个更清楚的了解，因为视窗讯息处理程式将处理更多的讯息。

行动迅速

Windows 98 和 Windows NT 都是优先权式的多工环境。这意味著当一个程式在进行一项长时间工作时，Windows 可以允许使用者将控制切换到另一个程式中。这是一件好事，也是现在的 Windows 优越於以前 16 位元 Windows 的地方。

然而，由於 Windows 设计的方式，这种优先权式多工并不总是以您希望的样子工作。例如，假设您的程式花费一分钟左右来处理某一个讯息。是的，使用者可以将控制切换到另一个程式，但是却无法对您的程式进行任何动作。使用者无法移动您的程式视窗、缩放它、最小化、关闭它、什么都不能做。这是因为您的视窗讯息处理程式正忙於进行一项长时间的作业。表面上并不是视窗讯息处理程式在执行它自己的移动和缩放操作，但实际上确实是它在做。这就是 DefWindowProc 部分的工作，它必须被考虑为您的视窗讯息处理程式的一部分。

如果您的程式在处理某些讯息时需要长时间的作业的话，可以选择我在第二十章里描述的那些方法来做得更有优雅一些。即使是在优先权式多工环境中，也不应该让您的程式呆在萤幕上一动不动。这会让使用者讨厌的，他们会认为您的程式中有 bug、不标准的动作，说明档案没写好。最好让使用者觉得程式只停了一下子就把全部讯息中快速料理完了。

第四章 输出文字

在前一章，您看到了一个简单的 Windows 98 程式，它在视窗中央，或者更准确地说，在显示区域中央显示一行文字。正如我们学到的，显示区域是整个應用程式视窗中未被标题列、视窗边框，以及可选的功能表列、工具列、状态列和卷动列占据的部分。简而言之，显示区域是视窗中可以由程式任意书写和传递视觉资讯的部分。

對於程式的显示区域，您几乎可以为所欲为，只不过您不能假定视窗大小是某一特定尺寸，或者在程式执行时其大小会保持不变。如果您不熟悉图形视窗环境的程式设计，这些限制可能会使您感到惊讶：不能再假设萤幕上的一行文字一定有 80 个字元了。您的程式必须与其他 Windows 程式共用视讯显示器。Windows 使用者控制程式视窗在萤幕上显示的方式。尽管可以建立固定大小的视窗（这對於计算器之类的应用是合理的），但在大多数情况下，使用者应该能够改变應用程式视窗的大小。您的程式必须能够接受指定给它的大小，并且合理地利用这一空间。

这有两种可能的情况。一种可能是，程式只有仅能显示「hello」的显示区域；还有另一种可能，即程式在一个大萤幕、高解析度的系统上执行，其显示区域大得足以显示两整页文字。灵活地处理这两种极端是 Windows 程式设计的要点之一。

这一章，我们将讲述程式在显示区域显示资讯的方式，但比上一章说明的显示方式更加复杂。当程式在显示区域显示文字或图形时，它经常要「绘制」它的显示区域。本章著重讲述绘制的方法。

尽管 Windows 为显示图形提供了强大的图形装置介面（GDI）函式，但在这一章中，我只介绍简单文字行的显示。我也将忽略 Windows 能够使用的不同字体外形及字体大小，仅使用 Windows 的内定系统字体。这看起来似乎是一种限制，其实不然，本章涉及和解决的问题适用於所有 Windows 程式设计。在混合显示文字和图形时，Windows 内定字体的字元大小通常决定了图形的尺寸。

本章表面上是讨论绘图的方法，实际上是讨论与装置无关的程式设计基础。Windows 程式只能对显示区域大小甚至字元的大小做很少的假定，相反地，必须使用 Windows 提供的功能来取得關於程式执行环境的资讯。

绘制和更新

在文字模式环境下，程式可以在显示器的任意部分输出，程式输出到萤幕

上的内容会停留在原处，不会神秘地消失。因此，程式可以丢掉重新生成萤幕显示时所需的资讯。

在 Windows 中，只能在视窗的显示区域绘制文字和图形，而且不能确保在显示区域内显示的内容会一直保留到程式下一次有意地改写它时还保留在那里。例如，使用者可能会在萤幕上移动另一个程式的视窗，这样就可能覆盖您的應用程式视窗的一部分。Windows 不会保存您的视窗中被其他程式覆盖的区域，当程式移开后，Windows 会要求您的程式更新显示区域的这个部分。

Windows 是一个讯息驱动系统。它通过把讯息投入應用程式讯息佇列中或者把讯息发送给合适的视窗讯息处理程式，将发生的各种事件通知给應用程式。Windows 通过发送 WM_PAINT 讯息通知视窗讯息处理程式，视窗的部分显示区域需要绘制。

WM_PAINT 讯息

大多数 Windows 程式在 WinMain 中进入讯息回圈之前的初始化期间都要呼叫函式 UpdateWindow。Windows 利用这个机会给视窗讯息处理程式发送第一个 WM_PAINT 讯息。这个讯息通知视窗讯息处理程式：必须绘制显示区域。此后，视窗讯息处理程式应在任何时刻都准备好处理其他 WM_PAINT 讯息，必要的话，甚至重新绘制视窗的整个显示区域。在发生下面几种事件之一时，视窗讯息处理程式会接收到一个 WM_PAINT 讯息：

- 在使用者移动视窗或显示视窗时，视窗中先前被隐藏的区域重新可见。

- 使用者改变视窗的大小（如果视窗类别样式有著 CS_HREDRAW 和 CS_VREDRAW 位元旗标的设定）。

- 程式使用 ScrollWindow 或 ScrollDC 函式滚动显示区域的一部分。

- 程式使用 InvalidateRect 或 InvalidateRgn 函式刻意产生 WM_PAINT 讯息。

- 在某些情况下，显示区域的一部分被临时覆盖，Windows 试图保存一个显示区域，并在以后恢复它，但这不一定能成功。在以下情况下，Windows 可能发送 WM_PAINT 讯息：

 - Windows 擦除覆盖了部分视窗的对话方块或讯息方块。

 - 功能表下拉出来，然後被释放。

 - 显示工具提示讯息。

- 在某些情况下，Windows 总是保存它所覆盖的显示区域，然後恢复它。这些情况是：

 - 滑鼠游标穿越显示区域。

 - 图示拖过显示区域。

处理 WM_PAINT 讯息要求程式写作者改变自己向显示器输出的思维方式。程式应该组织成可以保留绘制显示区域需要的所有资讯，并且仅当「回应要求」——即 Windows 给视窗讯息处理程式发送 WM_PAINT 讯息时才进行绘制。如果程式在其他时间需要更新其显示区域，它可以强制 Windows 产生一个 WM_PAINT 讯息。这看来似乎是在萤幕上显示内容的一种舍近求远的方法。但您的程式结构可以从中受益。

有效矩形和无效矩形

尽管视窗讯息处理程式一旦接收到 WM_PAINT 讯息之後，就准备更新整个显示区域，但它经常只需要更新一个较小的区域（最常见的是显示区域中的矩形区域）。显然，当对话方块覆盖了部分显示区域时，情况即是如此。在擦除对话方块之後，需要重画的只是先前被对话方块遮住的矩形区域。

这个区域称为「无效区域」或「更新区域」。正是显示区域内无效区域的存在，才会让 Windows 将一个 WM_PAINT 讯息放在应用程式的讯息佇列中。只有在显示区域的某一部分失效时，视窗才会接受 WM_PAINT 讯息。

Windows 内部为每个视窗保存一个「绘图资讯结构」，这个结构包含了包围无效区域的最小矩形的座标以及其他资讯，这个矩形就叫做「无效矩形」，有时也称为「无效区域」。如果在视窗讯息处理程式处理 WM_PAINT 讯息之前显示区域中的另一个区域变为无效，则 Windows 计算出一个包围两个区域的新的无效区域（以及一个新的无效矩形），并将这种变化後的资讯放在绘制资讯结构中。Windows 不会将多个 WM_PAINT 讯息都放在讯息佇列中。

视窗讯息处理程式可以通过呼叫 InvalidateRect 使显示区域内的矩形无效。如果讯息佇列中已经包含一个 WM_PAINT 讯息，Windows 将计算出新的无效矩形。否则，它将一个新的 WM_PAINT 讯息放入讯息佇列中。在接收到 WM_PAINT 讯息时，视窗讯息处理程式可以取得无效矩形的座标（我们马上就会看到这一点）。通过呼叫 GetUpdateRect，可以在任何时候取得这些座标。

在处理 WM_PAINT 讯息处理期间，视窗讯息处理程式在呼叫了 BeginPaint 之後，整个显示区域即变为有效。程式也可以通过呼叫 ValidateRect 函式使显示区域内的任意矩形区域变为有效。如果这呼叫具有令整个无效区域变为有效的效果，则目前佇列中的任何 WM_PAINT 讯息都将被删除。

GDI 简介

要在视窗的显示区域绘图，可以使用 Windows 的图形装置介面 (GDI) 函式。Windows 提供了几个 GDI 函式，用於将字串输出到视窗的显示区域内。我们已经

在上一章看过 DrawText 函式，但是目前使用最为普遍的文字输出函式是 TextOut。该函式的格式如下：

```
TextOut (hdc, x, y, psText, iLength) ;
```

TextOut 向视窗的显示区域写入字串。psText 参数是指向字串的指标，iLength 是字串的长度。x 和 y 参数定义了字串在显示区域的开始位置（不久会讲述关于它们的详细情况）。hdc 参数是「装置内容代号」，它是 GDI 的重要部分。实际上，每个 GDI 函式都需要将这个代号作为函式的第一个参数。

装置内容

读者可能还记得，代号只不过是一个数值，Windows 以它在内部使用物件。程式写作者从 Windows 取得代号，然后在其他函式中使用该代号。装置内容代号是 GDI 函式的视窗「通行证」，有了这种装置内容代号，程式写作者就能自如地在显示区域上绘图，使图形如自己所愿地变得好看或者难看。

装置内容（简称为「DC」）实际上是 GDI 内部保存的资料结构。装置内容与特定的显示设备（如视讯显示器或印表机）相关。对于视讯显示器，装置内容总是与显示器上的特定视窗相关。

装置内容中的有些值是图形「属性」，这些属性定义了 GDI 绘图函式工作的细节。例如，对于 TextOut，装置内容的属性确定了文字的颜色、文字的背景色、x 座标和 y 座标映射到视窗的显示区域的方式，以及显示文字时 Windows 使用的字体。

当程式需要绘图时，它必须先取得装置内容代号。在取得了该代号后，Windows 用内定的属性值填入内部装置内容结构。在后面的章节中您会看到，可以通过呼叫不同的 GDI 函式改变这些预设值。利用其他的 GDI 函式可以取得这些属性的目前值。当然，还有其他的 GDI 函式能够在视窗的显示区域真正地绘图。

当程式在显示区域绘图完毕後，它必须释放装置内容代号。代号被程式释放後就不再有效，且不能再被使用。程式必须在处理单个讯息处理期间取得和释放代号。除了呼叫 CreateDC（函式，在本章暂不讲述）建立的装置内容之外，程式不能在两个讯息之间保存其他装置内容代号。

Windows 應用程式一般使用两种方法来取得装置内容代号，以备在萤幕上绘图。

取得装置内容代号：方法一

在处理 WM_PAINT 讯息时，使用这种方法。它涉及 BeginPaint 和 EndPaint

两个函式，这两个函式需要视窗代号（作为参数传给视窗讯息处理程式）和 PAINTSTRUCT 结构的变数（在 WINUSER.H 表头档案中定义）的地址为参数。Windows 程式写作者通常把这一结构变数命名为 ps 并且在视窗讯息处理程式中定义它：

```
PAINTSTRUCT ps ;
```

在处理 WM_PAINT 讯息时，视窗讯息处理程式首先呼叫 BeginPaint。BeginPaint 函式一般在准备绘制时导致无效区域的背景被擦除。该函式也填入 ps 结构的栏位。BeginPaint 传回的值是装置内容代号，这一传回值通常被保存在叫做 hdc 的变数中。它在视窗讯息处理程式中的定义如下：

```
HDC hdc ;
```

HDC 资料型态定义为 32 位元的无正负号整数。然後，程式就可以使用需要装置内容代号的 TextOut 等 GDI 函式。呼叫 EndPaint 即可释放装置内容代号。

一般地，处理 WM_PAINT 讯息的形式如下：

```
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    使用 GDI 函式
    EndPaint (hwnd, &ps) ;
    return 0 ;
```

在处理 WM_PAINT 讯息时，必须成对地呼叫 BeginPaint 和 EndPaint。如果视窗讯息处理程式不处理 WM_PAINT 讯息，则它必须将 WM_PAINT 讯息传递给 Windows 中 DefWindowProc（内定视窗讯息处理程式）。DefWindowProc 以下列代码处理 WM_PAINT 讯息：

```
case WM_PAINT:
    BeginPaint (hwnd, &ps) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
```

这两个 BeginPaint 和 EndPaint 呼叫之间没有任何叙述，仅仅使先前无效区域变为有效。但以下方法是错误的：

```
case WM_PAINT:
    return 0 ; // WRONG !!!
```

Windows 将一个 WM_PAINT 讯息放到讯息伫列中，是因为显示区域的一部分无效。如果不呼叫 BeginPaint 和 EndPaint（或者 ValidateRect），则 Windows 不会使该区域变为有效。相反，Windows 将发送另一个 WM_PAINT 讯息，且一直发送下去。

绘图资讯结构

前面提到过，Windows 为每个视窗保存一个「绘图资讯结构」，这就是 PAINTSTRUCT，定义如下：

```
typedef struct tagPAINTSTRUCT
```



```

{
    HDC          hdc ;
    BOOL          fErase ;
    RECT          rcPaint ;
    BOOL          fRestore ;
    BOOL          fIncUpdate ;
    BYTE          rgbReserved[32] ;
} PAINTSTRUCT ;

```

在程式呼叫 BeginPaint 时，Windows 会适当填入该结构的各个栏位值。使用者程式只使用前三个栏位，其他栏位由 Windows 内部使用。hdc 栏位是装置内容代号。在旧版本的 Windows 中，BeginPaint 的传回值也曾是这个装置内容代号。在大多数情况下，fErase 被标志为 FALSE(0)，这意味著 Windows 已经擦除了无效矩形的背景。这最早在 BeginPaint 函式中发生（如果要在视窗讯息处理程式中自己定义一些背景擦除行为，可以自行处理 WM_ERASEBKGD 讯息）。Windows 使用 WNDCLASS 结构的 hbrBackground 栏位指定的画刷来擦除背景，这个 WNDCLASS 结构是程式在 WinMain 初始化期间登录视窗类别时使用的。许多 Windows 程式使用白色画刷。以下叙述设定视窗类别结构栏位值：

```
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
```

不过，如果程式通过呼叫 Windows 函式 InvalidateRect 使显示区域中的矩形失效，则该函式的最後一个参数会指定是否擦除背景。如果这个参数为 FALSE（即 0），则 Windows 将不会擦除背景，并且在呼叫完 BeginPaint 後 PAINTSTRUCT 结构的 fErase 栏位将为 TRUE（非零）。

PAINTSTRUCT 结构的 rcPaint 栏位是 RECT 型态的结构。您已经在第三章中看到，RECT 结构定义了一个矩形，其四个栏位为 left、top、right 和 bottom。PAINTSTRUCT 结构的 rcPaint 栏位定义了无效矩形的边界，如图 4-1 所示。这些值均以图素为单位，并相对於显示区域的左上角。无效矩形是应该重画的区域。

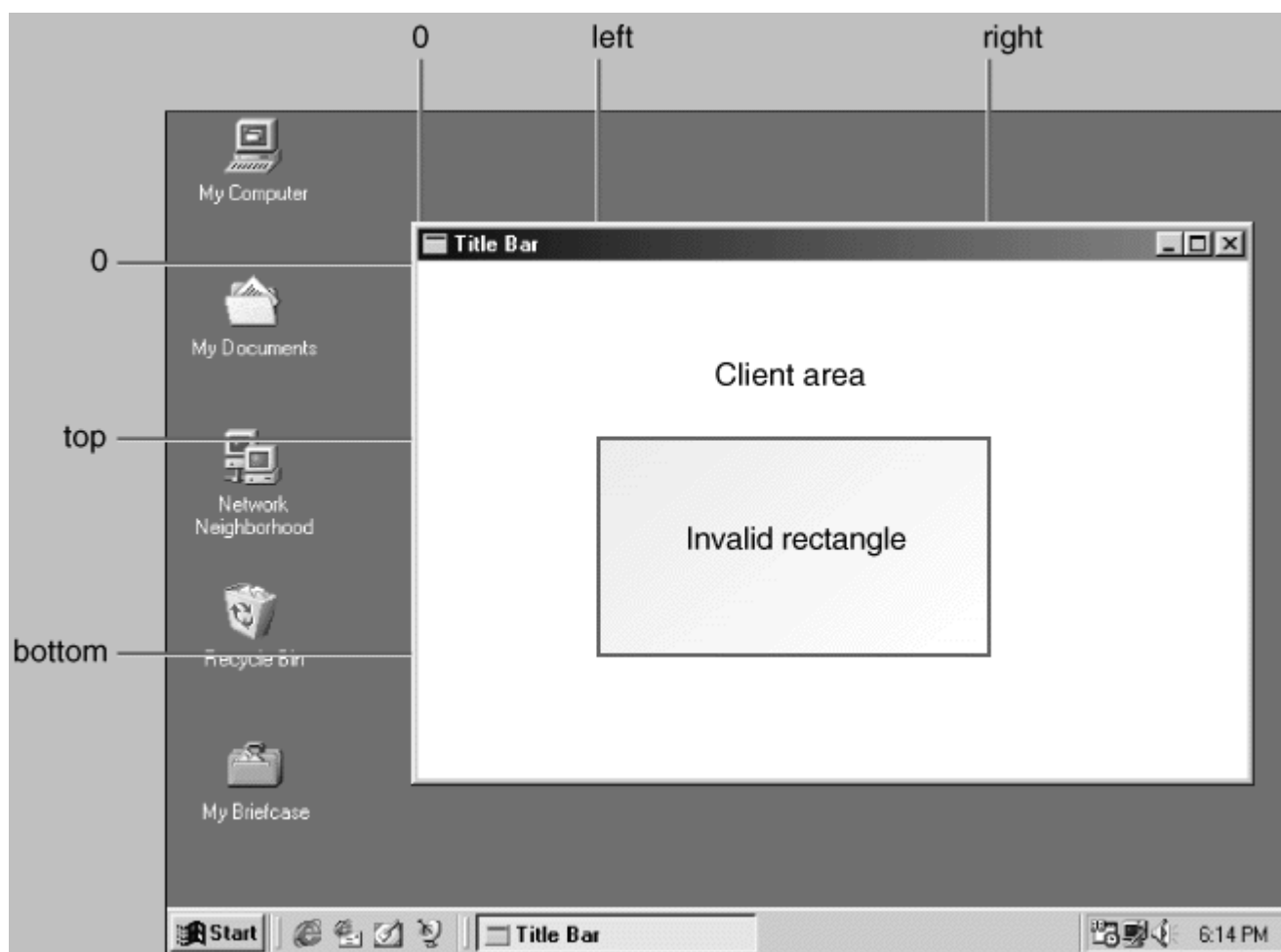


图 4-1 无效矩形的边界

PAINTSTRUCT 中的 rcPaint 矩形不仅是无效矩形，它还是一个「剪取」矩形。这意味著 Windows 将绘图操作限制在剪取矩形内（更确切地说，如果无效矩形区域不为矩形，则 Windows 将绘图操作限制在这个区域内）。

在处理 WM_PAINT 讯息时，为了在更新的矩形外绘图，可以使用如下呼叫：

```
InvalidateRect (hwnd, NULL, TRUE) ;
```

该呼叫在 BeginPaint 呼叫之前进行，它使整个显示区域变为无效，并擦除背景。但是，如果最後一个参数等於 FALSE，则不擦除背景，原有的东西将保留在原处。

通常这是 Windows 程式在无论何时收到 WM_PAINT 讯息而不考虑 rcPaint 结构的情况下简单地重画整个显示区域最方便的方法。例如，如果在显示区域的显示输出中包括了一个圆，但是只有圆的一部分落到了无效矩形中，它就使仅绘制圆的无效部分变得没有意义。这需要画整个圆。在您使用从 BeginPaint 传回的装置内容代号时，Windows 不会绘制 rcPaint 矩形外的任何部分。

在第三章的 HELLOWIN 程式中，我们并不关心处理 WM_PAINT 讯息时的无效矩形。如果文字显示区域恰巧在无效矩形内，则由 DrawText 恢复之。否则，在处理 DrawText 呼叫的某个时刻，Windows 会确定它无须向显示器上输出。不过，

这一决定需要时间。关心程式性能和速度的程式写作者希望在处理 WM_PAINT 期间使用无效矩形范围，以避免不必要的 GDI 呼叫。如果绘制时需要存取例如点阵图这样的磁片档案，则这就显得尤其重要。

取得装置内容代号：方法二

虽然最好是在处理 WM_PAINT 讯息处理期间更新整个显示区域，但是您也会发现在处理非 WM_PAINT 讯息处理期间绘制显示区域的某个部分也是非常有用的。或者您需要将装置内容代号用於其他目的，如取得装置内容的资讯。

要得到视窗显示区域的装置内容代号，可以呼叫 GetDC 来取得代号，在使用完後呼叫 ReleaseDC：

```
hdc = GetDC (hwnd) ;
使用 GDI 函式
ReleaseDC (hwnd, hdc) ;
```

与 BeginPaint 和 EndPaint 一样，GetDC 和 ReleaseDC 函式必须成对地使用。
如果在处理某讯息时呼叫 GetDC，则必须在退出视窗讯息处理程式之前呼叫 ReleaseDC。
不要在一个讯息中呼叫 GetDC 却在另一个讯息呼叫 ReleaseDC。

与从 BeginPaint 传回装置内容代号不同，GetDC 传回的装置内容代号具有一个剪取矩形，它等於整个显示区域。可以在显示区域的某一部分绘图，而不只是在无效矩形上绘图（如果确实存在无效矩形）。与 BeginPaint 不同，GetDC 不会使任何无效区域变为有效。如果需要使整个显示区域有效，可以呼叫

```
ValidateRect (hwnd, NULL) ;
```

一般可以呼叫 GetDC 和 ReleaseDC 来对键盘讯息（如在字处理程式中）和滑鼠讯息（如在画图程式中）作出反应。此时，程式可以立刻根据使用者的键盘或滑鼠输入来更新显示区域，而不需要考虑为了视窗的无效区域而使用 WM_PAINT 讯息。不过，一旦确实收到了 WM_PAINT 讯息，程式就必须收集足够的资讯後才能更新显示。

与 GetDC 相似的函式是 GetWindowDC。GetDC 传回用於写入视窗显示区域的装置内容代号，而 GetWindowDC 传回写入整个视窗的装置内容代号。例如，您的程式可以使用从 GetWindowDC 传回的装置内容代号在视窗的标题列上写入文字。然而，程式同样也应该处理 WM_NCPAINT （「非显示区域绘制」）讯息。

TextOut：细节

TextOut 是用於显示文字的最常用的 GDI 函式。语法是：

```
TextOut (hdc, x, y, psText, iLength) ;
```

以下将详细地讨论这个函式。

第一个参数是装置内容代号，它既可以是 GetDC 的传回值，也可以是在处理 WM_PAINT 讯息时 BeginPaint 的传回值。

装置内容的属性控制了被显示的字串的特徵。例如，装置内容中有一个属性指定文字颜色，内定颜色为黑色；内定装置内容还定义了白色的背景。在程式向显示器输出文字时，Windows 使用这个背景色来填入字元周围的矩形空间（称为「字元框」）。

该文字背景色与定义视窗类别时设置的背景并不相同。视窗类别中的背景是一个画刷，它是一种纯色或者非纯色组成的画刷，Windows 用它来擦除显示区域，它不是装置内容结构的一部分。在定义视窗类别结构时，大多数 Windows 应用程式使用 WHITE_BRUSH，以便内定装置内容中的内定文字背景颜色与 Windows 用以擦除显示区域背景的画刷颜色相同。

psText 参数是指向字串的指标，iLength 是字串中字元的个数。如果 psText 指向 Unicode 字串，则字串中的位元组数就是 iLength 值的两倍。字串中不能包含任何 ASCII 控制字元（如回车、换行、制表或退格），Windows 会将这些控制字元显示为实心块。TextOut 不识别作为字串结束标志的内容为零的位元组（对於 Unicode，是一个短整数型态的 0），而需要由 nLength 参数指明长度。

TextOut 中的 x 和 y 定义显示区域内字串的开始位置，x 是水平位置，y 是垂直位置。字串中第一个字元的左上角位於座标点 (x, y)。在内定的装置内容中，原点 (x 和 y 均为 0 的点) 是显示区域的左上角。如果在 TextOut 中将 x 和 y 设为 0，则将从显示区域左上角开始输出字串。

当您阅读 GDI 绘图函式（例如 TextOut）的文件时，就会发现传递给函式的座标常常被称为「逻辑座标」。在第五章会详细地解释这种情况。现在请注意，Windows 有许多「座标映射方式」，它们用来控制 GDI 函式指定的逻辑座标转换为显示器的实际图素座标的方式。映射方式在装置内容中定义，内定映射方式是 MM_TEXT（使用 WINGDI.H 中定义的识别字）。在 MM_TEXT 映射方式下，逻辑单位与实际单位相同，都是图素；x 的值从左向右递增，y 的值从上向下递增（参看图 4-2）。MM_TEXT 座标系与 Windows 在 PAINTSTRUCT 结构中定义无效矩形时使用的座标系相同，这为我们带来了很方便（但是，其他映射方式并非如此）。

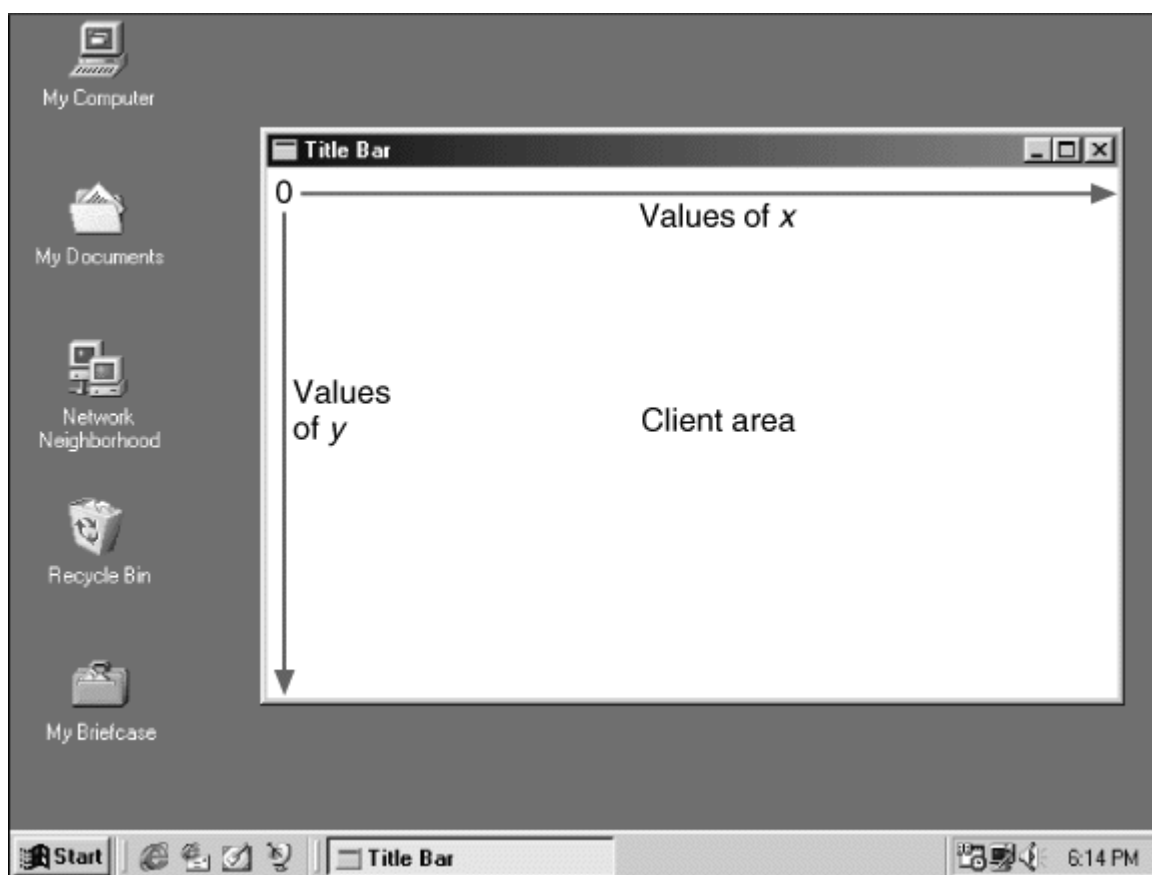


图 4-2 MM_TEXT 映射方式下的 x 座标和 y 座标

装置内容也定义了一个剪裁区域。您已经看到，对于从 GetDC 取得的装置内容代号，内定剪裁区域是整个显示区域；而对于从 BeginPaint 取得的装置内容代号，则为无效区域。Windows 不会在剪裁区域之外的任何位置显示字串。如果一个字元有一部分在剪裁区域外，则 Windows 将只显示此区域内的那部分。要想将输出写到视窗的显示区域之外不是那么容易的，所以不用担心会无意间出现这种事情。

系统字体

装置内容还定义了在您呼叫 TextOut 显示文字时 Windows 使用的字体。内定字体为「系统字体」，或用 Windows 表头档案中的识别字，即 SYSTEM_FONT。系统字体是 Windows 用来在标题列、功能表和对话方块中显示字串的内定字体。

在 Windows 的早期版本中，系统字体是等宽(fixed-pitch)字体，这意味著所有字元均具有同样的宽度，非常类似於打字机。然而，从 Windows 3.0 开始，系统字体成为一种变宽(variable-pitch)字体，这意味著不同的字元具有不同的大小，比如，「W」要比「i」宽。变宽字体比等宽字体好读，这已经是公认的事实。不过，可以想见，这一转变使很多原来的 Windows 程式码不再适用，从而要求程式写作者学习一些使用字体的新技术。

系统字体是一种「点阵字体」，这意味著字元被定义为图素块（在第十七

章，将讨论 TrueType 字体，它是由轮廓定义的）。至於确切的大小，系统字体的字元大小取决於视讯显示器的大小。系统字体设计为至少能在显示器上显示 25 行 80 列文字。

字元大小

要用 TextOut 显示多行文字，就必须确定字体的字元大小，可以根据字元的高度来定位字元的後续行，以及根据字元的宽度来定位字元的後续列。

系统字体的字元高度和平均宽度是多少？这个问题取决於视讯显示器的图素大小。Windows 需要的最小显示大小是 640 480，但是许多使用者更喜欢 800 600 或 1024 768 的显示大小。另外，对於这些较大的显示尺寸，Windows 允许使用者选择不同大小的系统字体。

程式可以呼叫 GetSystemMetrics 函式以取使用者介面各类视觉元件大小的资讯，呼叫 GetTextMetrics 取得字体大小。GetTextMetrics 传回装置内容中目前选取的字体资讯，因此它需要装置内容代号。Windows 将文字大小的不同值复制到在 WINGDI.H 中定义的 TEXTMETRIC 型态的结构中。TEXTMETRIC 结构有 20 个栏位，我们只使用前七个：

```
typedef struct tagTEXTMETRIC
{
    LONG tmHeight ;
    LONG tmAscent ;
    LONG tmDescent ;
    LONG tmInternalLeading ;
    LONG tmExternalLeading ;
    LONG tmAveCharWidth ;
    LONG tmMaxCharWidth ;
    其他结构栏位
}
TEXTMETRIC, * PTEXTMETRIC ;
```

这些栏位值的单位取决於选定的装置内容映射方式。在内定装置内容下，映射方式是 MM_TEXT，因此值的大小是以图素为单位。

要使用 GetTextMetrics 函式，需要先定义一个结构变数（通常称为 tm）：

```
TEXTMETRIC tm ;
```

在需要确定文字大小时，先取得装置内容代号，再呼叫 GetTextMetrics：

```
hdc = GetDC (hwnd) ;
GetTextMetrics (hdc, &tm) ;
ReleaseDC (hwnd, hdc) ;
```

此後，您就可以查看文字尺寸结构中的值，并有可能保存其中的一些以备将来使用。

文字大小：细节

TEXTMETRIC 结构提供了关于目前装置内容中选用的字体的丰富资讯。但是，字体的纵向大小只由 5 个值确定，其中 4 个值如图 4-3 所示。

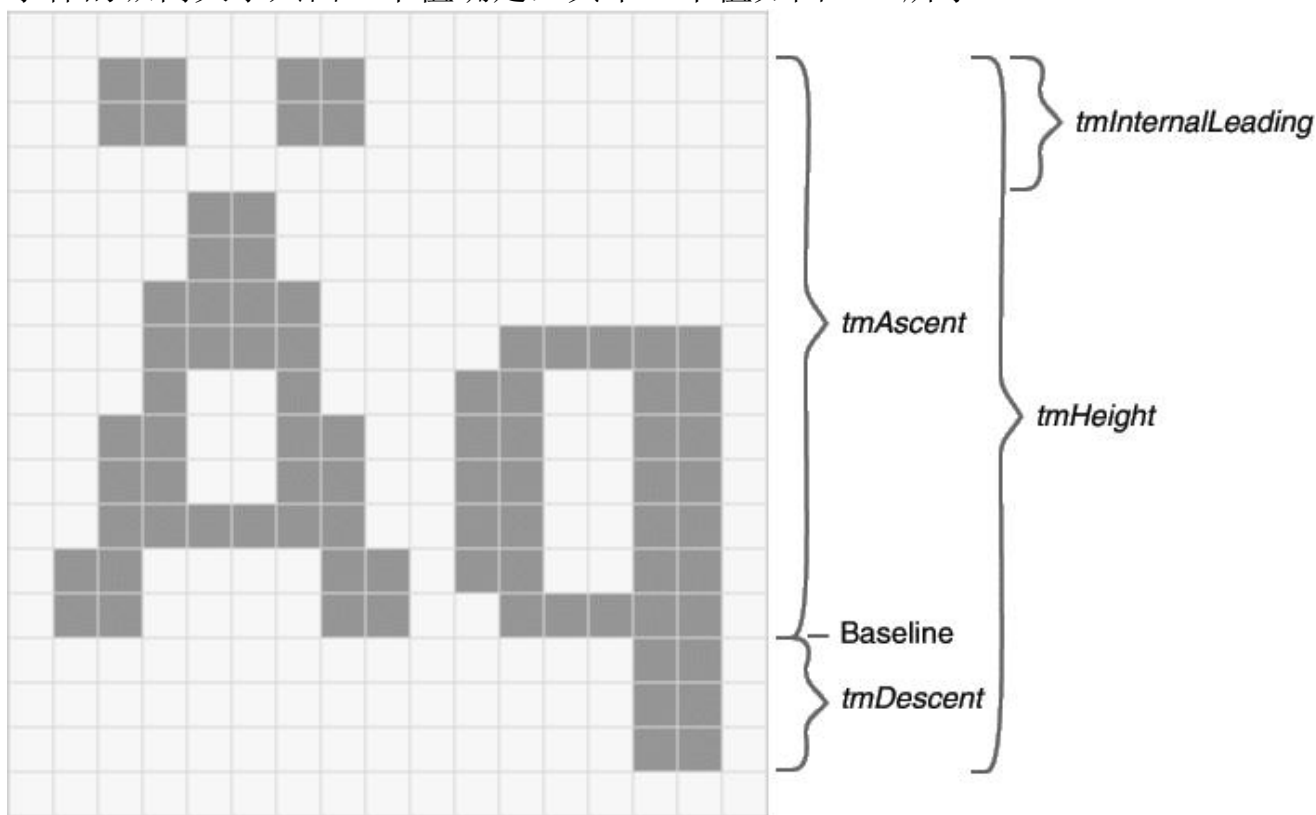


图 4-3 定义字体中纵向字元大小的 4 个值

最重要的值是 `tmHeight`，它是 `tmAscent` 和 `tmDescent` 的和。这两个值表示了基准线上下字元的最大纵向高度。「间距」(leading)指印表机在两行文字间插入的空间。在 TEXTMETRIC 结构中，内部的间距包括在 `tmAscent` 中（因此也在 `tmHeight` 中），并且它经常是重音符号出现的地方。`tmInternalLeading` 栏位可被设成 0，在这种情况下，加重音的字母会稍稍缩短以便容纳重音符号。

TEXTMETRIC 结构还包括一个不包含在 `tmHeight` 值中的栏位 `tmExternalLeading`。它是字体设计者建议加在横向字元之间的空间大小。在安排文字行之间的空隙时，您可以接受设计者建议的值，也可以拒绝它。在系统字体中 `tmExternalLeading` 可以为 0，因此我没有在图 4-3 中显示它。（尽管我不想告诉你们，图 4-3 确实就是 Windows 在 640 480 的显示解析度中使用的系统字体。）

TEXTMETRICS 结构包含有描述字元宽度的两个栏位，即 `tmAveCharWidth`（小写字母加权平均宽度）和 `tmMaxCharWidth`（字体中最宽字元的宽度）。对于定宽字体，这两个值是相等的（图 4-3 中这些值分别为 7 和 14）。

本章的范例程式还需要另一种字元宽度，即大写字母的平均宽度，这可以

用 `tmAveCharWidth` 乘以 150% 大致计算出来。

必须认识到，系统字体的大小取决於 Windows 所执行的视讯显示器的解析度，在某些情况下，取决於使用者选取的系统字体的大小。Windows 提供了一个与装置无关的图形介面，但程式写作者还是有事情要处理的。不要想当然耳地猜测字体大小来写作 Windows 程式，也不要把值定死，您可以使用 `GetTextMetrics` 函式取得这一资讯。

格式化文字

Windows 启动後，系统字体的大小就不会发生改变，所以在程式执行过程中，程式写作者只需要呼叫一次 `GetTextMetrics`。最好是在视窗讯息处理程式中处理 `WM_CREATE` 讯息时进行此呼叫，`WM_CREATE` 讯息是视窗讯息处理程式接收的第一个讯息。在 `WinMain` 中呼叫 `CreateWindow` 时，Windows 会以一个 `WM_CREATE` 讯息呼叫视窗讯息处理程式。

假设要编写一个 Windows 程式，在显示区域显示几行文字，这需要先取得字元宽度和高度。您可以在视窗讯息处理程式内定义两个变数来保存平均字元宽度 (`cxChar`) 和总的字元高度 (`cyChar`)：

```
static int cxChar, cyChar ;
```

变数名的字首 `c` 代表「count」，在这里指图素数，与 `x` 和 `y` 结合，分别指宽和高。这些变数定义为 `static` 静态变数，因为它们在视窗讯息处理程式中处理其他讯息（如 `WM_PAINT`）时也应该是有有效的。如果变数在函式外面定义，则不需要定义为 `static`。

下面是取得系统字体的字元宽度和高度的 `WM_CREATE` 程式码：

```
case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;

    cxChar = tm.tmAveCharWidth ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;

    ReleaseDC (hwnd, hdc) ;
    return 0 ;
```

注意我在计算 `cyChar` 时包括了 `tmExternalLeading` 栏位，虽然该栏位在系统字体中为 0，但是因为它使得文字的可读性更好，所以还是应该把它包括进去。沿著视窗向下每隔 `cyChar` 图素就会显示一行文字。

您会发现常常需要显示格式化的数字跟简单的字串。我在第二章讲到过，您不能使惯用的工具（可爱的 `printf` 函式）来完成这项工作，但是可以使用 `sprintf` 和 Windows 版的 `sprintf`——`wsprintf`。这些函式与 `printf` 相似，只

是把格式化字串放到字串中。然後，可以用 `TextOut` 将字串输出到显示器上。非常方便的是，从 `sprintf` 和 `wsprintf` 传回的值就是字串的长度。您可以将这个值传递给 `TextOut` 作为 `iLength` 参数。下面的程式码显示了 `wsprintf` 与 `TextOut` 的典型组合：

```
int    iLength ;
TCHAR szBuffer [40] ;
```

其他行程式

```
iLength = wsprintf (szBuffer, TEXT ("The sum of %i and %i is %i"),
                    iA, iB, iA + iB) ;
TextOut (hdc, x, y, szBuffer, iLength) ;
```

对于这样简单的情况，可以将 `nLength` 的定义值与 `TextOut` 放在同一条叙述中，从而无需定义 `iLength`：

```
TextOut (hdc, x, y, szBuffer,
         wsprintf (szBuffer, TEXT ("The sum of %i and %i is %i"),
                    iA, iB, iA + iB)) ;
```

虽然这样子写起来不好看，但是功能与前者是一样的。

综合使用

现在，我们似乎已经具备了在萤幕上显示多行文字所需要的所有知识。我们知道如何在 `WM_PAINT` 讯息处理期间取得一个装置内容代号，如何使用 `TextOut` 函式以及如何根据字元大小来安排字距，剩下的就是显示一点有意义的东西了。

在上一章里，我们大概知道从 Windows 的 `GetSystemMetrics` 函式中取得的资讯是很有意义的，该函式传回 Windows 中不同视觉元件的大小资讯，如图示、游标、标题列和卷动列等。它们的大小因显示卡和驱动程式的不同而有所不同。`GetSystemMetrics` 是在程式中完成与装置无关图形输出的重要函式。

该函式需要一个参数，叫做「索引」，在 Windows 表头档案定义了 75 个整数索引识别字（识别字的数量随著每个版本的 Windows 的发布而不断地增加，在 Windows 1.0 的程式写作者文件中仅列出了 26 个）。`GetSystemMetrics` 传回一个整数，这个整数通常就是参数中指定的图形元件大小。

让我们来编写一个程式，显示一些可以从 `GetSystemMetrics` 呼叫中取得的资讯，显示格式为每种视觉元件一行。如果我们建立一个表头档案，在表头档案中定义一个结构阵列，此结构包含 `GetSystemMetrics` 索引对应的 Windows 表头档案识别字和呼叫所传回的每个值对应的字串，这样处理起来要容易一些。表头档案名为 `SYSMETS.H`，如程式 4-1 所示。

程式 4-1 `SYSMETS.H`

```
/*-----
SYSMETS.H -- System metrics display structure
```

```

-----*/
#define NUMLINES ((int) (sizeof sysmetrics / sizeof sysmetrics [0]))
struct
{
    int    Index ;
    TCHAR *    szLabel ;
    TCHAR *    szDesc ;
}
sysmetrics [] =
{
    SM_CXSCREEN,    TEXT ("SM_CXSCREEN"),
                    TEXT ("Screen width in pixels"),
    SM_CYSCREEN,    TEXT ("SM_CYSCREEN"),
                    TEXT ("Screen height in pixels"),
    SM_CXVSCROLL,   TEXT ("SM_CXVSCROLL"),
                    TEXT ("Vertical scroll width"),
    SM_CXHSCROLL,   TEXT ("SM_CXHSCROLL"),
                    TEXT ("Horizontal scroll height"),
    SM_CYCAPTION,   TEXT ("SM_CYCAPTION"),
                    TEXT ("Caption bar height"),
    SM_CXBORDER,    TEXT ("SM_CXBORDER"),
                    TEXT ("Window border width"),
    SM_CYBORDER,    TEXT ("SM_CYBORDER"),
                    TEXT ("Window border height"),
    SM_CXFIXEDFRAME,TEXT ("SM_CXFIXEDFRAME"),
                    TEXT ("Dialog window frame width"),
    SM_CYFIXEDFRAME,TEXT ("SM_CYFIXEDFRAME"),
                    TEXT ("Dialog window frame height"),
    SM_CXVTHUMB,    TEXT ("SM_CXVTHUMB"),
                    TEXT ("Vertical scroll thumb height"),
    SM_CXHTHUMB,    TEXT ("SM_CXHTHUMB"),
                    TEXT ("Horizontal scroll thumb width"),
    SM_CXICON,      TEXT ("SM_CXICON"),
                    TEXT ("Icon width"),
    SM_CYICON,      TEXT ("SM_CYICON"),
                    TEXT ("Icon height"),
    SM_CXCURSOR,    TEXT ("SM_CXCURSOR"),
                    TEXT ("Cursor width"),
    SM_CYCURSOR,    TEXT ("SM_CYCURSOR"),
                    TEXT ("Cursor height"),
    SM_CYMENU,      TEXT ("SM_CYMENU"),
                    TEXT ("Menu bar height"),
    SM_CXFULLSCREEN,TEXT ("SM_CXFULLSCREEN"),
                    TEXT ("Full screen client area width"),
    SM_CYFULLSCREEN,TEXT ("SM_CYFULLSCREEN"),
                    TEXT ("Full screen client area height"),
    SM_CYKANJIWINDOW,TEXT ("SM_CYKANJIWINDOW"),
                    TEXT ("Kanji window height"),

```

```
SM_MOUSEPRESENT, TEXT ("SM_MOUSEPRESENT"),
                    TEXT ("Mouse present flag"),
SM_CVSCROLL,      TEXT ("SM_CVSCROLL"),
                    TEXT ("Vertical scroll arrow height"),
SM_CXHSCROLL,     TEXT ("SM_CXHSCROLL"),
                    TEXT ("Horizontal scroll arrow width"),
SM_DEBUG,         TEXT ("SM_DEBUG"),
                    TEXT ("Debug version flag"),
SM_SWAPBUTTON,    TEXT ("SM_SWAPBUTTON"),
                    TEXT ("Mouse buttons swapped flag"),
SM_CXMIN,         TEXT ("SM_CXMIN"),
                    TEXT ("Minimum window width"),
SM_CYMIN,         TEXT ("SM_CYMIN"),
                    TEXT ("Minimum window height"),
SM_CXSIZE,        TEXT ("SM_CXSIZE"),
                    TEXT ("Min/Max/Close button width"),
SM_CYSIZE,        TEXT ("SM_CYSIZE"),
                    TEXT ("Min/Max/Close button height"),
SM_CXSIZEFRAME,   TEXT ("SM_CXSIZEFRAME"),
                    TEXT ("Window sizing frame width"),
SM_CYSIZEFRAME,   TEXT ("SM_CYSIZEFRAME"),
                    TEXT ("Window sizing frame height"),
SM_CXMINTRACK,    TEXT ("SM_CXMINTRACK"),
                    TEXT ("Minimum window tracking width"),
SM_CYMINTRACK,    TEXT ("SM_CYMINTRACK"),
                    TEXT ("Minimum window tracking height"),
SM_CXDOUBLECLK,   TEXT ("SM_CXDOUBLECLK"),
                    TEXT ("Double click x tolerance"),
SM_CYDOUBLECLK,   TEXT ("SM_CYDOUBLECLK"),
                    TEXT ("Double click y tolerance"),
SM_CXICONSPACING, TEXT ("SM_CXICONSPACING"),
                    TEXT ("Horizontal icon spacing"),
SM_CYICONSPACING, TEXT ("SM_CYICONSPACING"),
                    TEXT ("Vertical icon spacing"),
SM_MENUDROPALIGNMENT, TEXT ("SM_MENUDROPALIGNMENT"),
                    TEXT ("Left or right menu drop"),
SM_PENWINDOWS,    TEXT ("SM_PENWINDOWS"),
                    TEXT ("Pen extensions installed"),
SM_DBCSENABLED,   TEXT ("SM_DBCSENABLED"),
                    TEXT ("Double-Byte Char Set enabled"),
SM_CMOUSEBUTTONS, TEXT ("SM_CMOUSEBUTTONS"),
                    TEXT ("Number of mouse buttons"),
SM_SECURE,        TEXT ("SM_SECURE"),
                    TEXT ("Security present flag"),
SM_CXEDGE,        TEXT ("SM_CXEDGE"),
                    TEXT ("3-D border width"),
SM_CYEDGE,        TEXT ("SM_CYEDGE"),
                    TEXT ("3-D border height"),
```

```

SM_CXMINSPACING,    TEXT ("SM_CXMINSPACING"),
                      TEXT ("Minimized window spacing width"),
SM_CYMINSPACING,    TEXT ("SM_CYMINSPACING"),
                      TEXT ("Minimized window spacing height"),
SM_CXSMICON,        TEXT ("SM_CXSMICON"),
                      TEXT ("Small icon width"),
SM_CYSMICON,        TEXT ("SM_CYSMICON"),
                      TEXT ("Small icon height"),
SM_CYSMCAPTION,     TEXT ("SM_CYSMCAPTION"),
                      TEXT ("Small caption height"),
SM_CXSMSIZE,        TEXT ("SM_CXSMSIZE"),
                      TEXT ("Small caption button width"),
SM_CYSMSIZE,        TEXT ("SM_CYSMSIZE"),
                      TEXT ("Small caption button height"),
SM_CXMENUSIZE,      TEXT ("SM_CXMENUSIZE"),
                      TEXT ("Menu bar button width"),
SM_CYMENUSIZE,      TEXT ("SM_CYMENUSIZE"),
                      TEXT ("Menu bar button height"),
SM_ARRANGE,         TEXT ("SM_ARRANGE"),
                      TEXT ("How minimized windows arranged"),
SM_CXMINIMIZED,     TEXT ("SM_CXMINIMIZED"),
                      TEXT ("Minimized window width"),
SM_CYMINIMIZED,     TEXT ("SM_CYMINIMIZED"),
                      TEXT ("Minimized window height"),
SM_CXMAXTRACK,      TEXT ("SM_CXMAXTRACK"),
                      TEXT ("Maximum draggable width"),
SM_CYMAXTRACK,      TEXT ("SM_CYMAXTRACK"),
                      TEXT ("Maximum draggable height"),
SM_CXMAXIMIZED,     TEXT ("SM_CXMAXIMIZED"),
                      TEXT ("Width of maximized window"),
SM_CYMAXIMIZED,     TEXT ("SM_CYMAXIMIZED"),
                      TEXT ("Height of maximized window"),
SM_NETWORK,         TEXT ("SM_NETWORK"),
                      TEXT ("Network present flag"),
SM_CLEANBOOT,       TEXT ("SM_CLEANBOOT"),
                      TEXT ("How system was booted"),
SM_CXDRAG,          TEXT ("SM_CXDRAG"),
                      TEXT ("Avoid drag x tolerance"),
SM_CYDRAG,          TEXT ("SM_CYDRAG"),
                      TEXT ("Avoid drag y tolerance"),
SM_SHOWSOUNDS,      TEXT ("SM_SHOWSOUNDS"),
                      TEXT ("Present sounds visually"),
SM_CXMENUCHECK,     TEXT ("SM_CXMENUCHECK"),
                      TEXT ("Menu check-mark width"),
SM_CYMENUCHECK,     TEXT ("SM_CYMENUCHECK"),
                      TEXT ("Menu check-mark height"),
SM_SLOWMACHINE,     TEXT ("SM_SLOWMACHINE"),
                      TEXT ("Slow processor flag"),

```

```

SM_MIDEASTENABLED,      TEXT ("SM_MIDEASTENABLED"),
                        TEXT ("Hebrew and Arabic enabled flag"),
SM_MOUSEWHEELPRESENT,  TEXT ("SM_MOUSEWHEELPRESENT"),
                        TEXT ("Mouse wheel present flag"),
SM_XVIRTUALSCREEN,     TEXT ("SM_XVIRTUALSCREEN"),
                        TEXT ("Virtual screen x origin"),
SM_YVIRTUALSCREEN,     TEXT ("SM_YVIRTUALSCREEN"),
                        TEXT ("Virtual screen y origin"),
SM_CXVIRTUALSCREEN,    TEXT ("SM_CXVIRTUALSCREEN"),
                        TEXT ("Virtual screen width"),
SM_CYVIRTUALSCREEN,    TEXT ("SM_CYVIRTUALSCREEN"),
                        TEXT ("Virtual screen height"),
SM_CMONITORS,          TEXT ("SM_CMONITORS"),
                        TEXT ("Number of monitors"),
SM_SAMEDISPLAYFORMAT,  TEXT ("SM_SAMEDISPLAYFORMAT"),
                        TEXT ("Same color format flag")
} ;

```

显示资讯的程式命名为 SYSMETS1。SYSMETS1.C 的原始码如程式 4-2 所示。现在大多数程式码看起来都很熟悉。WinMain 中的程式码实际上与 HELLOWIN 中的程式码相同，并且 WndProc 中的大部分程式码都已经讨论过了。

程式 4-2 SYSMETS1.C

```

/*-----
SYSMETS1.C -- System Metrics Display Program No. 1
              (c) Charles Petzold, 1998
-----*/
#include <windows.h>
#include "sysmets.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("SysMets1") ;
    HWND  hwnd ;
    MSG   msg ;
    WNDCLASS  wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

```

```
if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 1"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}
```

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxChar, cxCaps, cyChar ;
    HDC        hdc ;
    int         i ;
    PAINTSTRUCT ps ;
    TCHAR       szBuffer [10] ;
    TEXTMETRIC  tm ;

    switch (message)
    {
    case WM_CREATE:
        hdc = GetDC (hwnd) ;

        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

    case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;

        for (i = 0 ; i < NUMLINES ; i++)
```

```
        {
            TextOut (hdc, 0, cyChar * i,
                    sysmetrics[i].szLabel,
                    lstrlen (sysmetrics[i].szLabel)) ;

            TextOut (hdc, 22 * cxCaps, cyChar * i,
                    sysmetrics[i].szDesc,
                    lstrlen (sysmetrics[i].szDesc)) ;

            SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
            TextOut (hdc, 22 * cxCaps + 40 * cxChar, cyChar * i, szBuffer,
                    wsprintf (szBuffer, TEXT ("%5d"),
                    GetSystemMetrics (sysmetrics[i].iIndex))) ;
            SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;
    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图 4-4 显示了在标准 VGA 上执行的 SYSMETS1。在程式显示区域的前两行可以看到，萤幕宽度是 640 个图素，萤幕高度是 480 个图素，这两个值以及程式所显示的其他值可能会因视讯显示器型态的不同而不同。

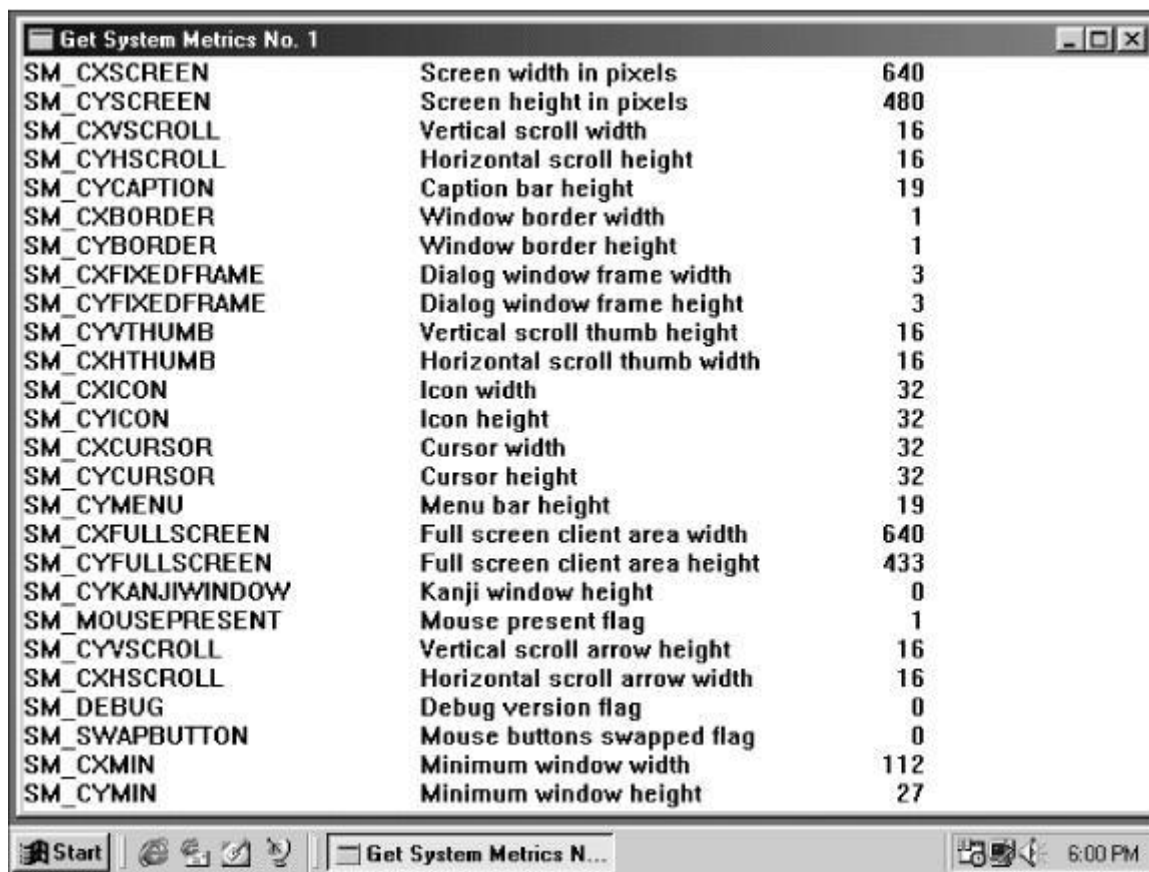


图 4-4 SYSMETS1 的显示

SYSMETS1.C 视窗讯息处理程式

SYSMETS1.C 程式中的 WndProc 视窗讯息处理程式处理三个讯息: WM_CREATE、WM_PAINT 和 WM_DESTROY。WM_DESTROY 讯息的处理方法与第三章的 HELLOWIN 程式相同。

WM_CREATE 讯息是视窗讯息处理程式接收到的第一个讯息。在 CreateWindow 函式建立视窗时, Windows 产生这个讯息。在处理 WM_CREATE 讯息时, SYSMETS1 呼叫 GetDC 取得视窗的装置内容, 并呼叫 GetTextMetrics 取得内定系统字体的文字大小。SYSMETS1 将平均字元宽度保存在 cxChar 中, 将字元的总高度(包括外部间距)保存在 cyChar 中。

SYSMETS1 还将大写字母的平均宽度保存在静态变数 cxCaps 中。对于固定宽度的字体, cxCaps 等於 cxChar。对于可变宽度字体, cxCaps 设定为 cxChar 乘以 150%。对于可变宽度字体, TEXTMETRIC 结构中的 tmPitchAndFamily 栏位的低位元为 1, 对于固定宽度字体, 该值为 0。SYSMETS1 使用这个位元从 cxChar 计算 cxCaps:

```
cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
```

SYSMETS1 在处理 WM_PAINT 讯息处理期间完成所有视窗建立工作。通常, 视窗讯息处理程式先呼叫 BeginPaint 取得装置内容代号, 然後用一道 for 叙述对

SYSMETS.H 中定义的 `sysmetrics` 结构的每一行进行回圈。三列文字用三个 `TextOut` 函式显示, 对於每一列, `TextOut` 的第三个参数都设定为:

```
cyChar * i
```

这个参数指示了字串顶端相对於显示区域顶部的图素位置。

第一条 `TextOut` 叙述在第一列显示了大写识别字。`TextOut` 的第二个参数是 0, 这是说文字从显示区域的左边缘开始。文字的内容来自 `sysmetrics` 结构的 `szLabel` 栏位。我使用 Windows 函式 `lstrlen` 来计算字串的长度, 它是 `TextOut` 需要的最後一个参数。

第二条 `TextOut` 叙述显示了对系统尺寸值的描述。这些描述存放在 `sysmetrics` 结构的 `szDesc` 栏位中。在这种情况下, `TextOut` 的第二个参数设定为:

```
22 * cxCaps
```

第一列显示的最长的大写识别字有 20 个字元, 因此第二列必须在第一列文字开头向右 20 `cxCaps` 处开始。我使用 22, 以在两列之间加一点多余的空间。

第三条 `TextOut` 叙述显示从 `GetSystemMetrics` 函式取得的数值。变宽字体使得格式化向右对齐的数值有些棘手。从 0 到 9 的数字具有相同的宽度, 但是这个宽度比空格宽度大。数值可以比一个数字宽, 所以不同的数值应该从不同的横向位置开始。

那么, 如果我们指定字串结束的图素位置, 而不是指定字串的开始位置, 以此向右对齐数值, 是否会容易一些呢? 用 `SetTextAlign` 函式就可以做到这一点。在 `SYSMETS1` 呼叫:

```
SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
```

之後, 传给後续 `TextOut` 函式的座标将指定字串的右上角, 而不是左上角。

显示列数的 `TextOut` 函式的第二个参数设定为:

```
22 * cxCaps + 40 * cxChar
```

值 `40*cxChar` 包含了第二列的宽度和第三列的宽度。在 `TextOut` 函式之後, 另一个对 `SetTextAlign` 的呼叫将对齐方式设定回普通方式, 以进行下次回圈。

空间不够

在 `SYSMETS1` 程式中存在著一个很难处理的问题: 除非您有一个大萤幕跟高解析度的显示卡, 否则就无法看到系统尺度列表的最後几行。如果视窗太窄, 甚至根本看不到值。

`SYSMETS1` 不知道这个问题。否则我们会显示一个讯息方块说「抱歉!」程式甚至不知道它的显示区域有多大, 它从视窗顶部开始输出文字, 并仰赖 Windows 裁剪超出显示区域底部的内容。

显然，这很不理想。为了解决这个问题，我们的第一个任务是确定程式在显示区域内能输出多少内容。

显示区域的大小

如果您使用过现有的 Windows 應用程式，可能会发现视窗的尺寸变化极大。视窗最大化时（假定视窗只有标题列并且没有功能表），显示区域几乎占据了整个萤幕。这一最大化了的显示区域的尺寸可以通过以 `SM_CXFULLSCREEN` 和 `SM_CYFULLSCREEN` 为参数呼叫 `GetSystemMetrics` 来获得。视窗的最小尺寸可以很小，有时甚至不存在，更不用说显示区域了。

在最近一章，我们使用 `GetClientRect` 函式来取得显示区域的大小。使用这个函式没有什么不好，但是在您每次要使用资讯时就去呼叫它一遍是没有效率的。确定视窗显示区域大小的更好方法是在视窗讯息处理程式中处理 `WM_SIZE` 讯息。在视窗大小改变时，Windows 给视窗讯息处理程式发送一个 `WM_SIZE` 讯息。传给视窗讯息处理程式的 `lParam` 参数的低字组中包含显示区域的宽度，高字组中包含显示区域的高度。要保存这些尺寸，需要在视窗讯息处理程式中定义两个静态变数：

```
static int cxClient, cyClient ;
```

与 `cxChar` 和 `cyChar` 相似，这两个变数在视窗讯息处理程式内定义为静态变数，因为在以后处理其他讯息时会用到它们。处理 `WM_SIZE` 的方法如下：

```
case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
```

实际上您会在每个 Windows 程式中看到类似的程式码。`LOWORD` 和 `HIWORD` 巨集在 Windows 表头档案 `WINDEF.H` 中定义。这些巨集的定义看起来像这样：

```
#define LOWORD(l) ((WORD)(l))
#define HIWORD(l) ((WORD)(((DWORD)(l) >> 16) & 0xFFFF))
```

这两个巨集传回 `WORD` 值（16 位元的无正负号整数，范围从 0 到 `0xFFFF`）。一般，将这些值保存在 32 位元有号整数中。这就不会牵扯到任何转换问题，并使得这些值在以后需要的任何计算中易於使用。

在许多 Windows 程式中，`WM_SIZE` 讯息必然跟著一个 `WM_PAINT` 讯息。为什么呢？因为在我们定义视窗类别时指定视窗类别样式为：

```
CS_HREDRAW | CS_VREDRAW
```

这种视窗类别样式告诉 Windows，如果水平或者垂直大小发生改变，则强制更新显示区域。

用如下公式计算可以在显示区域内显示的文字的总行数：

```
cyClient / cyChar
```

如果显示区域的高度太小以至无法显示一个完整的字元，这个公式的结果可以为 0。类似地，在显示区域的水平方向可以显示的小写字元的近似数目为：

```
cxClient / cxChar
```

如果在处理 WM_CREATE 讯息处理期间取得 cxChar 和 cyChar，则不用担心在这两个计算公式中会出现被 0 除的情况。在 WinMain 呼叫 CreateWindow 时，视窗讯息处理程式接收一个 WM_CREATE 讯息。在 WinMain 呼叫 ShowWindow 之後接收到第一个 WM_CREATE 讯息，此时 cxChar 和 cyChar 已经被赋予正的非零值了。

如果显示区域的大小不足以容纳所有的内容，那么，知道视窗显示区域的大小只是为使用者提供了在显示区域内卷动文字的第一步。如果您对其他有类似需求的 Windows 應用程式很熟悉，就很可能知道，这种情况下，我们需要使用「卷动列」。

卷动列

卷动列是图形使用者介面中最好的功能之一，它很容易使用，而且提供了很好的视觉回馈效果。您可以使用卷动列显示任何东西——无论是文字、图形、表格、资料库记录、图像或是网页，只要它所需的空間超出了视窗的显示区域所能提供的空间，就可以使用卷动列。

卷动列既有垂直方向的（供上下移动），也有水平方向的（供左右移动）。使用者可以使用滑鼠在卷动列两端的箭头上或者在箭头之间的区域中点一下，这时，「卷动方块」在卷动列内的移动位置与所显示的资讯在整个文件中的近似相关位置成比例。使用者也可以用滑鼠拖动卷动方块到特定的位置。图 4-5 显示了垂直卷动列的建议用法。

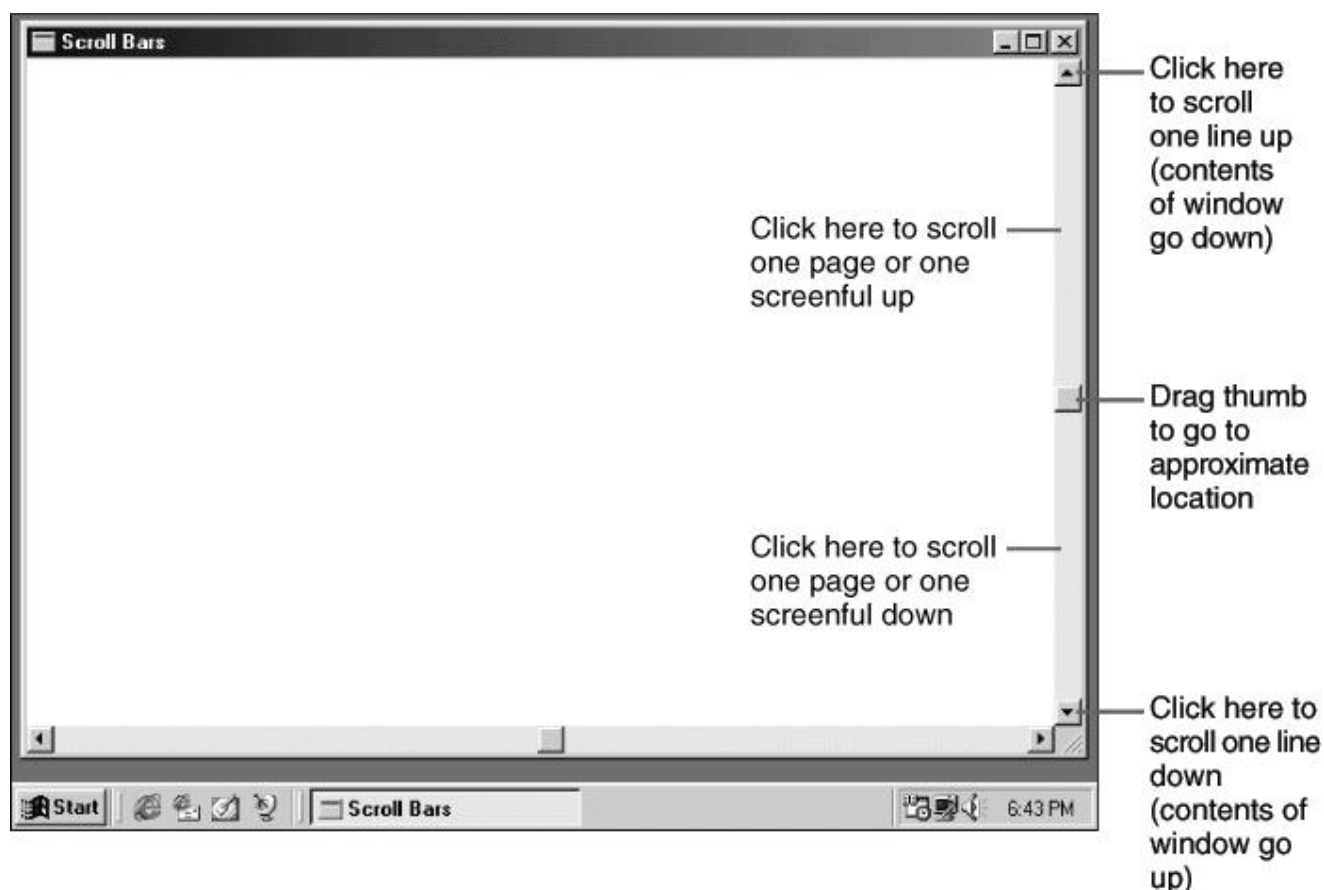


图 4-5 垂直卷动列

有时，程式写作者对卷动概念很难理解，因为他们的观点与使用者的观点不同：使用者向下卷动是想看到文件较下面的部分；但是，程式实际上是将文件相对於显示视窗向上移动。Windows 文件和表头档案识别字是依据使用者的观点：向上卷动意味著朝文件的开头移动；向下卷动意味著朝文件尾部移动。

很容易在应用程式中包含水平或者垂直的卷动列，程式写作者只需要在 `CreateWindow` 的第三个参数中包括视窗样式 (WS) 识别字 `WS_VSCROLL` (垂直卷动) 和/或 `WS_HSCROLL` (水平卷动) 即可。这些卷动列通常放在视窗的右部和底部，伸展为显示区域的整个长度或宽度。显示区域不包含卷动列所占据的空间。对於特定的显示驱动程式和显示解析度，垂直卷动列的宽度和水平卷动列的高度是恒定的。如果需要这些值，可以使用 `GetSystemMetrics` 呼叫来取得（如前面的程式那样）。

Windows 负责处理对卷动列的所有滑鼠操作，但是，视窗卷动列没有自动的键盘介面。如果想用游标键来完成卷动功能，则必须提供这方面的程式码（我们将在下一章另一个版本的 `SYSMETS` 程式中做到这一点）。

卷动列的范围和位置

每个卷动列均有一个相关的「范围」（这是一对整数，分别代表最小值和

最大值) 和「位置」(它是卷动方块在此范围内的位置)。当卷动方块在卷动列的顶部(或左部)时, 卷动方块的位置是范围的最小值; 在卷动列的底部(或右部)时, 卷动方块的位置是范围的最大值。

在内定情况下, 卷动列的范围是从 0 (顶部或左部) 至 100 (底部或右部), 但将范围改变为更方便於程式的数值也是很容易的:

```
SetScrollRange (hwnd, iBar, iMin, iMax, bRedraw) ;
```

参数 iBar 为 SB_VERT 或者 SB_HORZ, iMin 和 iMax 分别是范围的最小值和最大值。如果想要 Windows 根据新范围重画卷动列, 则设置 bRedraw 为 TRUE (如果在呼叫 SetScrollRange 後, 呼叫了影响卷动列位置的其他函式, 则应该将 bRedraw 设定为 FALSE 以避免过多的重画)。

卷动方块的位置总是离散的整数值。例如, 范围为 0 至 4 的卷动列具有 5 个卷动方块位置, 如图 4-6 所示。



图 4-6 具有 5 个卷动方块位置的卷动列

您可以使用 SetScrollPos 在卷动列范围内设置新的卷动方块位置:

```
SetScrollPos (hwnd, iBar, iPos, bRedraw) ;
```

参数 iPos 是新位置, 它必须在 iMin 至 iMax 的范围内。Windows 提供了类似的函式 (GetScrollRange 和 GetScrollPos) 来取得卷动列的目前范围和位置。

在程式内使用卷动列时, 程式写作者与 Windows 共同负责维护卷动列以及

更新卷动方块的位置。下面是 Windows 对卷动列的处理：

- 处理所有卷动列滑鼠事件
- 当使用者在卷动列内单击滑鼠时，提供一种「反相显示」的闪烁
- 当使用者在卷动列内拖动卷动方块时，移动卷动方块
- 为包含卷动列视窗的视窗讯息处理程式发送卷动列讯息

以下是程式写作者应该完成的工作：

- 初始化卷动列的范围和位置
- 处理视窗讯息处理程式的卷动列讯息
- 更新卷动列内卷动方块的位置
- 更改显示区域的内容以回应对卷动列的更改

像生活中的大多数事情一样，在我们看一些程式码时这些会显得更加有意义。

卷动列讯息

在用滑鼠单击卷动列或者拖动卷动方块时，Windows 给视窗讯息处理程式发送 WM_VSCROLL（供上下移动）和 WM_HSCROLL（供左右移动）讯息。在卷动列上的每个滑鼠动作都至少产生两个讯息，一条在按下滑鼠按钮时产生，一条在释放按钮时产生。

和所有的讯息一样，WM_VSCROLL 和 WM_HSCROLL 也带有 wParam 和 lParam 讯息参数。对于来自作为视窗的一部分而建立的卷动列讯息，您可以忽略 lParam；它只用于作为子视窗而建立的卷动列（通常在对话方块内）。

wParam 讯息参数被分为一个低字组和一个高字组。wParam 的低字组是一个数值，它指出了滑鼠对卷动列进行的操作。这个数值被看作一个「通知码」。通知码的值由以 SB（代表「scroll bar（卷动列）」）开头的识别字定义。以下是在 WINUSER.H 中定义的通知码：

```
#define SB_LINEUP          0
#define SB_LINELEFT       0
#define SB_LINEDOWN       1
#define SB_LINERIGHT      1
#define SB_PAGEUP         2
#define SB_PAGELLEFT      2
#define SB_PAGEDOWN       3
#define SB_PAGERIGHT      3
#define SB_THUMBPOSITION   4
#define SB_THUMBTRACK      5
#define SB_TOP            6
#define SB_LEFT            6
#define SB_BOTTOM         7
```



```
#define SB_RIGHT 7
#define SB_ENDSCROLL 8
```

包含 LEFT 和 RIGHT 的识别字用於水平卷动列, 包含 UP、DOWN、TOP 和 BOTTOM 的识别字用於垂直卷动列。滑鼠在卷动列的不同区域单击所产生的通知码如图 4-7 所示。

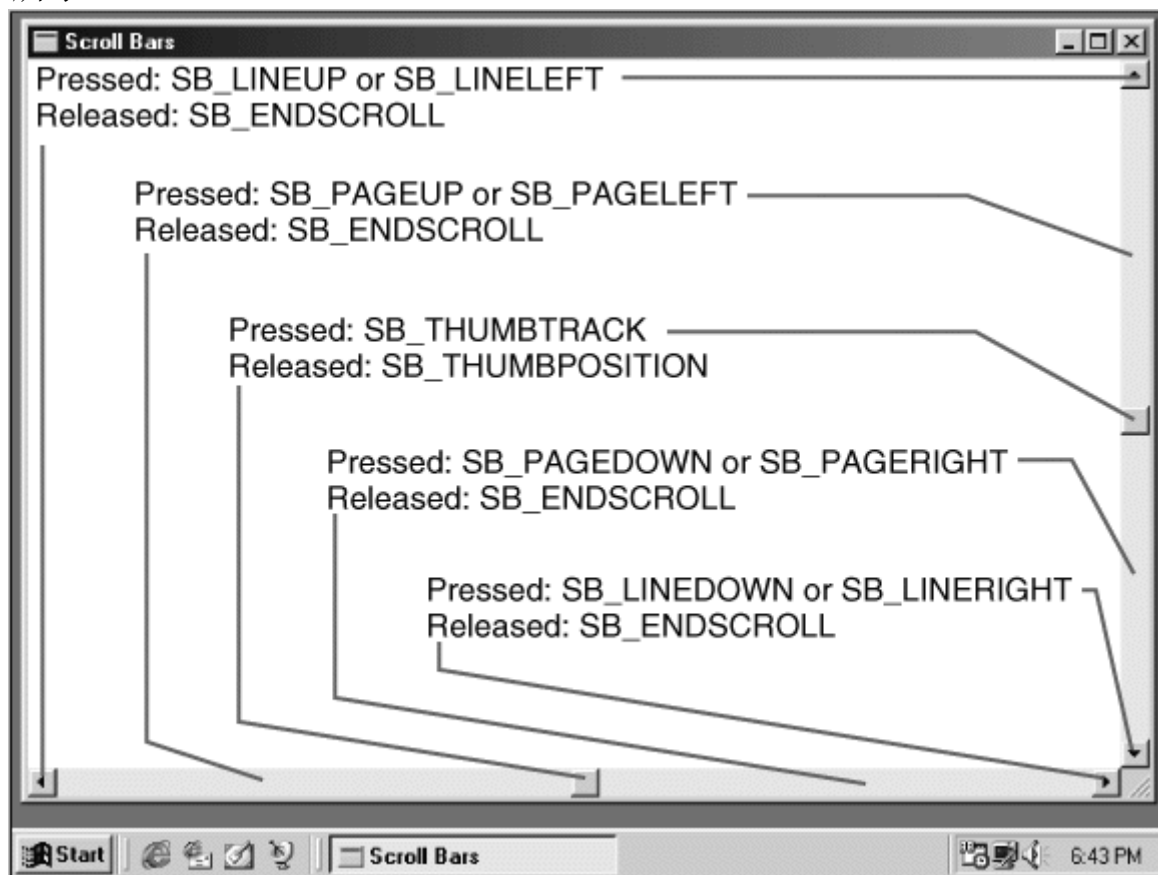


图 4-7 用於卷动列讯息的 wParam 值的识别字

如果在卷动列的各个部位按住滑鼠键, 程式就能收到多个卷动列讯息。当释放滑鼠键後, 程式会收到一个带有 SB_ENDSCROLL 通知码的讯息。一般可以忽略这个讯息, Windows 不会去改变卷动方块的位置, 而您可以在程式中呼叫 SetScrollPos 来改变卷动方块的位置。

当把滑鼠的游标放在卷动方块上并按住滑鼠键时, 您就可以移动卷动方块。这样就产生了带有 SB_THUMBTRACK 和 SB_THUMBPOSITION 通知码的卷动列讯息。在 wParam 的低字组是 SB_THUMBTRACK 时, wParam 的高字组是使用者在拖动卷动方块时的当前位置。该位置位於卷动列范围的最小值和最大值之间。在 wParam 的低字组是 SB_THUMBPOSITION 时, wParam 的高字组是使用者释放滑鼠键後卷动方块的最终位置。對於其他的卷动列操作, wParam 的高字组应该被忽略。

为了给使用者提供回馈, Windows 在您用滑鼠拖动卷动方块时移动它, 同时您的程式会收到 SB_THUMBTRACK 讯息。然而, 如果不通过呼叫 SetScrollPos 来处理 SB_THUMBTRACK 或 SB_THUMBPOSITION 讯息, 在使用者释放滑鼠键後, 卷动

方块会迅速跳回原来的位置。

程式能够处理 SB_THUMBTRACK 或 SB_THUMBPOSITION 讯息，但一般不同时处理两者。如果处理 SB_THUMBTRACK 讯息，在使用者拖动卷动方块时您需要移动显示区域的内容。而如果处理 SB_THUMBPOSITION 讯息，则只需在使用者停止拖动卷动方块时移动显示区域的内容。处理 SB_THUMBTRACK 讯息更好一些（但更困难），对于某些型态的资料，您的程式可能很难跟上产生的讯息。

WINUSER.H 表头档案还包括 SB_TOP、SB_BOTTOM、SB_LEFT 和 SB_RIGHT 通知码，指出卷动列已经被移到了它的最小或最大位置。然而，对于作为应用程式视窗一部分而建立的卷动列来说，永远不会接收到这些通知码。

在卷动列范围使用 32 位元的值也是有效的，尽管这不常见。然而，wParam 的高字组只有 16 位元的大小，它不能适当地指出 SB_THUMBTRACK 和 SB_THUMBPOSITION 操作的位置。在这种情况下，需要使用 GetScrollInfo 函式（在下面描述）来得到资讯。

在 SYSMETS 中加入卷动功能

前面的说明已经很详尽了，现在，要将那些东西动手做做看了。让我们开始时简单些，从垂直卷动著手，因为我们实在太需要垂直卷动了，而暂时还可以不用水平卷动。SYSMET2 如程式 4-3 所示。这个程式可能是卷动列的最简单的应用。

程式 4-3 SYSMETS2.C

```
/*-----
   SYSMETS2.C -- System Metrics Display Program No. 2
   (c) Charles Petzold, 1998
   -----*/
#include <windows.h>
#include "sysmets.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("SysMets2") ;
    HWND  hwnd ;
    MSG   msg ;
    WNDCLASS wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
```



```

    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
            szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 2"),
        WS_OVERLAPPEDWINDOW | WS_VSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxChar, cxCaps, cyChar, cyClient, iVscrollPos ;
    HDC        hdc ;
    int         i, y ;
    PAINTSTRUCT ps ;
    TCHAR       szBuffer[10] ;
    TEXTMETRIC  tm ;
    switch (message)
    {
case WM_CREATE:
        hdc = GetDC (hwnd) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        ReleaseDC (hwnd, hdc) ;
        SetScrollRange (hwnd, SB_VERT, 0, NUMLINES - 1, FALSE) ;
        SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;

```

```
        return 0 ;

case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
case SB_LINEUP:
    iVscrollPos -= 1 ;
    break ;

case SB_LINEDOWN:
    iVscrollPos += 1 ;
    break ;

case SB_PAGEUP:
    iVscrollPos -= cyClient / cyChar ;
    break ;

case SB_PAGEDOWN:
    iVscrollPos += cyClient / cyChar ;
    break ;

case SB_THUMBPOSITION:
    iVscrollPos = HIWORD (wParam) ;
    break ;

default :
    break ;
    }

iVscrollPos = max (0, min (iVscrollPos, NUMLINES - 1)) ;
if (iVscrollPos != GetScrollPos (hwnd, SB_VERT))
{
    SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
}

return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    for (i = 0 ; i < NUMLINES ; i++)
    {
        y = cyChar * (i - iVscrollPos) ;
        TextOut (hdc, 0, y,
            sysmetrics[i].szLabel,
            strlen (sysmetrics[i].szLabel)) ;
    }
```

```

        TextOut (hdc, 22 * cxCaps, y,
                sysmetrics[i].szDesc,
                lstrlen (sysmetrics[i].szDesc)) ;

        SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
        TextOut (hdc, 22 * cxCaps + 40 * cxChar, y, szBuffer,
                wsprintf (szBuffer, TEXT ("%5d"),
                        GetSystemMetrics (sysmetrics[i].iIndex))) ;
        SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}

return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

新的 CreateWindow 呼叫在第三个参数中包含了 WS_VSCROLL 视窗样式，从而在视窗中加入了垂直滚动列，其视窗样式为：

```
WS_OVERLAPPEDWINDOW | WS_VSCROLL
```

WndProc 视窗讯息处理程式在处理 WM_CREATE 讯息时增加了两条叙述，以设置垂直滚动列的范围和初始位置：

```
SetScrollRange (hwnd, SB_VERT, 0, NUMLINES - 1, FALSE) ;
SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
```

sysmetrics 结构具有 NUMLINES 行文字，所以滚动列范围被设定为 0 至 NUMLINES-1。滚动列的每个位置对应于在显示区域顶部显示的一个文字行。如果滚动方块的位置为 0，则第一行会被放置在显示区域的顶部。如果位置大于 0，其他行就会出现在显示区域的顶部。当位置为 NUMLINES-1 时，则最后一行文字出现在显示区域的顶部。

为了有助于处理 WM_VSCROLL 讯息，在视窗讯息处理程式中定义了一个静态变数 iVscrollPos，这一变数是滚动列内滚动方块的目前位置。对于 SB_LINEUP 和 SB_LINEDOWN，只需要将滚动方块调整一个单位的位置。对于 SB_PAGEUP 和 SB_PAGEDOWN，我们想移动一整面的内容，或者移动 cyClient /cyChar 个单位的位置。对于 SB_THUMBPOSITION，新的滚动方块位置是 wParam 的高字组。SB_ENDSCROLL 和 SB_THUMBTRACK 讯息被忽略。

在程式依据收到的 WM_VSCROLL 讯息计算出新的 iVscrollPos 值后，用 min 和 max 巨集来调整 iVscrollPos，以确保它在最大值与最小值之间。程式然后将 iVscrollPos 与呼叫 GetScrollPos 取得的先前位置相比较，如果滚动位置发生

了变化, 则使用 SetScrollPos 来进行更新, 并且呼叫 InvalidateRect 使整个视窗无效。

InvalidateRect 呼叫产生一个 WM_PAINT 讯息。SYSMETS1 在处理 WM_PAINT 讯息时, 每一行的 y 座标计算公式为:

```
cyChar * i
```

在 SYSMETS2 中, 计算公式为:

```
cyChar * (i - iVscrollPos)
```

回圈仍然显示 NUMLINES 行文字, 但是对於非零值的 iVscrollPos 是负数。程式实际上在显示区域以外显示这些文字行。当然, Windows 不会显示这些行, 因此萤幕显得乾净和漂亮。

前面说过, 我们一开始不想弄得太复杂, 这样的程式码很浪费, 效率很低。下面我们对此加以修改, 但是先要考虑在 WM_VSCROLL 讯息之後更新显示区域的方法。

绘图程式的组织

在处理完卷动列讯息後, SYSMETS2 不更新显示区域, 相反, 它呼叫 InvalidateRect 使显示区域失效。这导致 Windows 将一个 WM_PAINT 讯息放入讯息佇列中。

最好能使 Windows 程式在回应 WM_PAINT 讯息时完成所有的显示区域绘制功能。因为程式必须在一接收到 WM_PAINT 讯息时就更新整个显示区域, 如果在程式的其他部分也绘制的话, 将很可能使程式码重复。

首先, 您可能对这种拐弯抹角的方式感到厌烦。在 Windows 的早期, 因为这种方式与文字模式的程式设计差别太大, 程式写作者感到这种概念很难理解。并且, 程式要不断地通过马上绘制画面来回应键盘和滑鼠。这样做既方便又有效, 但是在很多情况下, 这完全不必要。当您掌握了在回应 WM_PAINT 讯息时积累绘制显示区域所需要的全部资讯的原则之後, 会对这种结果感到满意的。

如同 SYSMETS2 示范的, 程式仍然需要在处理非 WM_PAINT 讯息时更新特定的显示区域, 使用 InvalidateRect 就很方便, 您可以用它使显示区域内的特定矩形或者整个显示区域失效。

只将视窗显示区域标记为无效以产生 WM_PAINT 讯息, 对於某些应用程式来说也许不是完全令人满意的选择。在呼叫 InvalidateRect 之後, Windows 将 WM_PAINT 讯息放入讯息佇列中, 最後由视窗讯息处理程式处理它。然而, Windows 将 WM_PAINT 讯息当成低优先顺序讯息, 如果系统有许多其他的动作正在发生, 那么也许会让您等待一会儿工夫。这时, 当对话方块消失时, 将会出现一些空白的「洞」, 程式仍然等待更新它的视窗。

如果您希望立即更新无效区域，可以在呼叫 InvalidateRect 之後呼叫 UpdateWindow:

```
UpdateWindow (hwnd) ;
```

如果显示区域的任一部分无效，则 UpdateWindow 将导致 Windows 用 WM_PAINT 讯息呼叫视窗讯息处理程式（如果整个显示区域有效，则不呼叫视窗讯息处理程式）。这一 WM_PAINT 讯息不进入讯息伫列，直接由 Windows 呼叫视窗讯息处理程式。视窗讯息处理程式完成更新後立即退出，Windows 将控制传回给程式中 UpdateWindow 呼叫之後的叙述。

您可能注意到，UpdateWindow 与 WinMain 中用来产生第一个 WM_PAINT 讯息的函式相同。最初建立视窗时，整个显示区域内容变为无效，UpdateWindow 指示视窗讯息处理程式绘制显示区域。

建立更好的滚动

SYSMETS2 动作良好，但它只是模仿其他程式中的卷动列，并且效率很低。很快我将示范一个新的版本，改进它的不足。也许最有趣的是这个新版本不使用目前所讨论的四个卷动列函式。相反，它将使用 Win32 API 中才有的新函式。

卷动列资讯函式

卷动列文件（在/Platform SDK/User Interface Services/Controls/Scroll Bars 中）指出 SetScrollRange、SetScrollPos、GetScrollRange 和 GetScrollPos 函式是「过时的」，但这并不完全正确。这些函式在 Windows 1.0 中就出现了，在 Win32 API 中升级以处理 32 位元参数。它们仍然具有良好的功能。而且，它们不与 Windows 程式设计中新函式相冲突，这就是我在此书中仍使用它们的原因。

Win32 API 介绍的两个卷动列函式称作 SetScrollInfo 和 GetScrollInfo。这些函式可以完成以前函式的全部功能，并增加了两个新特性。

第一个功能涉及卷动方块的大小。您可能注意到，卷动方块大小在 SYSMETS2 程式中是固定的。然而，在您可能使用到的一些 Windows 應用程式中，卷动方块大小与在视窗中显示的文件大小成比例。显示的大小称作「页面大小」。演算法为：

$$\frac{\text{捲動方塊大小}}{\text{滾動長度}} \approx \frac{\text{頁面大小}}{\text{範圍}} \approx \frac{\text{顯示的文件數量}}{\text{文件的總大小}}$$

可以使用 SetScrollInfo 来设置页面大小（从而设置了卷动方块的大小），如将要看到的 SYSMETS3 程式所示。

GetScrollInfo 函式增加了第二个重要的功能,或者说它改进了目前 API 的不足。假设您要使用 65,536 或更大单位的范围,这在 16 位元 Windows 中是不可能的。当然在 Win32 中,函式被定义为可接受 32 位元参数,因此是没有问题的。(记住如果使用这样大的范围,滚动方块的实际物理位置数仍然由滚动列的图素大小限制)。然而,当使用 SB_THUMBTRACK 或 SB_THUMBPOSITION 通知码得到 WM_VSCROLL 或 WM_HSCROLL 讯息时,只提供了 16 位元资料来指出滚动方块的目前位置。通过 GetScrollInfo 函式可以取得真实的 32 位元值。

SetScrollInfo 和 GetScrollInfo 函式的语法是

```
SetScrollInfo (hwnd, iBar, &si, bRedraw) ;
GetScrollInfo (hwnd, iBar, &si) ;
```

像在其他滚动列函式中那样,iBar 参数是 SB_VERT 或 SB_HORZ,它还可以是用於滚动列控制的 SB_CTL。SetScrollInfo 的最後一个参数可以是 TRUE 或 FALSE,指出了是否要 Windows 重新绘制计算了新资讯後的滚动列。

两个函式的第三个参数是 SCROLLINFO 结构,定义为:

```
typedef struct tagSCROLLINFO
{
    UINT cbSize ;      // set to sizeof (SCROLLINFO)
    UINT fMask ;       // values to set or get
    int  nMin ;        // minimum range value
    int  nMax ;        // maximum range value
    UINT nPage ;       // page size
    int  nPos ;        // current position
    int  nTrackPos ;   // current tracking position
}
SCROLLINFO, * PSCROLLINFO ;
```

在程式中,可以定义如下的 SCROLLINFO 结构型态:

```
SCROLLINFO si ;
```

在呼叫 SetScrollInfo 或 GetScrollInfo 之前,必须将 cbSize 栏位设定为结构的大小:

```
si.cbSize = sizeof (si) ;
```

或

```
si.cbSize = sizeof (SCROLLINFO) ;
```

逐渐熟悉 Windows 後,您就会发现另外几个结构像这个结构一样,第一个栏位指出了结构大小。这个栏位使将来的 Windows 版本可以扩充结构并添加新的功能,并且仍然与以前编译的版本相容。

把 fMask 栏位设定为一个以上以 SIF 字首开头的旗标,并且可以使用 C 的位元操作 OR 运算符(|)组合这些旗标。

SetScrollInfo 函式使用 SIF_RANGE 旗标时,必须把 nMin 和 nMax 栏位设定为所需的滚动列范围。GetScrollInfo 函式使用 SIF_RANGE 旗标时,应把 nMin

和 nMax 栏位设定为从函式传回的目前范围。

SIF_POS 旗标也一样。当通过 SetScrollInfo 使用它时，必须把结构的 nPos 栏位设定为所需的位置。可以通过 GetScrollInfo 使用 SIF_POS 旗标来取得目前位置。

使用 SIF_PAGE 旗标能够取得页面大小。用 SetScrollInfo 函式把 nPage 设定为所需的页面大小。GetScrollInfo 使用 SIF_PAGE 旗标可以取得目前页面的大小。如果不想得到比例化的卷动列，就不要使用该旗标。

当处理带有 SB_THUMBTRACK 或 SB_THUMBPOSITION 通知码的 WM_VSCROLL 或 WM_HSCROLL 讯息时，通过 GetScrollInfo 只使用 SIF_TRACKPOS 旗标。从函式的传回中，SCROLLINFO 结构的 nTrackPos 栏位将指出目前的 32 位元的卷动方块位置。

在 SetScrollInfo 函式中仅使用 SIF_DISABLENOSCROLL 旗标。如果指定了此旗标，而且新的卷动列参数使卷动列消失，则该卷动列就不能使用了（下面会有更多的解释）。

SIF_ALL 旗标是 SIF_RANGE、SIF_POS、SIF_PAGE 和 SIF_TRACKPOS 的组合。在 WM_SIZE 讯息处理期间设置卷动列参数时，这是很方便的（在 SetScrollInfo 函式中指定 SIF_TRACKPOS 後，它会被忽略）。这在处理卷动列讯息时也是很方便的。

卷动范围

在 SYSMETS2 中，卷动范围设置最小为 0，最大为 NUMLINES-1。当卷动列位置是 0 时，第一行资讯显示在显示区域的顶部；当卷动列的位置是 NUMLINES-1 时，最後一行显示在显示区域的顶部，并且看不见其他行。

可以说 SYSMETS2 卷动范围太大。事实上只需把资讯最後一行显示在显示区域的底部而不是顶部即可。我们可以对 SYSMETS2 作出一些修改以达到此点。当处理 WM_CREATE 讯息时不设置卷动列范围，而是等到接收到 WM_SIZE 讯息後再做此工作：

```
iVscrollMax = max (0, NUMLINES - cyClient / cyChar) ;  
SetScrollRange (hwnd, SB_VERT, 0, iVscrollMax, TRUE) ;
```

假定 NUMLINES 等於 75，并假定特定视窗大小是：50(cyChar 除以 cyClient)。换句话说，我们有 75 行资讯但只有 50 行可以显示在显示区域中。使用上面的两行程式码，把范围设置最小为 0，最大为 25。当卷动列位置等於 0 时，程式显示 0 到 49 行。当卷动列位置等於 1 时，程式显示 1 到 50 行；并且当卷动列位置等於 25（最大值）时，程式显示 25 到 74 行。很明显需要对程式的其他部

分做出修改，但这是可行的。

新滚动列函式的一个好的功能是当使用与滚动列范围一样大的页面时，它已经为您做掉了一大堆杂事。可以像下面的程式码一样使用 SCROLLINFO 结构和 SetScrollInfo:

```
si.cbSize    = sizeof (SCROLLINFO) ;
si.cbMask    = SIF_RANGE | SIF_PAGE ;
si.nMin      = 0 ;
si.nMax      = NUMLINES - 1 ;
si.nPage     = cyClient / cyChar ;
SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
```

这样做之後，Windows 会把最大的滚动列位置限制为 $si.nMax - si.nPage + 1$ 而不是 $si.nMax$ 。像前面那样做出假设：NUMLINES 等於 75（所以 $si.nMax$ 等於 74）， $si.nPage$ 等於 50。这意味著最大的滚动列位置限制为 $74 - 50 + 1$ ，即 25。这正是我们想要的。

当页面大小与滚动列范围一样大时，会发生什么情况呢？在这个例子中，就是 $nPage$ 等於 75 或更大的情况。Windows 通常隐藏滚动列，因为它并不需要。如果不想隐藏滚动列，可在呼叫 SetScrollInfo 时使用 SIF_DISABLENOSCROLL，Windows 只是让那个滚动列不能被使用，而不隐藏它。

新 SYSMETS

SYSMETS3——此章中最後的 SYSMETS 程式版本——显示在程式 4-4 中。此版本使用 SetScrollInfo 和 GetScrollInfo 函式，添加左右卷动的水平滚动列，并能更有效地重画显示区域。

程式 4-4 SYSMETS3

```
SYSMETS3.C
/*-----
   SYSMETS3.C -- System Metrics Display Program No. 3
               (c) Charles Petzold, 1998
   -----*/
#include <windows.h>
#include "sysmets.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("SysMets3") ;
    HWND  hwnd ;
    MSG   msg ;
    WNDCLASS wndclass ;
```



```

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
            szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 3"),
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
    HDC        hdc ;
    int        i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
    PAINTSTRUCT ps ;
    SCROLLINFO si ;
    TCHAR      szBuffer[10] ;
    TEXTMETRIC tm ;

    switch (message)
    {
    case WM_CREATE:
        hdc = GetDC (hwnd) ;

```

```
GetTextMetrics (hdc, &tm) ;
cxChar = tm.tmAveCharWidth ;
cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
cyChar = tm.tmHeight + tm.tmExternalLeading ;

ReleaseDC (hwnd, hdc) ;
    // Save the width of the three columns
iMaxWidth = 40 * cxChar + 22 * cxCaps ;
return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    // Set vertical scroll bar range and page size
si.cbSize = sizeof (si) ;
si.fMask = SIF_RANGE | SIF_PAGE ;
si.nMin = 0 ;
si.nMax = NUMLINES - 1 ;
si.nPage = cyClient / cyChar ;
SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    // Set horizontal scroll bar range and page size
si.cbSize = sizeof (si) ;
si.fMask = SIF_RANGE | SIF_PAGE ;
si.nMin = 0 ;
si.nMax = 2 + iMaxWidth / cxChar ;
si.nPage = cxClient / cxChar ;
SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
return 0 ;

case WM_VSCROLL:
    // Get all the vertical scroll bar information
si.cbSize = sizeof (si) ;
si.fMask = SIF_ALL ;
GetScrollInfo (hwnd, SB_VERT, &si) ;
    // Save the position for comparison later on
iVertPos = si.nPos ;
switch (LOWORD (wParam))
{
case SB_TOP:
    si.nPos = si.nMin ;
    break ;

case SB_BOTTOM:
    si.nPos = si.nMax ;
    break ;

case SB_LINEUP:
```

```
        si.nPos -    = 1 ;
        break ;

case SB_LINEDOWN:
    si.nPos += 1 ;
    break ;

case SB_PAGEUP:
    si.nPos -= si.nPage ;
    break ;

case SB_PAGEDOWN:
    si.nPos += si.nPage ;
    break ;

case SB_THUMBTRACK:
    si.nPos = si.nTrackPos ;
    break ;

default:
    break ;
}

    // Set the position and then retrieve it. Due to adjustments
    // by Windows it may not be the same as the value set.

si.fMask = SIF_POS ;
SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
GetScrollInfo (hwnd, SB_VERT, &si) ;

    // If the position has changed, scroll the window and update it
if (si.nPos != iVertPos)
{
    ScrollWindow (    hwnd, 0, cyChar * (iVertPos - si.nPos),
                    NULL, NULL) ;

    UpdateWindow (hwnd) ;
}
return 0 ;
case WM_HSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;

    // Save the position for comparison later on
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos = si.nPos ;

    switch (LOWORD (wParam))
    {
```

```
case SB_LINELEFT:
    si.nPos -= 1 ;
    break ;

case SB_LINERIGHT:
    si.nPos += 1 ;
    break ;

case SB_PAGELEFT:
    si.nPos -= si.nPage ;
    break ;

case SB_PAGERIGHT:
    si.nPos += si.nPage ;
    break ;

case SB_THUMBPOSITION:
    si.nPos = si.nTrackPos ;
    break ;

    default :
    break ;
}

// Set the position and then retrieve it. Due to adjustments
// by Windows it may not be the same as the value set.

si.fMask = SIF_POS ;
SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
GetScrollInfo (hwnd, SB_HORZ, &si) ;

// If the position has changed, scroll the window

if (si.nPos != iHorzPos)
{
    ScrollWindow (    hwnd, cxChar * (iHorzPos - si.nPos), 0,
                    NULL, NULL) ;
}

return 0 ;
case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    // Get vertical scroll bar position
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_POS ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    iVertPos = si.nPos ;

    // Get horizontal scroll bar position
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
```

```

iHorzPos = si.nPos ;
    // Find painting limits
iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
iPaintEnd = min (NUMLINES - 1,
    iVertPos + ps.rcPaint.bottom / cyChar) ;

for (i = iPaintBeg ; i <= iPaintEnd ; i++)
{
    x = cxChar * (1 - iHorzPos) ;
    y = cyChar * (i - iVertPos) ;

    TextOut (hdc, x, y,
        sysmetrics[i].szLabel,
        lstrlen (sysmetrics[i].szLabel)) ;

    TextOut (hdc, x + 22 * cxCaps, y,
        sysmetrics[i].szDesc,
        lstrlen (sysmetrics[i].szDesc)) ;

    SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
    TextOut (hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
        wsprintf (szBuffer, TEXT ("%5d"),
            GetSystemMetrics (sysmetrics[i].iIndex))) ;

    SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
}

EndPaint (hwnd, &ps) ;
return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}

return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

这个版本的程式仰赖 Windows 保存卷动列资讯并做边界检查。在 WM_VSCROLL 和 WM_HSCROLL 处理的开始，它取得所有的卷动列资讯，根据通知码调整位置，然後呼叫 SetScrollInfo 设置其位置。程式然後呼叫 GetScrollInfo。如果该位置超出了 SetScrollInfo 呼叫的范围，则由 Windows 来纠正该位置并且在 GetScrollInfo 呼叫中传回正确的值。

SYSMETS3 使用 ScrollWindow 函式在视窗的显示区域中卷动资讯而不是重画它。虽然该函式很复杂（在新版本的 Windows 中已被更复杂的 ScrollWindowEx 所替代），SYSMETS3 仍以相当简单的方式使用它。函式的第二个参数给出了水

平卷动显示区域的数值，第三个参数是垂直卷动显示区域的数值，单位都是像素。

ScrollWindow 的最后两个参数设定为 NULL，这指出了要卷动整个显示区域。Windows 自动把显示区域中未被卷动操作覆盖的矩形设为无效。这会产生 WM_PAINT 讯息。再也不需要 InvalidateRect 了。注意 ScrollWindow 不是 GDI 函式，因为它不需装置内容代号。它是少数几个非 GDI 的 Windows 函式之一，它可以改变视窗的显示区域外观。很特殊但不方便，它是随卷动列函式一起记载在文件中。

WM_HSCROLL 处理拦截 SB_THUMBPOSITION 通知码并忽略 SB_THUMBTRACK。因而，如果使用者在水平卷动列上拖动卷动方块，在使用者释放滑鼠按钮之前，程式不会水平卷动视窗的内容。

WM_VSCROLL 的方法与之不同：程式拦截 SB_THUMBTRACK 讯息并忽略 SB_THUMBPOSITION。因而，程式随使用者在垂直卷动列上拖动卷动方块而垂直地滚动内容。这种想法很好，但应注意：一旦使用者发现程式会立即回应拖动的卷动方块，他们就会不断地来回拖动卷动方块。幸运的是现在的 PC 快得可以胜任这种严酷的测试。但是在较慢的机器上，可以考虑为 GetSystemMetrics 使用 SB_SLOWMACHINE 参数来替代这种处理。

加快 WM_PAINT 处理的一个方法由 SYSMETS3 展示：WM_PAINT 处理程式确定无效区域中的文字行并仅仅重画这些行。当然，程式码复杂一些，但速度很快。

不用滑鼠怎么办

在 Windows 的早期，有大量的使用者不喜欢使用滑鼠，而且，Windows 自身也不要求必须有滑鼠。虽然，没有滑鼠的 PC 现在走上了单色显示器和点阵印表机的没落之路，但我仍然建议您编写可以使用键盘来产生与滑鼠操作相同效果的程式，尤其对于像卷动列这样的基本操作物件更是如此。因为我们的键盘有一组游标移动键，所以应该实作同样的操作。

在下一章，您将学习使用键盘和在 SYSMETS3 中增加键盘介面的方法。您可能会注意到，SYSMETS3 似乎在通知码等於 SB_TOP 和 SB_BOTTOM 时处理了 WM_VSCROLL 讯息。前面已经提到过，视窗讯息处理程式不从卷动列接收这些讯息，所以，目前这是多余的程式码。当我们在下一章再次回到这个程式时，您将会明白这样做的原因。

第五章 图形基础

图形装置介面 (GDI: Graphics Device Interface) 是 Windows 的子系统, 它负责在视讯显示器和印表机上显示图形。正如您所认为的那样, GDI 是 Windows 非常重要的部分。不只您为 Windows 编写的应用系统在显示视觉资讯时使用 GDI, 就连 Windows 本身也使用 GDI 来显示使用者介面物件, 诸如功能表、卷动列、图示和滑鼠游标。

不幸的是, 如果要对 GDI 进行全面的讲述, 将需要一整本书——当然不是这本书。在本章中, 我只是想向您提供画线和填入区域的基本知识, 这對於理解下面几章的 GDI 已经足够了。在後面几章中会讲述 GDI 支援的点阵图、metafile 以及格式化文字。

GDI 的结构

从程式写作者的观点来看, GDI 由几百个函式呼叫和一些相关的资料型态、巨集和结构组成。但是在开始讲述这些函式的细节之前, 让我们先从巨观上了解一下 GDI 的整体结构。

GDI 原理

Windows 98 和 Microsoft Windows NT 中的图形主要由 GDI32.DLL 动态连结程式库输出的函式来处理。在 Windows 98 中, 这个 GDI32.DLL 实际是利用 16 位元 GDI.EXE 动态连结程式库来执行许多函式。在 Windows NT 中, GDI.EXE 只用於 16 位元的程式。

这些动态连结程式库呼叫您安装的视讯显示器和任何印表机呼叫驱动程序中的常式。视讯驱动程序存取视讯显示器的硬体, 印表机驱动程序将 GDI 命令转换为各种印表机能够理解的代码或者命令。显然, 不同的视讯显示卡和印表机要求不同的装置驱动程序。

因为 PC 相容机种上可以连接许多种不同的视讯设备, 所以, GDI 的主要目的之一是支援与装置无关的图形。Windows 程式应该能够毫无困难地在 Windows 支援的任意一种图形输出设备上执行, GDI 通过将您的程式和不同输出设备的特性隔离开来的方法来达到这一目的。

图形输出设备分为两大类: 位元映射设备和向量设备。大多数 PC 的输出设备是位元映射设备, 这意味著它们以图点构成的阵列来表示图像, 这类设备包括视讯显示卡、点阵印表机和雷射印表机。向量设备使用线来绘制图像, 通常

局限於绘图机。

许多传统的电脑图形程式设计方式都是完全以向量为主的，这意味著使用向量图形系统的程式与硬体有著一定层次的隔离。输出设备用图素表示图形，但是程式与程式介面之间并不是用图素进行沟通的。您当然可以使用 Windows GDI 作为一个高阶的向量绘制系统，同时也可以将它用於比较低阶的图素操作。

从这方面来看，Windows GDI 和传统的图形介面语言之间的关系，就如同 C 和其他程式设计语言之间的关系一样。C 以它在不同作业系统和环境之间的高度可携性而闻名，然而 C 也以允许程式写作者进行低阶系统呼叫而闻名，这些呼叫在其他高阶语言中通常是不可能的。正如 C 有时被认为是一种「高级组合语言」一样，您可以认为 GDI 是图形设备硬体之间的一种高阶介面。

您已经看到，Windows 内定使用图素座标系统。大多数传统的图形语言使用「虚拟」座标系，其水平和垂直轴的范围在 0 到 32,767 之间。虽然有些图形语言不让您使用图素座标，但是 Windows GDI 允许您使用两种座标系统之一（甚至依据实际度量衡的座标系）。您可以使用虚拟座标系以便让程式独立於硬体之外，或者也可以使用设备座标系而完全迎合硬体设备提供的环境。

某些程式写作者认为一旦开始使用操作图素的程式设计方式，就放弃了装置无关性。我们在上一章看到，这不完全是正确的，其中的诀窍是在与装置无关的方式中使用图素。这要求图形介面语言为程式提供一些方法来确定设备的硬体特徵，并进行适当的调节。例如，在 SYSMETS 程式中，我们根据标准系统字体字元的图素大小来确定萤幕上的文字间距，这种方法允许程式针对解析度、文字大小和方向比例各不相同的显示卡进行相应的调节。您将在本章看到一些用於确定显示尺寸的其他方法。

早期，许多使用者在单色显示器上执行 Windows。即使是几年前，笔记本电脑也还只有灰阶显示。为此，GDI 的设计保证了您可以在编写一个程式时不必太担心色彩问题——也就是说，Windows 可以将色彩转换为灰阶显示。甚至在今天，Windows 98 使用的视讯显示已经具有了不同的色彩能力（16 色、256 色、「high-Color」以及「true-color」）。虽然，彩色喷墨印表机的成本已经很低了，但是大多数使用者仍然坚持使用黑白印表机。盲目地使用这些设备是可以的，但是您的程式也应该能决定在某种显示设备上有多少色彩可以使用，从而最佳利用硬体功能。

当然，就如同您编写 C 程式时，为了使它在其他电脑上执行而遇到一些微妙的移植性问题一样，您也可能不小心让装置依赖性溜进您的 Windows 程式，这就是不与硬体完全隔离的代价。您还应该知道 Windows GDI 的局限。虽然可以在显示器上到处移动图形物件，但 GDI 通常是一个静态的显示系统，只有有

限的动画支援。如果需要为游戏编写复杂的动画，就应该研究一下 Microsoft DirectX，它提供了您需要的支援。

GDI 函式呼叫

组成 GDI 的几百个函式呼叫可以分为几大类：

取得（或者建立）和释放（或者清除）装置内容的函式 我们在前面的章节中已经看到过，您在绘图时需要装置内容代号。GetDC 和 ReleaseDC 函式让您在非 WM_PAINT 的讯息处理期间来做到这一点，而 BeginPaint 和 EndPaint 函式（虽然在技术上它们是 USER 模组而不是 GDI 模组的一部分）在进行绘图的 WM_PAINT 讯息处理期间使用。我们马上还会介绍有关装置内容的其他一些函式。

取得有关装置内容资讯的函式 再以第四章中 SYSMETS 程式为例，我们使用 GetTextMetrics 函式来取得有关装置内容中目前所选字体的尺寸资讯。在本章後面，我们将看到一个取得非常广泛的装置内容资讯的 >DEVCAPS1 程式。

绘图函式 显然，在所有前提条件都得以满足之後，这些函式是真正重要的部分。在上一章中，我们使用 TextOut 函式在视窗的显示区域显示一些文字。我们将看到，其他 GDI 函式还可以让您画线、填入区域。在第十四章和第十五章还会看到如何建立点阵图图像。

设定和取得装置内容参数的函式 装置内容的「属性」决定有关绘图函式如何工作的细节。例如，用 SetTextColor 来指定 TextOut（或者其他文字输出函式）所绘制的文字色彩。在第四章中 SYSMETS 程式中，我们使用 SetTextAlign 来告诉 GDI：TextOut 函式中的字串的开始位置应该在字串的右边而不是内定的左边。装置内容的所有属性都有预设值，取得装置内容时这些预设值就设定好了。对於所有的 Set 函式，都有相应的 Get 函式，以允许您取得目前装置内容属性。

使用 GDI 物件的函式 GDI 在这里变得有点混乱。首先举一个例子：内定时使用 GDI 绘制的所有直线都是实线并具有一个标准的宽度。您可能希望绘制更细的直线，或者是由一系列的点或短划线组成的直线。这种线的宽度和这种线的画笔样式不是装置内容的属性，而是一个「逻辑画笔」的特徵。您可以通过在 CreatePen、CreatePenIndirect 或 ExtCreatePen 函式中指定这些特徵来建立一个逻辑画笔，这些函式传回一个逻辑画笔的代号（虽然这些函式被认为是 GDI 的一部分，但是和大多数 GDI 函式呼叫不一样，它们不要求装置内容的代号）。要使用这个画笔，就要将画笔代号选进装置内容。我们认为，装置内容中目前选中的画笔就是装置内容的一个属性。这样，您画任何线都使用这个画笔，然後，您可以取消装置内容中的画笔选择，并清除画笔物件。清除画笔物件是必

要的，因为画笔定义占用了分配的记忆体空间。除了画笔以外，GDI 物件还用于建立填入封闭区域的画刷、字体、点阵图以及 GDI 的其他一些方面。

GDI 基本图形

您在萤幕或印表机上显示的图形型态本身可以被分为几类，通常被称为「基本图形」，它们是：

直线和曲线 线条是所有向量图形绘制系统的基础。GDI 支援直线、矩形、椭圆（包括椭圆的子集，也就是我们所说的「圆」）、椭圆圆周上的部分曲线即所谓的「弧」以及贝塞尔曲线 (Bezier spline)，我们将在本章中分别对它们进行介绍。所有更复杂的曲线可由折线 (polyline) 代替，折线通过一组非常短的直线来定义一条曲线。线条用装置内容中选中的目前画笔绘制。

填入区域 当一系列直线或者曲线封闭了一个区域时，该区域可以使用目前 GDI 画刷物件进行填图。这个画刷可以是实心色彩、图案（可以是一系列的水平、垂直或者对角标记）或者是在区域内垂直或者水平重复的点阵图图像。

点阵图 点阵图是位元的矩形阵列，这些位元对应于显示设备上的图素，它们是位元映射图形的基础工具。点阵图通常用于在视讯显示器或者印表机上显示复杂（一般都是真实的）图像。点阵图还可以用于显示必须快速绘制的小图像（诸如图示、滑鼠游标以及在应用工具条中出现的按钮等）。GDI 支援两种型态的点阵图——旧式的（虽然还非常有用）「装置相关」点阵图，是 GDI 物件；和新的（如 Windows 3.0 的）「装置无关」点阵图（或者 DIB），可以储存在磁片档案中。第十四章和第十五章讨论点阵图。

文字 文字的数学味道不像电脑图形的其他方面那样浓。文字和几百年的传统印刷术有关，它被许多印刷工人看作为一门艺术。因此，文字通常不仅是所有的电脑图形系统中最复杂的部分，而且（如果识字还是社会基本要求的话）也是最重要的部分。用于定义 GDI 字体物件和取得字体资讯的资料结构是 Windows 中最庞大的部分之一。从 Windows 3.1 开始，GDI 开始支援 TrueType 字体，该字体是在填入轮廓线基础上建立的，这样的填入轮廓线可由其他 GDI 函式处理。依据相容性和储存大小的考虑，Windows 98 继续支援旧式的点阵字体。我会在第十七章讨论字体。

其他部分

GDI 的其他部分无法这么容易地分类，它们是：

映射模式和变换 虽然内定以图素为单位进行绘图，但是您并非局限于此。GDI 映射模式允许您以英寸（或者甚至以几分之一英寸）、毫米或者任何您想使

用的单位来绘图 (Windows NT 还支援传统的以三乘三矩阵表示的「座标变换」, 这允许倾斜和旋转图形物件。不幸的是, 在 Windows 98 中不支援座标变换)。

Metafile Metafile 是以二进位形式储存的 GDI 命令集合。Metafile 主要用於通过剪贴板传输向量图形, 第十八章会讨论 metafile。

绘图区域 绘图区域是形状任意的复杂区域, 通常定义为较简单的绘图区域组合。在 GDI 内部, 绘图区域除了储存为最初用来定义绘图区域的线条组合以外, 还以一系列扫描线的形式储存。您可以将绘图区域用於绘制轮廓、填入图形和剪裁。

路径 路径是 GDI 内部储存的直线和曲线的集合。路径可以用於绘图、填入图形和剪裁, 还可以转换为绘图区域。

剪裁 绘图可以限制在显示区域的某一部分中, 这就是所谓的剪裁。剪裁区域是不是矩形都可以, 剪裁通常是通过区域或者路径来定义的。

调色盘 自订调色盘通常限於显示 256 色的显示器。Windows 仅保留这些色彩之中的 20 种以供系统使用, 您可以改变其他 236 种色彩, 以准确显示按点阵图形式储存的真实图像。第十六章会讨论调色盘。

列印 虽然本章限於讨论视讯显示, 但是您在本章中所学到的全部知识都适用於列印。第十三章会讨论列印。

装置内容

在开始绘图之前, 让我们比第四章更精确地讨论一下装置内容。

当您想在一个图形输出设备 (诸如萤幕或者印表机) 上绘图时, 您首先必须获得一个装置内容 (或者 DC) 的代号。将代号传回给程式时, Windows 就给了您使用设备的许可权。然後您在 GDI 函式中将这个代号作为一个参数, 向 Windows 标识您想在其上进行绘图的设备。

装置内容中包含许多确定 GDI 函式如何在设备上工作的目前「属性」, 这些属性允许传递给 GDI 函式的参数只包含起始座标或者尺寸资讯, 而不必包含 Windows 在设备上显示物件时需要的所有其他资讯。例如, 呼叫 TextOut 时, 您只需要在函式中给出装置内容代号、起始座标、文字和文字的长度。您不必指定字体、文字颜色、文字後面的背景色彩以及字元间距, 因为这些属性都是装置内容的一部分。当您想改变这些属性之一时, 您呼叫一个可以改变装置内容中属性的函式, 以後针对该装置内容的 TextOut 呼叫来使用改变後的属性。

取得装置内容代号

Windows 提供了几种取得装置内容代号的方法。如果在处理一个讯息时取得

了装置内容代号，应该在退出视窗函式之前释放它（或者删除它）。一旦释放了代号，它就不再有效了。对于印表机装置内容代号，规则就没有这么严格。在第十三章会讨论列印。

最常用的取得并释放装置内容代号的方法是，在处理 WM_PAINT 讯息时，使用 BeginPaint 和 EndPaint 呼叫：

```
hdc = BeginPaint (hwnd, &ps) ;
```

其他行程式

```
EndPaint (hwnd, &ps) ;
```

变数 ps 是型态为 PAINTSTRUCT 的结构，该结构的 hdc 栏位是 BeginPaint 传回的装置内容代号。PAINTSTRUCT 结构又包含一个名为 rcPaint 的 RECT（矩形）结构，rcPaint 定义一个包围视窗显示区域无效范围的矩形。使用从 BeginPaint 获得的装置内容代号，只能在这个区域内绘图。BeginPaint 呼叫使该区域有效。

Windows 程式还可以在处理非 WM_PAINT 讯息时取得装置内容代号：

```
hdc = GetDC (hwnd) ;
```

其他行程式

```
ReleaseDC (hwnd, hdc) ;
```

这个装置内容适用於视窗代号为 hwnd 的显示区域。这些呼叫与 BeginPaint 和 EndPaint 的组合之间的基本区别是，利用从 GetDC 传回的代号可以在整个显示区域上绘图。当然，GetDC 和 ReleaseDC 不使显示区域中任何可能的无效区域变成有效。

Windows 程式还可以取得适用於整个视窗（而不仅限于视窗的显示区域）的装置内容代号：

```
hdc = GetWindowDC (hwnd) ;
```

其他行程式

```
ReleaseDC (hwnd, hdc) ;
```

这个装置内容除了显示区域之外，还包括视窗的标题列、功能表、卷动列和框架（frame）。GetWindowDC 函式很少使用，如果想尝试用一用它，则必须拦截处理 WM_NCPAINT 讯息，Windows 使用该讯息在视窗的非显示区域上绘图。

BeginPaint、GetDC 和 GetWindowDC 获得的装置内容都与视讯显示器上的某个特定视窗相关。取得装置内容代号的另一个更通用的函式是 CreateDC：

```
hdc = CreateDC (pszDriver, pszDevice, pszOutput, pData) ;
```

其他行程式

```
DeleteDC (hdc) ;
```

例如，您可以通过下面的呼叫来取得整个萤幕的装置内容代号：

```
hdc = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
```

在视窗之外写入画面一般是不恰当的，但对于一些不同寻常的应用程式来

说,这样做很方便(您还可通过在呼叫 GetDC 时使用一个 NULL 参数,从而取得整个萤幕的装置内容代号,不过这在文件中已经提到了)。在第十三章中,我们将使用 CreateDC 函式来取得一个印表机装置内容代号。

有时您只是需要取得关于某装置内容的一些资讯而并不进行任何绘画,在这种情况下,您可以使用 CreateIC 来取得一个「资讯内容」的代号,其参数与 CreateDC 函式相同,例如:

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
```

您不能用这个资讯内容代号往设备上写东西。

使用点阵图时,取得一个「记忆体装置内容」有时是有用的:

```
hdcMem = CreateCompatibleDC (hdc) ;
```

其他行程式

```
DeleteDC (hdcMem) ;
```

您可以将点阵图选进记忆体装置内容,然后使用 GDI 函式在点阵图上绘画。我将在第十四章讨论这些技术。

前面已经提到过,metafile 是一些 GDI 呼叫的集合,以二进位形式编码。您可以通过取得 metafile 装置内容来建立 metafile:

```
hdcMeta = CreateMetaFile (pszFilename) ;
```

其他行程式

```
hmf = CloseMetaFile (hdcMeta) ;
```

在 metafile 装置内容有效期间,任何用 hdcMeta 所做的 GDI 呼叫都变成 metafile 的一部分而不会显示。在呼叫 CloseMetaFile 之后,装置内容代号变为无效,函式传回一个指向 metafile (hmf) 的代号。我会在第十八章讨论 metafile。

取得装置内容资讯

一个装置内容通常是指一个实际显示设备,如视讯显示器和印表机。通常,您需要取得有关该设备的资讯,包括显示器的大小(单位为图素或者实际长度单位)和色彩显示能力。您可以通过呼叫 GetDeviceCaps (「取得设备功能」) 函式来取得这些资讯:

```
iValue = GetDeviceCaps (hdc, iIndex) ;
```

其中,参数 iIndex 取值为 WINGDI.H 表头档案中定义的 29 个识别字之一。例如,iIndex 为 HORZRES 时将使 GetDeviceCaps 传回设备的宽度(单位为图素);iIndex 为 VERTRES 时将让 GetDeviceCaps 传回设备的高度(单位为图素)。如果 hdc 是印表机装置内容的代号,则 GetDeviceCaps 传回印表机显示区域的高度和宽度,它们也是以图素为单位的。

还可以使用 GetDeviceCaps 来确定设备处理不同型态图形的能力,这对于

视讯显示器并不很重要，但是对於列印设备却是非常重要。例如，大多数绘图机不能画点阵图图像，GetDeviceCaps 就可以将这一情况告诉您。

DEVCAPS1 程式

程式 5-1 所示的 DEVCAPS1 程式显示了以一个视讯显示器的装置内容为参数时，可以从 GetDeviceCaps 函式中获得的部分资讯（该程式的另一个扩充版本 DEVCAPS2 将在第十三章给出，用於取得印表机资讯）。

程式 5-1 DEVCAPS1

```
DEVCAPS1.C
/*-----
   DEVCAPS1.C -- Device Capabilities Display Program No. 1
               (c) Charles Petzold, 1998
   -----*/
#include <windows.h>
#define NUMLINES ((int) (sizeof devcaps / sizeof devcaps [0]))
struct
{
    int    iIndex ;
    TCHAR *  szLabel ;
    TCHAR *  szDesc ;
}
devcaps [] =
{
    HORZSIZE,  TEXT ("HORZSIZE"),TEXT ("Width in millimeters:"),
    VERTSIZE,  TEXT ("VERTSIZE"),TEXT ("Height in millimeters:"),
    HORZRES,   TEXT ("HORZRES"),TEXT ("Width in pixels:"),
    VERTRES,   TEXT ("VERTRES"),TEXT ("Height in raster lines:"),
    BITSPIXEL, TEXT ("BITSPIXEL"),TEXT ("Color bits per pixel:"),
    PLANES,    TEXT ("PLANES"),  TEXT ("Number of color planes:"),
    NUMBRUSHES,TEXT ("NUMBRUSHES"), TEXT ("Number of device brushes:"),
    NUMPENS,   TEXT ("NUMPENS"),  TEXT ("Number of device pens:"),
    NUMMARKERS,TEXT ("NUMMARKERS"), TEXT ("Number of device markers:"),
    NUMFONTS,  TEXT ("NUMFONTS"), TEXT ("Number of device fonts:"),
    NUMCOLORS, TEXT ("NUMCOLORS"), TEXT ("Number of device colors:"),
    PDEVICESIZE, TEXT ("PDEVICESIZE"), TEXT ("Size of device
structure:"),
    ASPECTX,   TEXT ("ASPECTX"), TEXT ("Relative width of pixel:"),
    ASPECTY,   TEXT ("ASPECTY"),TEXT ("Relative height of pixel:"),
    ASPECTXY,  TEXT ("ASPECTXY"), TEXT ("Relative diagonal of pixel:"),
    LOGPIXELSX,TEXT ("LOGPIXELSX"), TEXT ("Horizontal dots per inch:"),
    LOGPIXELSY,TEXT ("LOGPIXELSY"), TEXT ("Vertical dots per inch:"),
    SIZEPALETTE, TEXT ("SIZEPALETTE"), TEXT ("Number of palette
entries:"),
    NUMRESERVED, TEXT ("NUMRESERVED"), TEXT ("Reserved palette
entries:"),
```

```

        COLORRES, TEXT ("COLORRES"), TEXT ("Actual color resolution:")
    } ;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("DevCaps1") ;
    HWND        hwnd ;
    MSG         msg ;
    WNDCLASS    wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Device Capabilities"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

```

```
{
    static int  cxChar, cxCaps, cyChar ;
    TCHAR      szBuffer[10] ;
    HDC        hdc ;
    int        i ;
    PAINTSTRUCT ps ;
    TEXTMETRIC tm ;

    switch (message)
    {
    case WM_CREATE:
        hdc = GetDC (hwnd) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar      = tm.tmAveCharWidth ;
        cxCaps      = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
        cyChar      = tm.tmHeight + tm.tmExternalLeading ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        for (i = 0 ; i < NUMLINES ; i++)
        {
            TextOut (  hdc, 0, cyChar * i,
                      devcaps[i].szLabel,
                      lstrlen (devcaps[i].szLabel)) ;

            TextOut (  hdc, 14 * cxCaps, cyChar * i,
                      devcaps[i].szDesc,
                      lstrlen (devcaps[i].szDesc)) ;

            SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
            TextOut (hdc, 14*cxCaps+35*cxChar, cyChar*i, szBuffer,
                    wsprintf (szBuffer, TEXT ("%5d"),
                               GetDeviceCaps (hdc, devcaps[i].iIndex))) ;

            SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```


}

可以看到，这个程式非常类似第四章的 SYSMETS1。为了保持程式码的短小，我没有使用滚动列，因为我知道资讯可以在一个画面上显示出来。在 256 色，640 480 的 VGA 上显示的结果如图 5-1 所示。

HORZSIZE	Width in millimeters:	169
VERTSIZE	Height in millimeters:	127
HORZRES	Width in pixels:	640
VERTRES	Height in raster lines:	480
BITSPIXEL	Color bits per pixel:	8
PLANES	Number of color planes:	1
NUMBRUSHES	Number of device brushes:	-1
NUMPENS	Number of device pens:	16
NUMMARKERS	Number of device markers:	0
NUMFONTS	Number of device fonts:	0
NUMCOLORS	Number of device colors:	20
PDEVICESIZE	Size of device structure:	1112
ASPECTX	Relative width of pixel:	36
ASPECTY	Relative height of pixel:	36
ASPECTXY	Relative diagonal of pixel:	51
LOGPIXELSX	Horizontal dots per inch:	96
LOGPIXELSY	Vertical dots per inch:	96
SIZEPALETTE	Number of palette entries:	256
NUMRESERVED	Reserved palette entries:	20
COLORRES	Actual color resolution:	18

图 5-1 256 色，640×480VGA 上的 DEVCAPS1 显示

装置的大小

假定要绘制边长为 1 英寸的正方形，您（程式写作者）或 Windows（作业系统）需要知道视讯显示上 1 英寸对应多少图素。使用 GetDeviceCaps 函式能取得有关如视讯显示器和印表机之类输出设备的实际显示大小资讯。

视讯显示器和印表机是两个不同的设备。但也许最不明显的区别是「解析度」与装置联系起来的方式。对于印表机，我们经常用「每英寸的点数 (dpi)」表示解析度。例如，大多数雷射印表机有 300 或 600dpi 的解析度。然而，视讯显示器的解析度是以水平和垂直的总图素数来表示的，例如，1024 768。大多数人不会告诉您他的印表机在一张纸上水平和垂直列印多少图素或他们的视讯显示器上每英寸有多少图素。

在本书中，我用「解析度」来严格定义每度量单位（一般为英寸）内的图素数。我使用「图素大小」或「图素尺寸」表示设备水平或垂直显示的总图素数。「度量大小」或「度量尺寸」是以英寸或毫米为单位的设备显示区域的大小。（对于印表机页面，它不是整个页面，只是可列印的区域。）图素大小除

以度量大小就得到解析度。

现在 Windows 使用的大多数视讯显示器的萤幕都是宽比高多 33%。这就表示纵横比为 1.33:1 或（一般写法）4:3。历史上，该比例可追溯到 Thomas Edison 制作电影的年代。它一直作为电影的标准纵横比，直到 1953 年出现各种型态的宽银幕投影机。电视机萤幕的纵横比也是 4:3。

然而，Windows 应用程式不应假设视讯显示器具有 4:3 的纵横比。人们进行文字处理时希望视讯显示器与一张纸的长和宽类似。最普通的选择是把 4:3 变为 3:4 显示，把标准显示翻转一下。

如果设备的水平解析度与垂直解析度相等，就称设备具有「正方形图素」。现在，Windows 普遍使用的视讯显示器都具有正方形图素，但也有例外。（应用程式也不应假设视讯显示器总是具有正方形图素。）Windows 第一次发表时，标准显示卡卡是 IBM Color Graphics Adapter (CGA)，它有 640 200 的图素大小；Enhanced Graphics Adapter (EGA) 有 640 350 的图素大小；Hercules Graphics Card 有 720 348 的图素大小。所有这些显示卡都使用 4:3 纵横比的显示器，但是水平和垂直图素数的比值都不是 4:3。

执行 Windows 的使用者很容易确定视讯显示器的图素大小。在「控制台」中执行「显示器」，并选择「设定」页面标签。在标有「桌面区域」的栏位中，可以看到这些图素尺寸之一：

- 640×480 图素
- 800×600 图素
- 1024×768 图素
- 1280×1024 图素
- 1600×1200 图素

所有这些都是 4:3。（除了 1280 1024 图素大小。这不但有些不好，还有些令人反感。所有这些图素尺寸都认为在 4:3 的显示器上会产生正方形的图素。）

Windows 应用程式可以使用 SM_CXSCREEN 和 SM_CYSCREEN 参数从 GetSystemMetrics 得到图素尺寸。从 DEVCAPS1 程式中您会注意到，程式可以用 HORZRES（水平解析度）和 VERTRES 参数从 GetDeviceCaps 中得到同样的值。这里「解析度」指的是图素大小而不是每度量单位的图素数。

这些是设备大小的简单部分，现在开始复杂的部分。

前两个设备能力，HORZSIZE 和 VERTSIZE，文件中称为「以毫米计的实际萤幕的宽度」及「以毫米计的实际萤幕的高度」（在 Platform SDK/Graphics 和 Multimedia Services/GDI/Device Contexts/Device Context Reference/Device Context Functions/GetDeviceCaps 中）。这些看起来更像

直接的定义。例如，给出视讯显示卡和显示器的介面特性，Windows 如何真正知道显示器的大小呢？如果您有台膝上型电脑（它的视讯驱动程式能知道准确的萤幕大小）并且连接了外部显示器，又是哪种情况呢？如果把视讯投影机连接到电脑上呢？

在 Windows 的 16 位元版本中（及在 Windows NT 中），Windows 为 HORZSIZE 和 VERTSIZE 使用「标准」的显示大小。然而，从 Windows 95 开始，HORZSIZE 和 VERTSIZE 值是从 HORZRES、VERTRES、LOGPIXELSX 和 LOGPIXELSY 值中衍生出来的。这是它的工作方式。

当您在「控制台」中使用「显示器」程式选择显示的图素大小时，也可以选择系统字体的大小。这个选项的原因是用於 640 480 显示的字体在提升到 1024 768 或更大时字太小，而您可能想要更大的系统字体。这些系统字体大小指「显示器」程式的「设定」页面标签中的「小字体」和「大字体」。

在传统的排版中，字体的字母大小由「点」表示。1 点大约 1/72 英寸，在电脑排版中 1 点正好为 1/72 英寸。

理论上，字体的点值是从字体中最高的字元顶部到例如 j、p、q 和 y 等字母下部的字元底部的距离，其中不包括重音符号。例如，在 10 点的字体中此距离是 10/72 英寸。根据 TEXTMETRIC 结构，字体的点值等於 tmHeight 栏位减去 tmInternalLeading 栏位，如图 5-2 所示（该图与上一章的图 4-3 一样）。

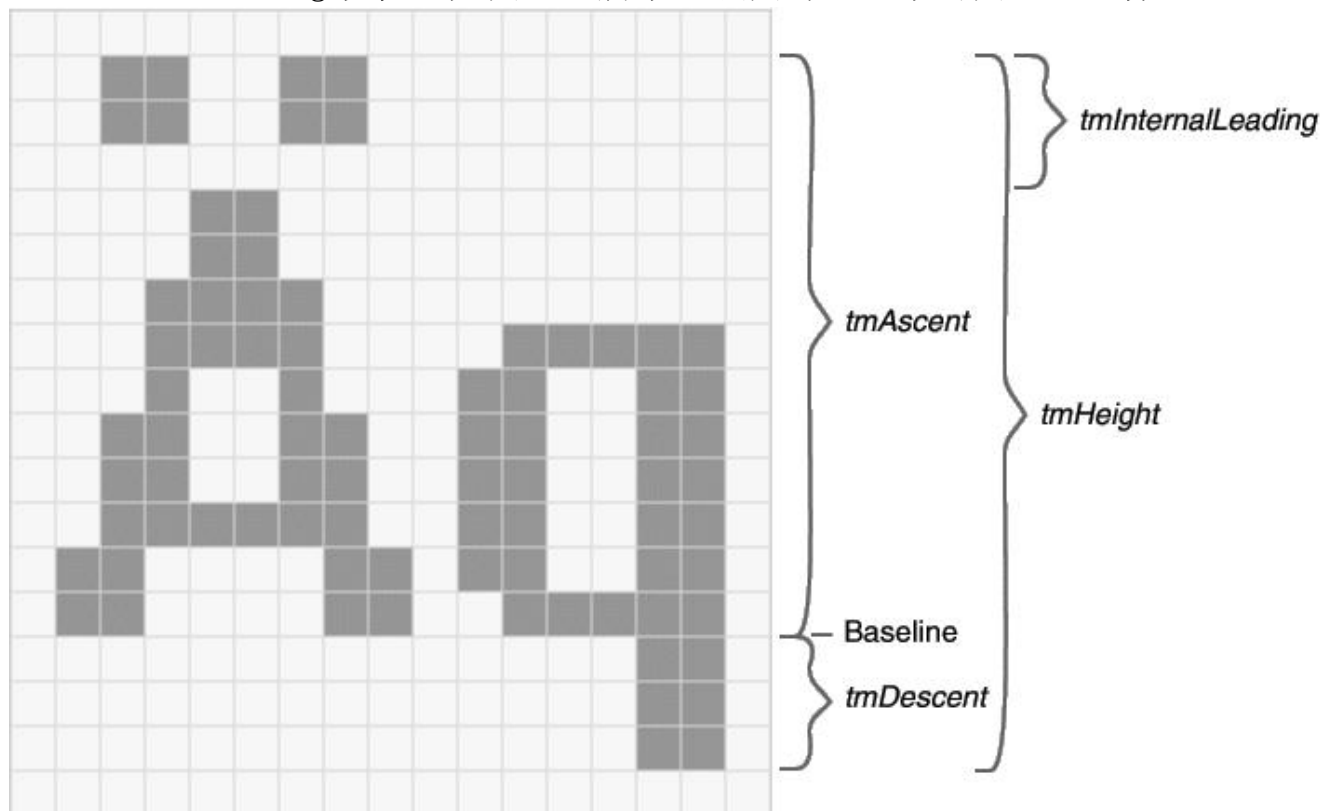


图 5-2 小字体和 TEXTMETRIC 栏位。

在真正的排版中，字体的点值与字体字母的实际大小并不正好相等。字体

的设计者做出的实际字元比点值指示的要大一些或小一些。毕竟，字体设计是一种艺术而不是科学。

TEXTMETRIC 结构的 `tmHeight` 栏位指出文字的连续行在萤幕或印表机上间隔的方式。这也可以用点来测量。例如，12 点的行距指出文字连续行的基准线应该间隔 $12/72$ （或 $1/6$ ）英寸。不应该为 10 点字体使用 10 点行距，因为文字的连续行会碰到一起。

10 点字体读起来很舒服。小于 10 点的字体不益於长时间阅读。

Windows 系统字体——不考虑是大字体还是小字体，也不考虑所选择的视频图素大小——固定假设为 10 点字体和 12 点行距。这听起来很奇怪，如果字体都是 10 点，为什么还把它们称为大字体和小字体呢？

解答是：**当您在「控制台」的「显示」程式上选择小字体或大字体时，实际上是选择了一个假定的视讯显示解析度，单位是每英寸的点数**。当选择小字体时，即要 Windows 假定视讯显示解析度为每英寸 96 点。当选择大字体时，即要 Windows 假定视讯显示解析度为每英寸 120 点。

再看看图 5-2。那是小字体，它依据的显示解析度为每英寸 96 点。我说过它是 10 点字体。10 点即是 $10/72$ 英寸，如果乘以 96 点，每英寸大概就为 13 图素。这即是 `tmHeight` 减去 `tmInternalLeading` 的值。行距是 12 点，或 $12/72$ 英寸，它乘以 96 点，每英寸就为 16 图素。这即是 `tmHeight` 的值。

图 5-3 显示大字体。这是依据每英寸 120 点的解析度。同样，它是 10 点字体， $10/72$ 乘以 120 点，每英寸等於 16 图素，即是 `tmHeight` 减 `tmInternalLeading` 的值。12 点行距等於 20 图素，即是 `tmHeight` 的值。（像第四章一样，**再次强调所显示的是实际的度量大小，因此您可以理解它工作的方式。不要在您的程式中对此写作程式。**）

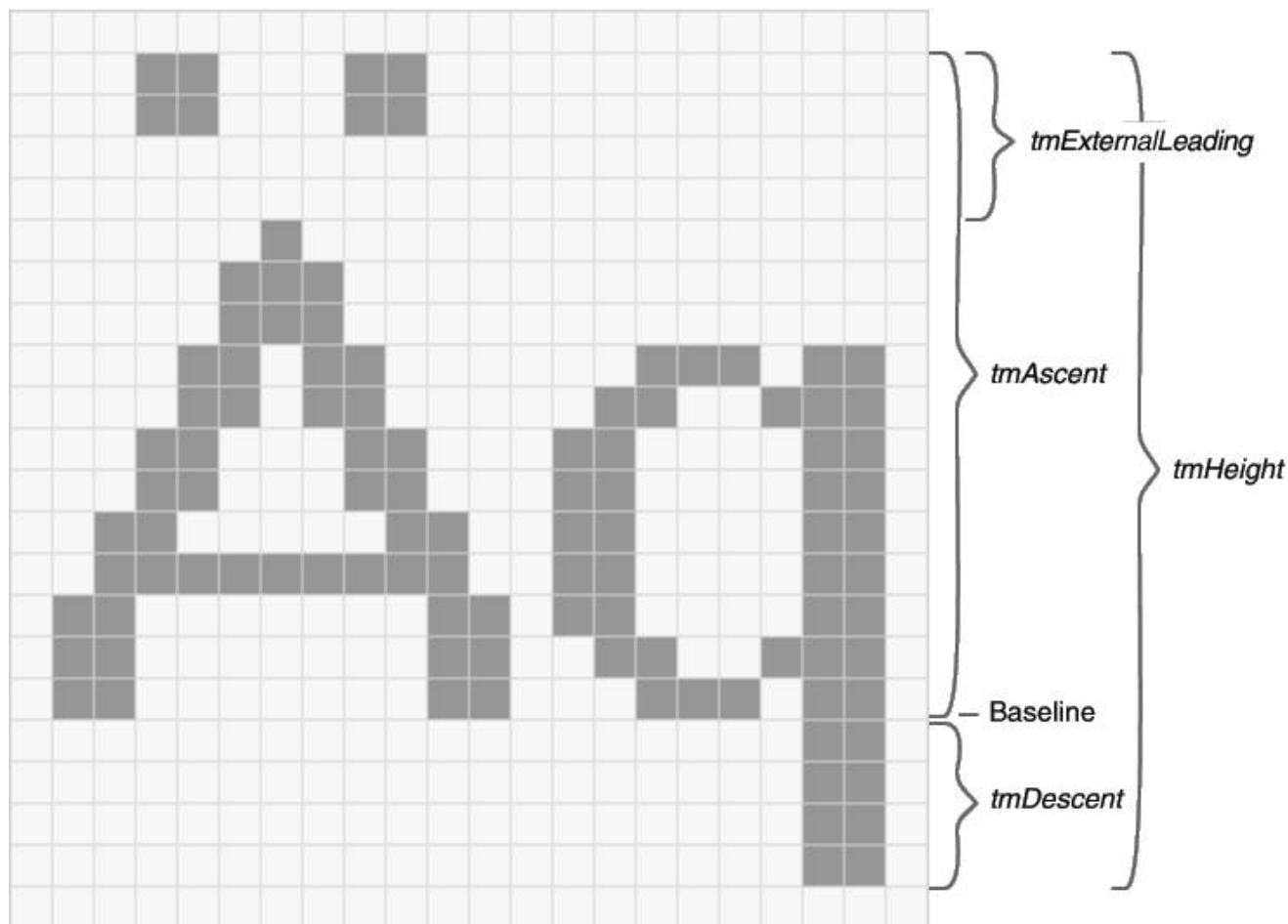


图 5-3 大字体和 FONTMETRIC 栏位

在 Windows 程式中，您可以使用 `GetDeviceCaps` 函式取得使用者在「控制台」的「显示器」程式中选择的以每英寸的点数为单位的假定解析度。要得到这些值（如果视讯显示器不具有正方形图素，在理论上这些值是不同的），可以使用索引 `LOGPIXELSX` 和 `LOGPIXELSY`。`LOGPIXELS` 指逻辑图素，它的基本意思是「以每英寸的图素数为单位的非实际解析度」。

用 `HORZSIZE` 和 `VERTSIZE` 索引从 `GetDeviceCaps` 得到的设备能力，在文件上称为「实际萤幕的宽度，单位毫米」及「实际萤幕的高度，单位毫米」。因为这些值是从 `HORZRES`、`VERTRES`、`LOGPIXELSX` 和 `LOGPIXELSY` 值中衍生出来的，所以它们应该称为「逻辑宽度」和「逻辑高度」。公式是：

$$\text{水平大小(mm)} = 25.4 \times \frac{\text{水平解析度(图素)}}{\text{逻辑图素X(每英寸的点数)}}$$

$$\text{垂直大小(mm)} = 25.4 \times \frac{\text{垂直解析度(图素)}}{\text{逻辑图素Y(每英寸的点数)}}$$

常数 25.4 用于把英寸转变为毫米。

这看起来是种不合逻辑的退步。毕竟，视讯显示器是可以用品以毫米为单位的大小（至少是近似的）衡量的。但是 Windows 98 并不关心这个大小。相反，

它以使用者选择的显示图素大小和系统字体大小为基础计算以毫米为单位的显示大小。更改显示的图素大小并根据 `GetDeviceCaps` 更改度量大小。这有什么意义呢？

这非常有意义。假定有一个 17 英寸的显示器。实际的显示大小大约是 12 英寸乘 9 英寸。假定在最小要求的 640 480 图素大小下执行 Windows。这意味著实际的解析度是每英寸 53 点。10 点字体（在纸上便於阅读）在萤幕上从 A 的顶部到 q 的底部只有 7 个图素。这样的字体很难看而且不易读。（可问问那些在旧的 Color Graphics Adapter 上执行 Windows 的人们。）

现在，把您的电脑接上视讯投影机。投影的视讯显示器是 4 英尺宽，3 英尺高。同样的 640 480 图素大小现在是大约每英寸 13 点的解析度。在这种条件下试图显示 10 点的字体是很可笑的。

10 点字体在视讯显示器上应是可读的，因为它在列印时是肯定可读的。所以 10 点字体就成为一个重要的参照。当 Windows 應用程式确保 10 点萤幕字体为平均大小时，就能够使用 8 点字体显示较小的文字（仍可读），或用大於 10 点的字体显示较大的文字。因而，视频解析度（以每英寸的点数为单位）由 10 点字体的图素大小来确定是很有意义的。

然而，在 Windows NT 中，用老的方法定义 `HORZSIZE` 和 `VERTSIZE` 值。这种方法与 Windows 的 16 位元版本一致。`HORZRES` 和 `VERTRES` 值仍然表示水平和垂直图素的数值，`LOGPIXELSX` 和 `LOGPIXELSY` 仍然与在「控制台」的「显示器」程式中选择的字体有关。在 Windows 98 中，`LOGPIXELSX` 和 `LOGPIXELSY` 的典型值是 96 和 120 dpi，这取决於您选择的是小字体还是大字体。

在 Windows NT 中的区别是 `HORZSIZE` 和 `VERTSIZE` 值固定表示标准显示器大小。对於普通的显示卡，取得的 `HORZSIZE` 和 `VERTSIZE` 值分别是 320 和 240 毫米。这些值是相同的，与选择的图素大小无关。因此，这些值与用 `HORZRES`、`VERTRES`、`LOGPIXELSX` 和 `LOGPIXELSY` 索引从 `GetDeviceCaps` 中得到的值不同。然而，可以用前面的公式计算在 Windows 98 下的 `HORZSIZE` 和 `VERTSIZE` 值。

如果程式需要实际的视讯显示大小该怎么办？也许最好的解决方法是用对话方块让使用者输入它们。

最後，来自 `GetDeviceCaps` 的另三个值与视讯大小有关。`ASPECTX`、`ASPECTY` 和 `ASPECTXY` 值是每一个图素的相对宽度、高度和对角线大小，四舍五入到整数。对於正方形图素，`ASPECTX` 和 `ASPECTY` 值相同。无论如何，`ASPECTXY` 值应等於 `ASPECTX` 与 `ASPECTY` 平方和的平方根，就像直角三角形一样。

关于色彩

如果视讯显示卡仅显示黑色图素和白色图素，则每个图素只需要记忆体中的一位元。彩色显示器中每个图素需要多个位元。位元数越多，色彩越多，或者更具体地说，可以同时显示的不同色彩的数目等於 2 的位元数次方。

「Full-Color」视讯显示器的解析度是每个图素 24 位元——8 位元红色、8 位元绿色以及 8 位元蓝色。红、绿、蓝即「色光三原色」。混合这三种基本颜色可以生成许多其他的颜色，您通过放大镜看显示幕，就可以看出来。

「High-Color」显示解析度是每个图素 16 位元——5 位元红色、6 位元绿色以及 5 位元蓝色。绿色多一位元是因为人眼对绿色更敏感一些。

显示 256 种颜色的显示卡每个图素需要 8 位元。然而，这些 8 位元的值一般由定义实际颜色的调色盘组织的。我会在第十六章详细地讨论它们。

最後，显示 16 种颜色的显示卡每个图素需要 4 位元。这 16 种颜色一般固定分为暗的或亮的红、黑、蓝、青、紫、黄、两种灰色。这 16 种颜色要回溯到老式的 IBM CGA。

只有在某些怪异的程式中才需要知道视讯显示卡上的记忆体是如何组织的，但是 GetDeviceCaps 使程式写作者可以知道显示卡的储存组织以及它能够表示的色彩数目，下面的呼叫传回色彩平面的数目：

```
iPlanes = GetDeviceCaps (hdc, PLANES) ;
```

下面的呼叫传回每个图素的色彩位元数：

```
iBitsPixel = GetDeviceCaps (hdc, BITSPIXEL) ;
```

大多数彩色图形显示设备使用多个色彩平面或每图素有多个色彩位元的设计，但是不能同时一齐使用这两种方式；换句话说，这两个呼叫必有一个传回 1。显示卡能够表示的色彩数可以用如下公式来计算：

```
iColors = 1 << (iPlanes * iBitsPixel) ;
```

这个值与用 NUMCOLORS 参数得到的色彩数值可能一样，也可能不一样：

```
iColors = GetDeviceCaps (hdc, NUMCOLORS) ;
```

我提到过，256 色的显示卡使用色彩调色盘。在那种情况下，以 NUMCOLORS 为参数时，GetDeviceCaps 传回由 Windows 保留的色彩数，值为 20，剩余的 236 种颜色可以由 Windows 程式用调色盘管理器设定。对于 High-Color 和 True-Color 显示解析度，带有 NUMCOLORS 参数的 GetDeviceCaps 通常传回-1，这样就无法得到需要的资讯，因此应该使用前面所示的带有 PLANES 和 BITSPIXEL 值的 iColors 公式。

在大多数 GDI 函式呼叫中，使用 COLORREF 值（只是一个 32 位元的无正负号长整数）来表示一种色彩。COLORREF 值按照红、绿和蓝色的亮度指定了一种颜色，通常叫做「RGB 色彩」。32 位元的 COLORREF 值的设定如图 5-4 所示。

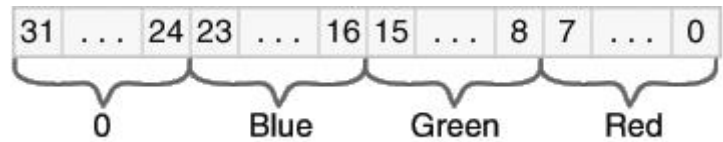


图 5-4 32 位 COLORREF 值

注意最前面是标为 0 的 8 个位元，并且每种原色都指定为一个 8 位元的值。理论上，COLORREF 可以指定二的二十四次方种或一千六百万种色彩。

这个无正负号长整数常常称为一个「RGB 色彩」。Windows 表头档案 WINGDI.H 提供了几种使用 RGB 色彩值的巨集。RGB 巨集要求三个参数分别代表红、绿和蓝值，然后将它们组合为一个无正负号长整数：

```
#define RGB(r,g,b) ((COLORREF)((BYTE)(r) | \
                        ((WORD)((BYTE)(g)) << 8) | \
                        ((DWORD)(BYTE)(b)) << 16)))
```

注意三个参数的顺序是红、绿和蓝。因此，值：

```
RGB (255, 255, 0)
```

是 0x0000FFFF，或黄色（红色和绿色的合成）。当所有三个参数设定为 0 时，色彩为黑色；当所有参数设定为 255 时，色彩为白色。GetRValue、GetGValue 和 GetBValue 巨集从 COLORREF 值中抽取出原色值。当您在使用传回 RGB 色彩值的 Windows 函式时，这些巨集有时会很方便。

在 16 色或 256 色显示卡上，Windows 可以使用「混色」来类比设备能够显示的颜色之外的色彩。混色利用了由多种色彩的图素组成的图素图案。可以呼叫 GetNearestColor 来决定与某一色彩最接近的纯色：

```
crPureColor = GetNearestColor (hdc, crColor) ;
```

装置内容属性

前面已经提到过，Windows 使用装置内容来保存控制 GDI 函式在显示器上如何操作的「属性」。例如，在用 TextOut 函式显示文字时，程式写作者不必指定文字的色彩和字体，Windows 从装置内容取得这个资讯。

程式取得一个装置内容的代号时，Windows 用预设值设定所有的属性（在下一节会看到如何取代这种设定）。表 5-1 列出了 Windows 98 支援的装置内容属性，程式可以改变或者取得任何一种属性。

表 5-1

装置内容属性	预设值	修改该值的函式	取得该值的函式
Mapping Mode	MM_TEXT	SetMapMode	GetMapMode
Window Origin	(0, 0)	SetWindowOrgEx OffsetWindowOrgEx	GetWindowOrgEx

Viewport Origin	(0, 0)	SetViewportOrgEx OffsetViewportOrgEx	GetViewportOrgEx
Window Extents	(1, 1)	SetWindowExtEx SetMapMode ScaleWindowExtEx	GetWindowExtEx
Viewport Extents	(1, 1)	SetViewportExtEx SetMapMode ScaleViewportExtEx	GetViewportExtEx
Pen	BLACK_PEN	SelectObject	SelectObject
Brush	WHITE_BRUSH	SelectObject	SelectObject
Font	SYSTEM_FONT	SelectObject	SelectObject
Bitmap	None	SelectObject	SelectObject
Current Position	(0, 0)	MoveToEx LineTo PolylineTo PolyBezierTo	GetCurrentPositionEx
Background Mode	OPAQUE	SetBkMode	GetBkMode
Background Color	White	SetBkColor	GetBkColor
Text Color	Black	SetTextColor	GetTextColor
Drawing Mode	R2_COPYPEN	SetROP2	GetROP2
Stretching Mode	BLACKONWHITE	SetStretchBltMode	GetStretchBltMode
Polygon Fill Mode	ALTERNATE	SetPolyFillMode	GetPolyFillMode
Intercharacter Spacing	0	SetTextCharacterExtra	GetTextCharacterExtra
Brush Origin	(0, 0)	SetBrushOrgEx	GetBrushOrgEx
Clipping Region	None	SelectObject SelectClipRgn IntersectClipRgn OffsetClipRgn ExcludeClipRect SelectClipPath	GetClipBox

保存装置内容

通常，在您呼叫 GetDC 或 BeginPaint 时，Windows 用预设值建立一个新的装置内容，您对属性所做的一切改变在装置内容用 ReleaseDC 或 EndPaint 呼叫释放时，都会丢失。如果您的程式需要使用非内定的装置内容属性，则您必须在每次取得装置内容代号时初始化装置内容：

```
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    装置内容属性
    绘制视窗显示区域
    EndPaint (hwnd, &ps) ;
    return 0 ;
```

虽然在通常情况下这种方法已经很令人满意了，但是您可能想要在释放装置内容之後，仍然保存程式中对装置内容属性所做的改变，以便在下一次呼叫 GetDC 和 BeginPaint 时它们仍然能够起作用。为此，可在登录视窗类别时，将 CS_OWNDC 旗标纳入视窗类别的一部分：

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC ;
```

现在，依据这个视窗类别所建立的每个视窗都将拥有自己的装置内容，它一直存在，直到视窗被删除。如果使用了 CS_OWNDC 风格，就只需初始化装置内容一次，可以在处理 WM_CREATE 讯息处理期间完成这一操作：

```
case WM_CREATE:
    hdc = GetDC (hwnd) ;
```

初始化装置内容属性

```
ReleaseDC (hwnd, hdc) ;
```

这些属性在改变之前一直有效。

CS_OWNDC 风格只影响 GetDC 和 BeginPaint 获得的装置内容，不影响其他函数（如 GetWindowDC）获得的装置内容。以前不提倡使用 CS_OWNDC 风格，因为它需要记忆体；现在，在处理大量图形的 Windows NT 應用程式中，它可以提高性能。即使用了 CS_OWNDC，您仍然应该在退出视窗讯息处理程式之前释放装置内容。

某些情况下，您可能想改变某些装置内容属性，用改变後的属性进行绘图，然後恢复原来的装置内容。要简化这一过程，可以通过如下呼叫来保存装置内容的状态：

```
idSaved = SaveDC (hdc) ;
```

现在，可以改变一些属性，在想要回到呼叫 SaveDC 前存在的装置内容时，呼叫：

```
RestoreDC (hdc, idSaved) ;
```

您可以在呼叫 RestoreDC 之前呼叫 SaveDC 数次。

大多数程式写作者以不同的方式使用 SaveDC 和 RestoreDC。然而，更像组合语言中的 PUSH 和 POP 指令，当您呼叫 SaveDC 时，不需要保存传回值：

```
SaveDC (hdc) ;
```

然後，您可以更改某些属性并再次呼叫 SaveDC。要将装置内容恢复到一个已经保存的状态，呼叫：

```
RestoreDC (hdc, -1) ;
```

这就将装置内容恢复到最近由 SaveDC 函式保存的状态中。

画点和线

在第一章，我们谈论过 Windows 图形装置介面将图形输出设备的装置驱动程序与电脑连在一起的方式。在理论上，只要提供 SetPixel 和 GetPixel 函式，就可以使用图形装置驱动程序绘制一切东西了。其余的一切都可以使用 GDI 模组中实作的更高阶的常式来处理。例如，画线时，只需 GDI 呼叫 SetPixel 数次，并适当地调整 x 和 y 座标。

在实际情况中，也的确可以仅使用 SetPixel 和 GetPixel 函式进行您需要的任何绘制。您也可以在这些函式的基础上设计出简洁和构造良好的图形编程系统。唯一的问题是启能。如果一个函式通过几次呼叫才能到达 SetPixel 函式，那么它执行起来会非常慢。如果一个图形系统画线和进行其他复杂的图形操作是在装置驱动程序的层次上，它就会更有效得多，因为装置驱动程序对完成这些操作的程式码进行了最佳化。此外，一些显示卡包含了图形辅助运算器，它允许视讯硬体自己绘制图形。

设定图素

即使 Windows GDI 包含了 SetPixel 和 GetPixel 函式，但很少使用它们。在本书，仅在第七章的 CONNECT 程式中使用了 SetPixel 函式，仅在第八章的 WHATCLR 程式中使用了 GetPixel 函式。尽管如此，由它们开始来研究图形仍是非常方便。

SetPixel 函式在指定的 x 和 y 座标以特定的颜色设定图素：

```
SetPixel (hdc, x, y, crColor) ;
```

如同在任何绘图函式中一样，第一个参数是装置内容的代号。第二个和第三个参数指明了座标位置。通常要获得视窗显示区域的装置内容，并且 x 和 y 相对于该显示区域的左上角。最後一个参数是 COLORREF 型态指定了颜色。如果在函式中指定的颜色视讯显示器不支援，则函式将图素设定为最接近的纯色并从函式传回该值。

GetPixel 函式传回指定座标处的图素颜色：

```
crColor = GetPixel (hdc, x, y) ;
```

直线

Windows 可以画直线、椭圆线（椭圆圆周上的曲线）和贝塞尔曲线。Windows 98 支援的 7 个画线函式是：

- LineTo 画直线。
- Polyline 和 PolylineTo 画一系列相连的直线。
- PolyPolyline 画多组相连的线。
- Arc 画椭圆线。
- PolyBezier 和 PolyBezierTo 画贝塞尔曲线。

另外，Windows NT 还支援 3 种画线函式：

- ArcTo 和 AngleArc 画椭圆线。
- PolyDraw 画一系列相连的线以及贝塞尔曲线。

这三个函式 Windows 98 不支援。

在本章的後面我将介绍一些既画线也填入所画图形的封闭区域的函式，这些函式是：

- Rectangle 画矩形。
- Ellipse 画椭圆。
- RoundRect 画带圆角的矩形。
- Pie 画椭圆的一部分，使其看起来像一个扇形。
- Chord 画椭圆的一部分，以呈弓形。

装置内容的五个属性影响著用这些函式所画线的外观：目前画笔的位置（仅用於 LineTo、PolylineTo、PolyBezierTo 和 ArcTo）、画笔、背景方式、背景色和绘图模式。

画一条直线，必须呼叫两个函式。第一个函式指定了线的开始点，第二个函式指定了线的终点：

```
MoveToEx (hdc, xBeg, yBeg, NULL) ;
LineTo (hdc, xEnd, yEnd) ;
```

MoveToEx 实际上不会画线，它只是设定了装置内容的「目前位置」属性。然後 LineTo 函式从目前的位置到它所指定的点画一条直线。目前位置只是用於其他几个 GDI 函式的开始点。在内定的装置内容中，目前位置最初设定在点 (0,0)。如果在呼叫 LineTo 之前没有设定目前位置，那么它将从显示区域的左上角开始画线。

小历史：

Windows 的 16 位元版本中，用来改变目前位置的函式是 MoveTo。该函式只调整三个参数——装置内容代号、x 和 y 座标。函式通过两个 16 位元数拼成的 32 位元无正负号长整数传回先前的目前位置。然而，在 Windows 的 32 位元版本中，座标是 32 位元的数值，而 C 的 32 位元版本中又没有定义 64 位元的整数资料型态，因此这种改变意味著 MoveTo 在其传回值中不再指出先前的目前位置。在实际的程式写作中，由 MoveTo 传回的值几乎从来不用，因此就需要一个新函式，这就是 MoveToEx。

MoveToEx 的最後一个参数是指向 POINT 结构的指标。从该函式传回後，POINT 结构的 x 和 y 栏位指出了先前的目前位置。如果您不需要这种资讯（通常如此），可以简单地如上面的例子所示的那样将最後一个参数设定为 NULL。

警告：

尽管 Windows 98 中的座标值看起来是 32 位元的，实际上却只用到了低 16 位元，座标值实际上被限制在 -32,768 到 32,767 之间。在 Windows NT 中，使用完整的 32 位元值。

如果您需要目前位置，就可以通过以下呼叫获得：

```
GetCurrentPositionEx (hdc, &pt) ;
```

其中，pt 是 POINT 结构的。

下面的程式码从视窗的左上角开始，在显示区域中画一个网格，线与线之间相隔 100 个图素，其中 hwnd 是视窗代号，hdc 是装置内容代号，而 x 和 y 是整数：

```
GetClientRect (hwnd, &rect) ;
for ( x = 0 ; x < rect.right ; x+= 100)
{
    MoveToEx (hdc, x, 0, NULL) ;
    LineTo (hdc, x, rect.bottom) ;
}
for (y = 0 ; y < rect.bottom ; y += 100)
{
    MoveToEx (hdc, 0, y, NULL) ;
    LineTo (hdc, rect.right, y) ;
}
```

虽然用两个函式来画一条直线显得有些麻烦，但是在希望画一组相连的直线时，目前画笔位置属性又会变得很有用。例如，您可能想定义一个包含 5 个点（10 个值）的阵列，来画一个矩形的边界框：

```
POINT apt[5] = { 100, 100, 200, 100, 200, 200, 100, 200, 100, 100 } ;
```

注意，最後一个点与第一个点相同。现在，只需要使用 MoveToEx 移到第一个点，并对後面的点使用 LineTo:

```
MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
for ( i = 1 ; i < 5 ; i++)
    LineTo (hdc, apt[i].x, apt[i].y) ;
```

由於 LineTo 从目前位置画到（但不包括）LineTo 函式中给出的点，所以这段程式码没有在任何座标处画两次。虽然在显示器上多输出几次不存在问题，但是在绘图机上或者在其他绘图方式（下面马上会讲到）下，视觉效果就不太好了。

当您要將阵列中的点连接成线时，使用 Polyline 函式要简单得多。下面这条叙述画出与上面一段程式码相同的矩形:

```
Polyline (hdc, apt, 5) ;
```

最後一个参数是点的数目。我们还可以使用 (sizeof (apt) / sizeof (POINT)) 来表示这个值。Polyline 与一个 MoveToEx 函式後面加几个 LineTo 函式的效果相同，但是，Polyline 既不使用也不改变目前位置。PolylineTo 有些不同，这个函式使用目前位置作为开始点，并将目前位置设定为最後一根线的终点。下面的程式码画出与上面所示一样的矩形:

```
MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
PolylineTo (hdc, apt + 1, 4) ;
```

您可以对几条线使用 Polyline 和 PolylineTo，这些函式在绘制复杂曲线最有用了。您使用由几百甚至几千条线组成的极短线段，把它们连在一起就像一条曲线一样。例如，画正弦波就是这样的，程式 5-2 所示的 SINEWAVE 程式显示了如何做到这一点。

程式 5-2 SINEWAVE

```
SINEWAVE.C
/*-----
   SINEWAVE.C -- Sine Wave Using Polyline
   (c) Charles Petzold, 1998
   -----*/
#include <windows.h>
#include <math.h>

#define NUM1000
#define TWOPI    (2 * 3.14159)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("SineWave") ;
    HWND          hwnd ;
```

```

MSG          msg ;
WNDCLASS     wndclass ;

wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc= WndProc ;
wndclass.cbClsExtra  = 0 ;
wndclass.cbWndExtra  = 0 ;
wndclass.hInstance  = hInstance ;
wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName = NULL ;
wndclass.lpszClassName = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Sine Wave Using Polyline"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}

return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxClient, cyClient ;
    HDC        hdc ;
    int         i ;
    PAINTSTRUCT ps ;
    POINT       apt [NUM] ;

    switch (message)
    {
    case WM_SIZE:

```

```
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    MoveToEx (hdc, 0,          cyClient / 2, NULL) ;
    LineTo   (hdc, cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++)
    {
        apt[i].x = i * cxClient / NUM ;
        apt[i].y = (int) (cyClient / 2 * (1 - sin (TWOPI * i / NUM))) ;
    }

    Polyline (hdc, apt, NUM) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个程式有一个含有 1000 个 POINT 结构的阵列。随著 for 回圈从 0 增加到 999，结构的 x 成员设定为从 0 递增到数值 cxClient。结构的 y 成员设定为一个周期的正弦曲线值，并被放大以填满显示区域。整个曲线的绘制仅仅使用了一个 Polyline 呼叫。因为 Polyline 函式是在装置驱动程式层次上实作的，因此它要比呼叫 1000 次 LineTo 快得多，结果如图 5-5 所示。

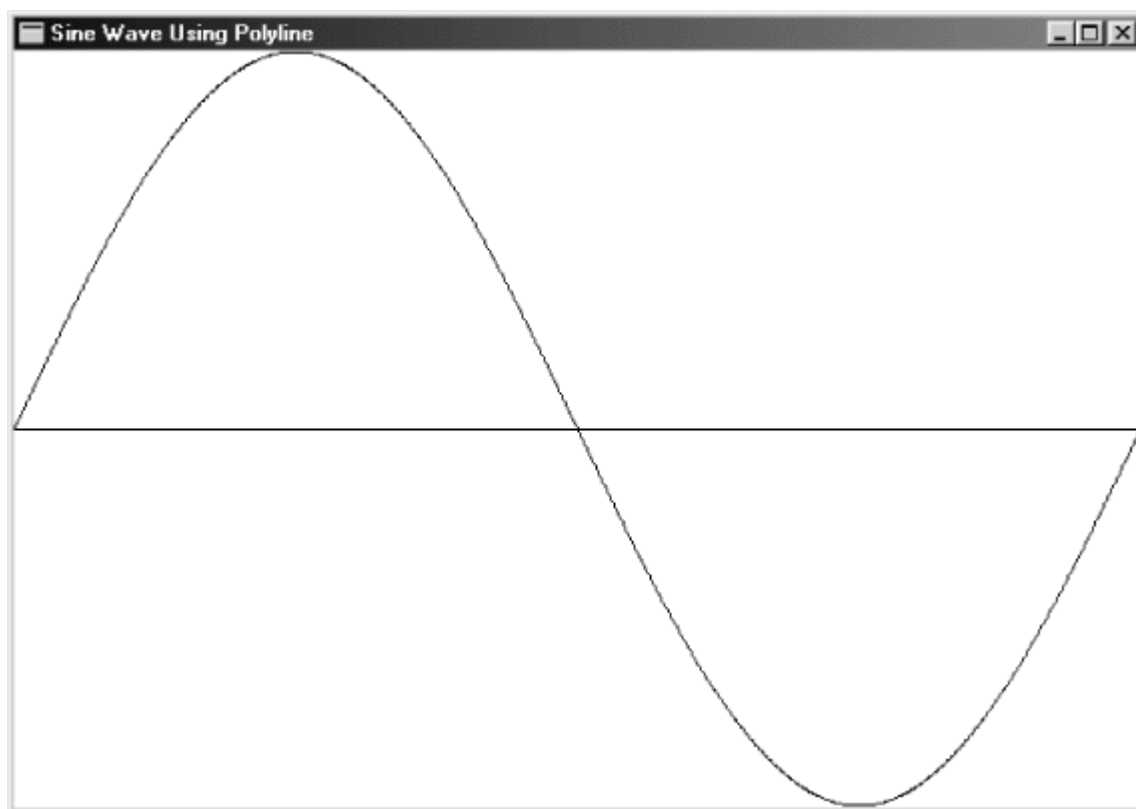


图 5-5 SINEWAVE 显示

边界框函式

下面我想讨论的是 Arc 函式，它绘制椭圆曲线。然而，如果不先讨论一下 Ellipse 函式，那么 Arc 函式将难以理解；而如果不先讨论 Rectangle 函式，那么 Ellipse 函式又将难以理解；而如果讨论 Ellipse 和 Rectangle 函式，那么我又会讨论 RoundRect、Chord 和 Pie 函式。

问题在於，Rectangle、Ellipse、RoundRect、Chord 和 Pie 函式严格来说不是画线函式。没错，这些函式是在画线，但它们同时又填入画刷填入一个封闭区域。这个画刷内定为白色，因此当您第一次使用这些函式时，您可能不会注意到它们不只是画线。严格地说，这些函式属于後面「填入区域」的小节，不过，我还是在这里讨论它们。

上面提到的函式有一个共同特性，即它们都是依据一个矩形边界框来绘图的。您定义一个包含该物件的框，即「边界框 (bounding box)」；Windows 就在这个框内画出该物件。

这些函式中最简单的就是画一个矩形：

```
Rectangle (hdc, xLeft, yTop, xRight, yBottom) ;
```

点 (xLeft, yTop) 是矩形的左上角，(xRight, yBottom) 是矩形的右下角。用函式 Rectangle 画出的图形如图 5-6 所示，矩形的边总是平行于显示器的水平和垂直边。

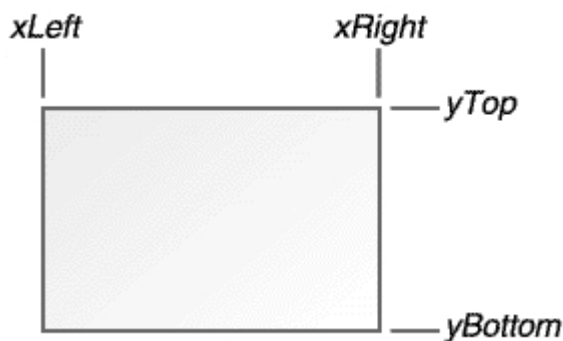


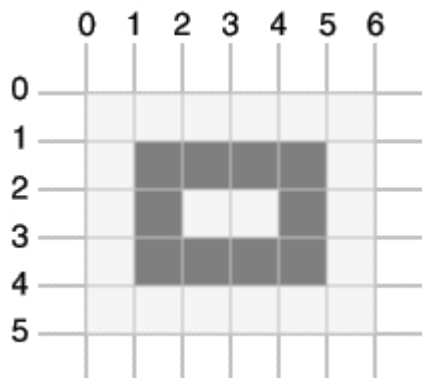
图 5-6 使用 Rectangle 函式画出的图形

以前写过图形程式的程式写作者熟悉图素偏差的问题。有些图形系统画出的图形包含右座标和底座标，而有些则只画到（而不包含）右座标和底座标。Windows 采用後一种方法，不过有一种更简单的方法来思考这个问题。

考虑下面的函式呼叫：

```
Rectangle (hdc, 1, 1, 5, 4) ;
```

上面我们提到，Windows 在边界框内画图。可以将显示器想像成一个网格，其中，每个图素都在一个网格单元内。边界框画在网格上，然後在边界框内画矩形，下面说明了图形画出来时的样子：



将矩形和显示区域左上角分开的区域有 1 个图素宽。

我以前提到过，Rectangle 严格地说不是画线函式，GDI 也填入封闭区域。然而，因为内定用白色填入区域，因此 GDI 填入区域并不明显。

您知道了如何画矩形，也就知道了如何画椭圆，因为它们使用的参数都是相同的：

```
Ellipse (hdc, xLeft, yTop, xRight, yBottom) ;
```

用 Ellipse 函式画出的图形如图 5-7 所示（加上了虚线构成的边界框）。

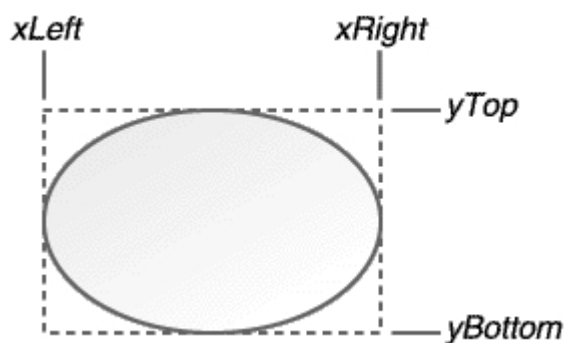


图 5-7 用 Ellipse 函式画出的图形

画圆角矩形的函式使用与函式 Rectangle 及 Ellipse 函式相同的边界框，还包含另外两个参数：

```
RoundRect (hdc, xLeft, yTop, xRight, yBottom,
           xCornerEllipse, yCornerEllipse) ;
```

用这个函式画出的图形如 5-8 所示。

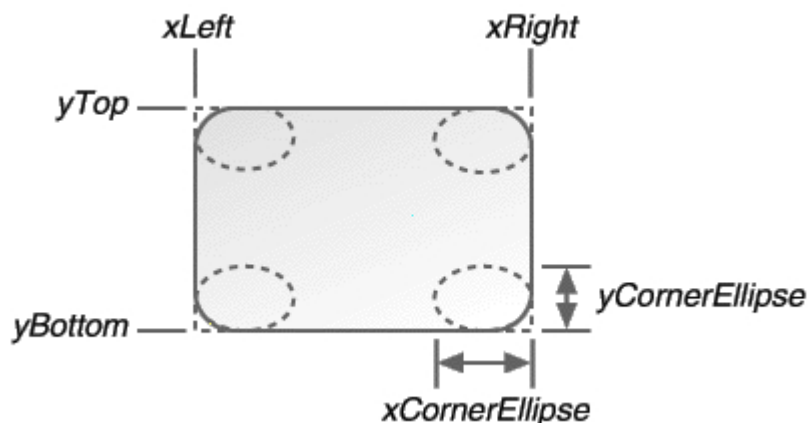


图 5-8 用 RoundRect 函式画出的图形

Windows 使用一个小椭圆来画圆角，这个椭圆的宽为 xCornerEllipse，高为 yCornerEllipse。可以想像这个小椭圆分为了四个部分，一个象限一个，每个刚好用在矩形的一个角上。xCornerEllipse 和 yCornerEllipse 的值越大，角就越明显。如果 xCornerEllipse 等於 xLeft 与 xRight 的差，且 yCornerEllipse 等於 yTop 与 yBottom 的差，那么 RoundRect 函式将画出一个椭圆。

在绘制图 5-8 所示的圆角矩形时，用了下面的公式来计算角上椭圆的尺寸。

```
xCornerEllipse = (xRight - xLeft) / 4 ;
yCornerEllipse = (yBottom - yTop) / 4 ;
```

这是一种简单的方法，但是结果看起来有点不对劲，因为角的弯曲部分在矩形长的一边要大些。要矫正这一问题，您可以让 xCornerEllipse 与 yCornerEllipse 的值相等。

Arc、Chord 和 Pie 函式都只要相同的参数：

```
Arc (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd) ;
Chord (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd) ;
Pie (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd) ;
```

用 Arc 函式画出的线如图 5-9 所示；用 Chord 和 Pie 函式画出的线分别如图 5-10 和 5-11 所示。Windows 用一条假想的线将 (xStart, yStart) 与椭圆的中心连接，从该线与边界框的交点开始，Windows 按反时针方向，沿著椭圆画一条弧。Windows 还用另一条假想的线将 (xEnd, yEnd) 与椭圆的中心连接，在该线与边界框的交点处，Windows 停止画弧。

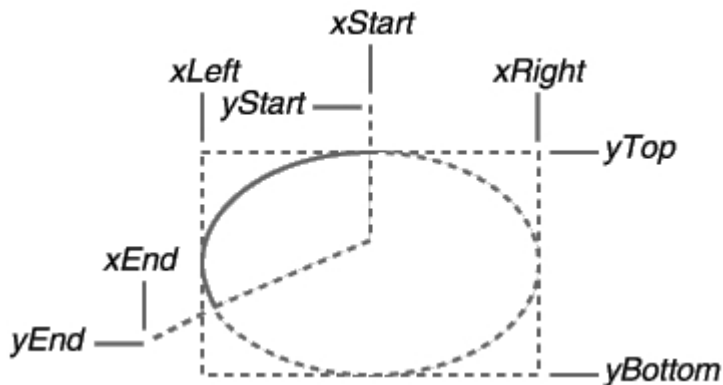


图 5-9 Arc 函式画出的线

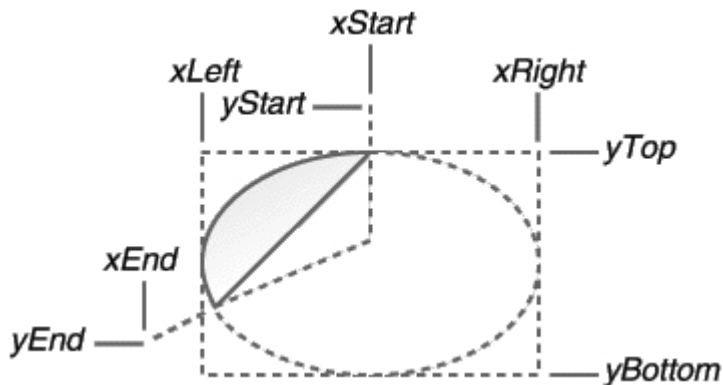


图 5-10 Chord 函式画出的线

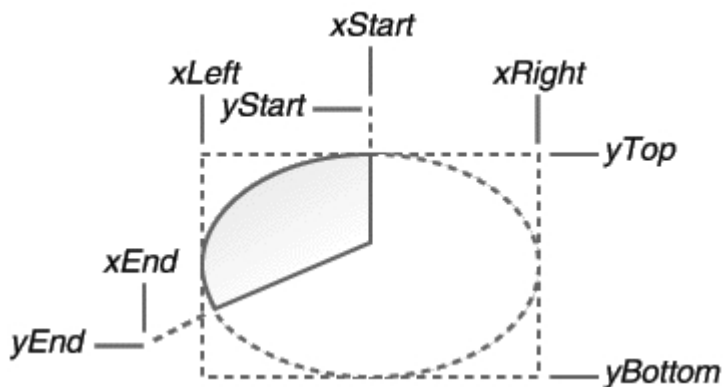


图 5-11 Pie 函式画出的线

对于 Arc 函式，这样就结束了。因为弧只是一条椭圆形的线而已，而不是一个填入区域。对于 Chord 函式，Windows 连接弧线的端点。而对于 Pie 函式，Windows 将弧的两个端点与椭圆的中心相连接。弦与扇形图的内部以目前画刷填入。

您可能不太明白在 Arc、Chord 和 Pie 函式中开始和结束位置的用法，为什

么不简单地在椭圆的周线上指定开始和结束点呢？是的，您可以这么做，但是您将不得不算出这些点。Windows 的方法在不要求这种精确性的条件下，却完成了相同的工作。

程式 5-3 LINEDEMO 画一个矩形、一个椭圆、一个圆角矩形和两条线段，不过不是按这一顺序。程式表明了定义封闭区域的函式实际上对这些区域进行了填入，因为在椭圆后面的线被遮住了，结果如图 5-12 中所示。

程式 5-3 LINEDEMO

```
LINEDEMO.C
/*-----
   LINEDEMO.C -- Line-Drawing Demonstration Program
                   (c) Charles Petzold, 1998
   -----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("LineDemo") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Line Demonstration"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
```

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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}

return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxClient, cyClient ;
    HDC        hdc ;
    PAINTSTRUCT ps ;

    switch (message)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        Rectangle (hdc,      cxClient / 8,      cyClient / 8,
                    7 * cxClient / 8, 7 * cyClient / 8) ;

        MoveToEx (hdc,      0,      0, NULL) ;
        LineTo   (hdc, cxClient, cyClient) ;

        MoveToEx (hdc,      0, cyClient, NULL) ;
        LineTo   (hdc, cxClient,      0) ;

        Ellipse  (hdc,      cxClient / 8,      cyClient / 8,
                    7 * cxClient / 8, 7 * cyClient / 8) ;

        RoundRect (hdc,      cxClient / 4,      cyClient / 4,
                    3 *      cxClient / 4, 3 * cyClient / 4,
                    cxClient / 4,      cyClient / 4) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;
    }
```

```
case WM_DESTROY:  
    PostQuitMessage (0) ;  
    return 0 ;  
}  
return DefWindowProc (hwnd, message, wParam, lParam) ;  
}
```

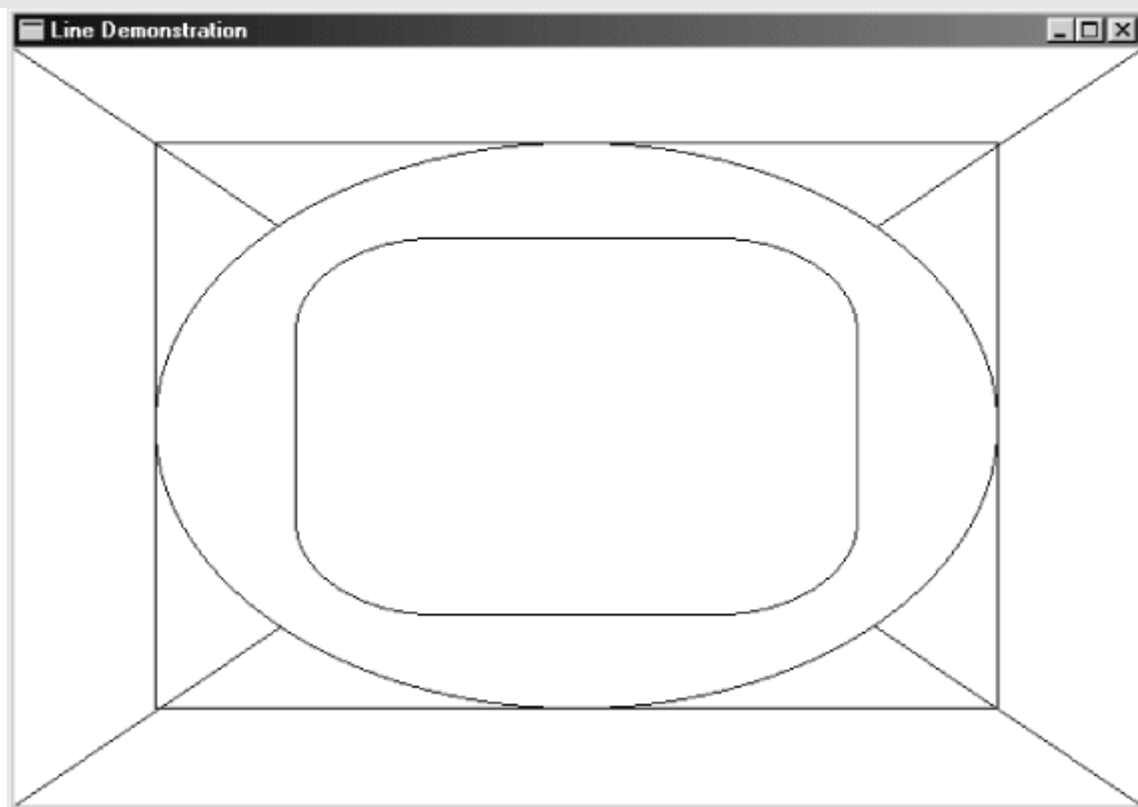


图 5-12 LINEDEMO 显示

贝塞尔曲线

「曲尺」这个词从前指的是一片木头、橡皮或者金属，用来在纸上画曲线。比如说，如果您有一些不同图点，您想要在它们之间画一条曲线（内插或者外插），您首先将这些点描在绘图纸上，然后，将曲尺定在这些点上，并用铅笔沿著曲尺绕著这些点弯曲的方向画曲线。

当然，时至今日，曲尺已经数学公式化了。有很多种不同的曲尺公式，它们各有千秋。贝塞尔曲线是电脑程式设计中用得最广的曲尺公式之一，它是直到最近才加到作业系统层次的图形支援中的。在六十年代 Renault 汽车公司进行了由手工设计车体（要用到粘土）到电脑辅助设计的转变。他们需要一些数学工具，而 Pierm Bezier 找到了一套公式，最后显示出这套公式应付这样的工作非常有用。

此后，二维的贝塞尔曲线成了电脑图学中最有用的曲线（在直线和椭圆之后）。在 PostScript 中，所有曲线都用贝塞尔曲线表示——椭圆线用贝塞尔曲

线来逼近。贝塞尔曲线也用於定义 PostScript 字体的字元轮廓 (TrueType 使用一种更简单更快速的曲尺公式)。

一条二维的贝塞尔曲线由四个点定义——两个端点和两个控制点。曲线的端点在两个端点上, 控制点就好像「磁石」一样把曲线从两个端点间的直线处拉走。这一点可以由底下的 BEZIER 互动交谈程式做出最好的展示, 如程式 5-4 所示。

程式 5-4 BEZIER

```
BEZIER.C
/*-----
    BEZIER.C -- Bezier Splines Demo
                (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Bezier") ;
    HWND        hwnd ;
    MSG         msg ;
    WNDCLASS    wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (    szAppName, TEXT ("Bezier Splines"),
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
```



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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void DrawBezier (HDC hdc, POINT apt[])
{
    PolyBezier (hdc, apt, 4) ;
    MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
    LineTo   (hdc, apt[1].x, apt[1].y) ;

    MoveToEx (hdc, apt[2].x, apt[2].y, NULL) ;
    LineTo   (hdc, apt[3].x, apt[3].y) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static POINT apt[4] ;
    HDC         hdc ;
    int         cxClient, cyClient ;
    PAINTSTRUCT ps ;

    switch (message)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;

        apt[0].x = cxClient / 4 ;
        apt[0].y = cyClient / 2 ;

        apt[1].x = cxClient / 2 ;
        apt[1].y = cyClient / 4 ;

        apt[2].x = cxClient / 2 ;
        apt[2].y = 3 * cyClient / 4 ;

        apt[3].x = 3 * cxClient / 4 ;
        apt[3].y = cyClient / 2 ;

        return 0 ;
    }
```

```
case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:
case WM_MOUSEMOVE:
    if (wParam & MK_LBUTTON || wParam & MK_RBUTTON)
    {
        hdc = GetDC (hwnd) ;
        SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
        DrawBezier (hdc, apt) ;

        if (wParam & MK_LBUTTON)
        {
            apt[1].x = LOWORD (lParam) ;
            apt[1].y = HIWORD (lParam) ;
        }

        if (wParam & MK_RBUTTON)
        {
            apt[2].x = LOWORD (lParam) ;
            apt[2].y = HIWORD (lParam) ;
        }

        SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
        DrawBezier (hdc, apt) ;
        ReleaseDC (hwnd, hdc) ;
    }
    return 0 ;
case WM_PAINT:
    InvalidateRect (hwnd, NULL, TRUE) ;

    hdc = BeginPaint (hwnd, &ps) ;

    DrawBezier (hdc, apt) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

由於这个程式要用到一些在第七章才讲的滑鼠处理方式，所以我不在这里讨论它的内部运作（不过，这也是简单的），而是用这个程式来实验性地操纵贝塞尔曲线。在这个程式中，两个顶点设定在显示区域的上下居中、左右位於

1/4 和 3/4 处的位置；两个控制点可以改变，按住滑鼠左键或右键并拖动滑鼠可以分别改动两个控制点之一。图 5-13 是一个典型的例子。

除了贝塞尔曲线本身，程式还从第一个控制点向左边的第一个端点（也叫做开始点）画一条直线，并从第二个控制点向右边的端点画一条直线。

由於下面几个特点，贝塞尔曲线在电脑辅助设计中非常有用。首先，经过少量练习，就可以把曲线调整到与想要的形状非常接近。

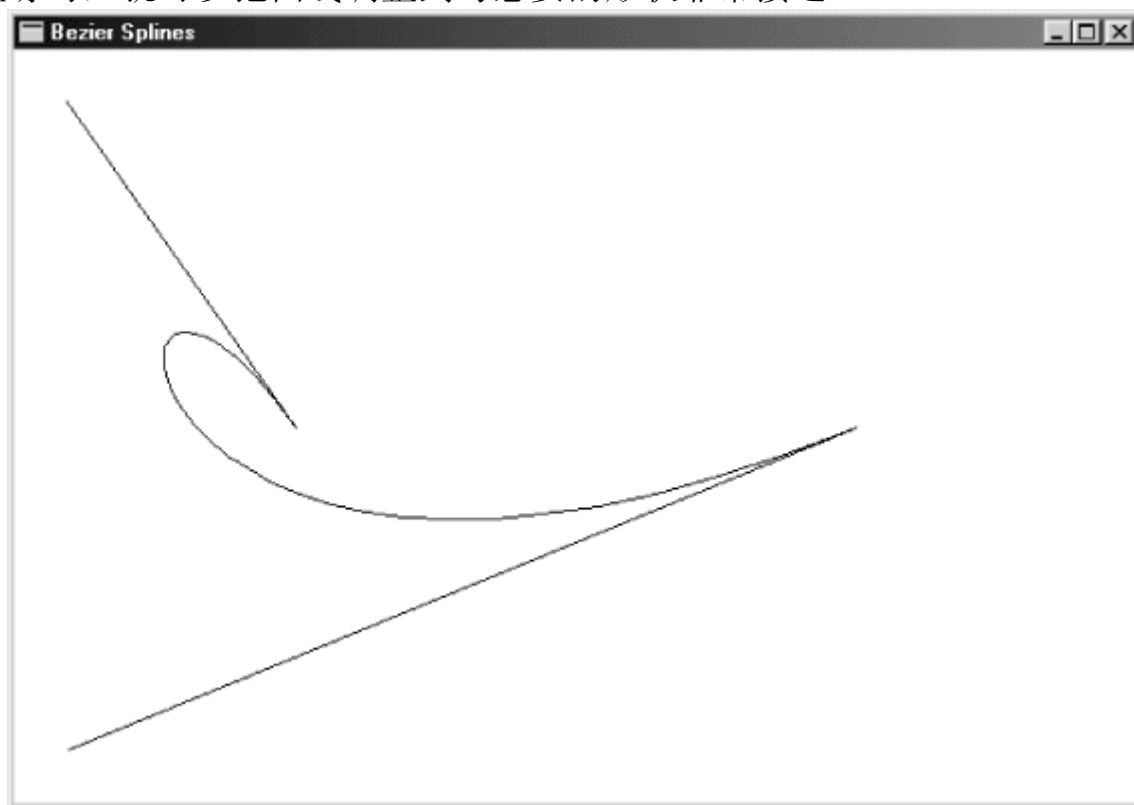


图 5-13 BEZIER 程式的显示

其次，贝塞尔曲线非常好控制。对於有的曲尺种类来说，曲线不经过任何一个定义该曲线的点。贝塞尔曲线总是由其两个端点开始和结束的（这是在推导贝塞尔公式时所做的假设之一）。另外，有些形式的曲尺公式有奇异点，在这些点处曲线趋向无穷远，这在电脑辅助设计中通常是很不合适的。事实上，贝塞尔曲线总是受限於一个四边形（叫做「凸包」），这个四边形由端点和控制点连接而成。

第三个特点涉及端点和控制点之间的关系。曲线总是与第一个控制点到起点的直线相切，并保持同一方向；同时，也与第二个控制点到终点的直线相切，并保持同一方向。这是用於推导贝塞尔公式时所做的另外两个假设。

第四，贝塞尔曲线通常比较具有美感。我知道这是一个主观评价的问题，不过，并非只有我才这样想。

在 32 位元的 Windows 版本之前，您必须利用 Polyline 来自己建立贝塞尔曲线，并且还需要知道下面的贝塞尔曲线的参数方程。起点是 (x0, y0)，终点

是 (x3, y3)，两个控制点是 (x1, y1) 和 (x2, y2)，随著 t 的值从 0 到 1 的变化，就可以画出曲线：

$$x(t) = (1 - t)^3 x_0 + 3t(1 - t)^2 x_1 + 3t^2(1 - t)x_2 + t^3 x_3$$

$$y(t) = (1 - t)^3 y_0 + 3t(1 - t)^2 y_1 + 3t^2(1 - t)y_2 + t^3 y_3$$

在 Windows 98 中，您不需要知道这些公式。要画一条或多条连接的贝塞尔曲线，只需呼叫：

```
PolyBezier (hdc, apt, iCount) ;
```

或

```
PolyBezierTo (hdc, apt, iCount) ;
```

两种情况下，apt 都是 POINT 结构的阵列。对 PolyBezier，前四个点（按照顺序）给出贝塞尔曲线的起点、第一个控制点、第二个控制点和终点。此後的每一条贝塞尔曲线只需给出三个点，因为後一条贝塞尔曲线的起点就是前一条贝塞尔曲线的终点，如此类推。iCount 参数等於 1 加上您所绘制的这些首尾相接曲线条数的三倍。

PolyBezierTo 函式使用目前点作为第一个起点，第一条以及後续的贝塞尔曲线都只需要给出三个点。当函式传回时，目前点设定为最後一个终点。

一点提示：在画一系列相连的贝塞尔曲线时，只有当第一条贝塞尔曲线的第二个控制点、第一条贝塞尔曲线的终点（也就是第二条曲线的起点）和第二条贝塞尔曲线的第一个控制点线性相关时，也就是说这三个点在同一条直线上时，曲线在连接点处才是光滑的。

使用现有画笔 (Stock Pens)

当您呼叫这一节中讨论的任何画线函式时，Windows 使用装置内容中目前选中的「画笔」来画线。画笔决定线的色彩、宽度和画笔样式，画笔样式可以是实线、点划线或者虚线，内定装置内容中画笔为 BLACK_PEN。不管映射方式是什么，这种画笔都画出一个图素宽的黑色实线来。BLACK_PEN 是 Windows 提供的三种现有画笔之一，其他两种是 WHITE_PEN 和 NULL_PEN，NULL_PEN 什么都不画。您也可以自己自订画笔。

Windows 程式以代号来使用画笔。Windows 表头档案 WINDEF.H 中包含一个叫做 HPEN 的型态定义，即画笔的代号，可以定义这个型态的变数（例如 hPen）：

```
HPEN hPen ;
```

呼叫 GetStockObject，可以获得现有画笔的代号。例如，假设您想使用名为 WHITE_PEN 的现有画笔，可以如下取得画笔的代号：

```
hPen = GetStockObject (WHITE_PEN) ;
```

现在必须将画笔选进装置内容：

```
SelectObject (hdc, hPen) ;
```

目前的画笔是白色。在这个呼叫後，您画的线将使用 WHITE_PEN，直到您将另外一个画笔选进装置内容或者释放装置内容代号为止。

您也可以不定义 hPen 变数，而将 GetStockObject 和 SelectObject 呼叫合并成一个叙述：

```
SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
```

如果想恢复到使用 BLACK_PEN 的状态，可以用一个叙述取得这种画笔的代号，并将其选进装置内容：

```
SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
```

SelectObject 的传回值是此呼叫前装置内容中的画笔代号。如果启动一个新的装置内容并呼叫

```
hPen = SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
```

则装置内容中的目前画笔将为 WHITE_PEN，变数 hPen 将会是 BLACK_PEN 的代号。以後通过呼叫

```
SelectObject (hdc, hPen) ;
```

就能够将 BLACK_PEN 选进装置内容。

画笔的建立、选择和删除

尽管使用现有画笔非常方便，但却受限於实心的黑画笔、实心的白画笔或者没有画笔这三种情况。如果想得到更丰富多彩的效果，就必须建立自己的画笔。

这一过程通常是：使用函式 CreatePen 或 CreatePenIndirect 建立一个「逻辑画笔」，这仅仅是对画笔的描述。这些函式传回逻辑画笔的代号；然後，呼叫 SelectObject 将画笔选进装置内容。现在，就可以使用新的画笔来画线了。在任何时候，都只能有一种画笔选进装置内容。在释放装置内容（或者在选择了一种画笔到装置内容中）之後，就可以呼叫 DeleteObject 来删除所建立的逻辑画笔了。在删除後，该画笔的代号就不再有效了。

逻辑画笔是一种「GDI 物件」，它是您可以建立的六种 GDI 物件之一，其他五种是画刷、点阵图、区域、字体和调色盘。除了调色盘之外，这些物件都是通过 SelectObject 选进装置内容的。

在使用画笔等 GDI 物件时，应该遵守以下三条规则：

- 最後要删除自己建立的所有 GDI 物件。
- 当 GDI 物件正在一个有效的装置内容中使用时，不要删除它。
- 不要删除现有物件。

这些规则当然是有道理的，而且有时这道理还挺微妙的。下面我们将举些例子来帮助理解这些规则。

CreatePen 函式的语法形如：

```
hPen = CreatePen (iPenStyle, iWidth, crColor) ;
```

其中，iPenStyle 参数确定画笔是实线、点线还是虚线，该参数可以是 WINGDI.H 表头档案中定义的以下识别字，图 5-14 显示了每种画笔产生的画笔样式。

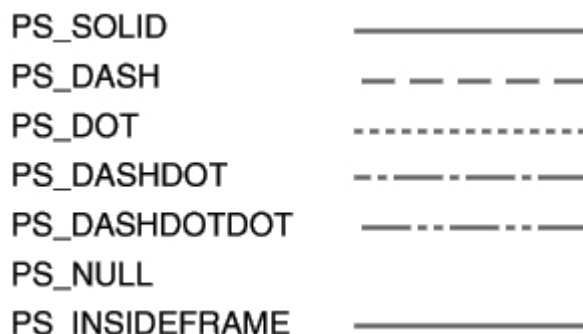


图 5-14 七种画笔样式

对于 PS_SOLID、PS_NULL 和 PS_INSIDEFRAME 画笔样式，iWidth 参数是画笔的宽度。iWidth 值为 0 则意味著画笔宽度为一个图素。现有画笔是一个图素宽。如果指定的是点划线或者虚线式画笔样式，同时又指定一个大於 1 的实际宽度，那么 Windows 将使用实线画笔来代替。

CreatePen 的 crColor 参数是一个 COLORREF 值，它指定画笔的颜色。对于除了 PS_INSIDEFRAME 之外的画笔样式，如果将画笔选入装置内容中，Windows 会将颜色转换为设备所能表示的最相近的纯色。PS_INSIDEFRAME 是唯一一种可以使用混色的画笔样式，并且只有在宽度大於 1 的情况下才如此。

在与定义一个填入区域的函式一起使用时，PS_INSIDEFRAME 画笔样式还有另外一个奇特之处：对于除了 PS_INSIDEFRAME 以外的所有画笔样式来说，如果用来画边界框的画笔宽度大於 1 个图素，那么画笔将居中对齐在边界框线上，这样边界框线的一部分将位於边界框之外；而对于 PS_INSIDEFRAME 画笔样式来说，整条边界框线都画在边界框之内。

您也可以通过建立一个型态为 LOGPEN (「逻辑画笔」) 的结构，并呼叫 CreatePenIndirect 来建立画笔。如果您的程式使用许多能在原始码中初始化的画笔，那么使用这种方法将有效得多。

要使用 CreatePenIndirect，首先定义一个 LOGPEN 型态的结构：

```
LOGPEN logpen ;
```

此结构有三个成员：lopStyle (无正负号整数或 UINT) 是画笔样式，lopWidth (POINT 结构) 是按逻辑单位度量的画笔宽度，lopColor (COLORREF) 是画笔颜色。Windows 只使用 lopWidth 结构的 x 值作为画笔宽度，而忽略 y 值。

将结构的位址传递给 CreatePenIndirect 结构就可以建立画笔了：

```
hPen = CreatePenIndirect (&logpen) ;
```

注意, CreatePen 和 CreatePenIndirect 函式不需要装置内容代号作为参数。这些函式建立与装置内容没有联系的逻辑画笔。直到呼叫 SelectObject 之後, 画笔才与装置内容发生联系。因此, 可以对不同的设备 (如萤幕和印表机) 使用相同的逻辑画笔。

下面是建立、选择和删除画笔的一种方法。假设您的程式使用三种画笔——一种宽度为 1 的黑画笔、一种宽度为 3 的红画笔和一种黑色点式画笔, 您可以先定义三个变数来存放这些画笔的代号:

```
static HPEN hPen1, hPen2, hPen3 ;
```

在处理 WM_CREATE 期间, 您可以建立这三种画笔:

```
hPen1 = CreatePen (PS_SOLID, 1, 0) ;
hPen2 = CreatePen (PS_SOLID, 3, RGB (255, 0, 0)) ;
hPen3 = CreatePen (PS_DOT, 0, 0) ;
```

在处理 WM_PAINT 期间, 或者是在拥有一个装置内容有效代号的任何时间里, 您都可以将这三个画笔之一选进装置内容并用它来画线:

```
SelectObject (hdc, hPen2) ;
```

画线函式

```
SelectObject (hdc, hPen1) ;
```

其他画线函式

在处理 WM_DESTROY 期间, 您可以删除您建立的三种画笔:

```
DeleteObject (hPen1) ;
DeleteObject (hPen2) ;
DeleteObject (hPen3) ;
```

这是建立、选择和删除画笔最直接的方法。但是您的程式必须知道执行期间需要哪些逻辑画笔, 为此, 您可能想要在每个 WM_PAINT 讯息处理期间建立画笔, 并在呼叫 EndPaint 之後删除它们 (您可以在呼叫 EndPaint 之前删除它们, 但是要小心, 不要删除装置内容中目前选择的画笔)。

您可能还希望随时建立画笔, 并将 CreatePen 和 SelectObject 呼叫组合到同一个叙述中:

```
SelectObject (hdc, CreatePen (PS_DASH, 0, RGB (255, 0, 0))) ;
```

现在再开始画线, 您将使用一个红色虚线画笔。在画完红色虚线之後, 可以删除画笔。糟了! 由於没有保存画笔代号, 怎么才能删除这些画笔呢? 不要紧, 请记住, SelectObject 将传回装置内容中上一次选择的画笔代号。所以, 您可以通过呼叫 SelectObject 将 BLACK_PEN 选进装置内容, 并删除从 SelectObject 传回的值:

```
DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
```

下面是另一种方法, 在将新建立的画笔选进装置内容时, 保存 SelectObject 传回的画笔代号:

```
hPen = SelectObject (hdc, CreatePen (PS_DASH, 0, RGB (255, 0, 0))) ;
```


现在 hPen 是什么呢？如果这是在取得装置内容之後第一次呼叫 SelectObject，则 hPen 是 BLACK_PEN 物件的代号。现在，可以将 hPen 选进装置内容，并删除所建立的画笔（第二次 SelectObject 呼叫传回的代号），只要一道叙述即可：

```
DeleteObject (SelectObject (hdc, hPen)) ;
```

如果有一个画笔的代号，就可以通过呼叫 GetObject 取得 LOGPEN 结构各个成员的值：

```
GetObject (hPen, sizeof (LOGPEN), (LPVOID) &logpen) ;
```

如果需要目前选进装置内容的画笔代号，可以呼叫：

```
hPen = GetCurrentObject (hdc, OBJ_PEN) ;
```

在第十七章将讨论另一个建立画笔的函式 ExtCreatePen。

填入空隙

使用点式画笔和虚线画笔会产生一个有趣的问题：点和虚线之间的空隙会怎样呢？您所需要的是什麼？

空隙的著色取决於装置内容的两个属性——背景模式和背景颜色。内定背景模式为 OPAQUE，在这种方式下，Windows 使用背景色来填入空隙，内定的背景色为白色。这与许多程式在视窗类别中用 WHITE_BRUSH 来擦除视窗背景的做法是一致的。

您可以通过如下呼叫来改变 Windows 用来填入空隙的背景色：

```
SetBkColor (hdc, crColor) ;
```

与画笔色彩所使用的 crColor 参数一样，Windows 将这里的背景色转换为纯色。可以通过用 GetBkColor 来取得装置内容中定义的目前背景色。

通过将背景模式转换为 TRANSPARENT，可以阻止 Windows 填入空隙：

```
SetBkMode (hdc, TRANSPARENT) ;
```

此後，Windows 将忽略背景色，并且不填入空隙，可以通过呼叫 GetBkMode 来取得目前背景模式（TRANSPARENT 或者 OPAQUE）。

绘图方式

装置内容中定义的绘图方式也影响显示器上所画线的外观。设想画这样一条直线，它的色彩由画笔色彩和画线区域原来的色彩共同决定。设想用同一种画笔在白色表面上画出黑线而在黑色表面上画出白线，而且不用知道表面是什麼色彩。这样的功能对您有用吗？通过绘图方式的设定，这些都可以实作。

当 Windows 使用画笔来画线时，它实际上执行画笔图素与目标位置处原来图素之间的某种位元布林运算。图素间的位元布林运算叫做「位元映射运算」，

简称为「ROP」。由於画一条直线只涉及两种图素（画笔和目标），因此这种布林运算又称为「二元位元映射运算」，简记为「ROP2」。Windows 定义了 16 种 ROP2 代码，表示 Windows 组合画笔图素和目标图素的方式。在内定装置内容中，绘图方式定义为 R2_COPYPEN，这意味著 Windows 只是将画笔图素复制到目标图素，这也是我们通常所熟知的。此外，还有 15 种 ROP2 码。

16 种不同的 ROP2 码是怎样得来的呢？为了示范的需要，我们假设使用单色系统，目标色（视窗显示区域的色彩）为黑色（用 0 来表示）或者白色（用 1 来表示），画笔也可以为黑色或者白色。用黑色或者白色画笔在黑色或者白色目标上画图有四种组合：白笔与白目标、白笔与黑目标、黑笔与白目标、黑笔与黑目标。

画笔在目标上绘制後会得到什么呢？一种可能是不管画笔和目标的色彩，画出的线总是黑色的，这种绘图方式由 ROP2 代码 R2_BLACK 表示。另一种可能是只有当画笔与目标都为黑色时，画出的结果才是白色，其他情况下画出的都是黑色。尽管这似乎有些奇怪，Windows 还是为这种方式起了一个名字，叫做 R2_NOTMERGEPEN。Windows 执行目标图素与画笔图素的位元「或」运算，然後翻转所得色彩。

表 5-2 显示了所有 16 种 ROP2 绘图方式，表中指示了画笔色彩(P)与目标色彩(D)是如何组合而成结果色彩的。在标有「布林操作」的那一栏中，用 C 语言的表示法给出了目标图素与画笔图素的组合方式。

表 5-2

画笔 (P)：目标 (D)：	1 1	1 0	0 1	0 0	布林 操作	绘图模式
结果：	0	0	0	0	0	R2_BLACK
	0	0	0	1	$\sim(P \mid D)$	R2_NOTMERGEPEN
	0	0	1	0	$\sim P \ \& \ D$	R2_MASKNOTPEN
	0	0	1	1	$\sim P$	R2_NOTCOPYPEN
	0	1	0	0	$P \ \& \ \sim D$	R2_MASKPENNOT
	0	1	0	1	$\sim D$	R2_NOT
	0	1	1	0	$P \ ^ \wedge \ D$	R2_XORPEN
	0	1	1	1	$\sim(P \ \& \ D)$	R2_NOTMASKPEN
	1	0	0	0	$P \ \& \ D$	R2_MASKPEN
	1	0	0	1	$\sim(P \ ^ \wedge \ D)$	R2_NOTXORPEN
	1	0	1	0	D	R2_NOP
	1	0	1	1	$\sim P \ \mid \ D$	R2_MERGEOTPEN
	1	1	0	0	P	R2_COPYPEN (内定)

	1	1	0	1	P ~D	R2_MERGEPENNOT
	1	1	1	0	P D	R2_MERGEPEN
	1	1	1	1	1	R2_WHITE

可以通过以下呼叫在装置内容中设定新的绘图模式：

```
SetROP2 (hdc, iDrawMode) ;
```

iDrawMode 参数是表中「绘图模式」一栏中给出的值之一。您可以用函式：

```
iDrawMode = GetROP2 (hdc) ;
```

来取得目前绘图方式。装置内容中的内定设定为 R2_COPYPEN，它用画笔色彩替代目标色彩。在 R2_NOTCOPYPEN 方式下，若画笔为黑色，则画成白色；若画笔为白色，则画成黑色。R2_BLACK 方式下，不管画笔和背景色为何种色彩，总是画成黑色。与此相反，R2_WHITE 方式下总是画成白色。R2_NOP 方式就是「不操作」，让目标保持不变。

现在，我们已经讨论了单色系统。然而，大多数系统是彩色的。在彩色系统中，Windows 为画笔和目标图素的每个颜色位元执行绘图方式的位元运算，并再次使用上表描述的 16 种 ROP2 代码。R2_NOT 绘图方式总是翻转目标色彩来决定线的颜色，而不管画笔的色彩是什么。例如，在青色目标上的线会变成紫色。R2_NOT 方式总是产生可见的画笔，除非画笔在中等灰度的背景上绘图。我将在第七章的 BLOKOUT 程式中展示 R2_NOT 绘图方式的使用。

绘制填入区域

现在再更进一步，从画线到画图形。Windows 中七个用来画带边缘的填入图形的函式列於表 5-3 中。

表 5-3

函式	图形
Rectangle	直角矩形
Ellipse	椭圆
RoundRect	圆角矩形
Chord	椭圆周上的弧，两端以弦连接
Pie	椭圆上的圆形图
Polygon	多边形
PolyPolygon	多个多边形

Windows 用装置内容中选择的目前画笔来画图形的边界框，边界框还使用目前背景方式、背景色彩和绘图方式，这跟 Windows 画线时一样。关于直线的一切也适用于这些图形的边界框。

图形以目前装置内容中选择的画刷来填入。内定情况下，使用现有物件，这意味著图形内部将画为白色。Windows 定义六种现有画刷：WHITE_BRUSH、LTGRAY_BRUSH、GRAY_BRUSH、DKGRAY_BRUSH、BLACK_BRUSH 和 NULL_BRUSH（也叫 HOLLOW_BRUSH）。您可以将任何一种现有画刷选入您的装置内容中，就和您选择一种画笔一样。Windows 将 HBRUSH 定义为画刷的代号，所以可以先定义一个画刷代号变数：

```
HBRUSH hBrush ;
```

您可以通过呼叫 GetStockObject 来取得 GRAY_BRUSH 的代号：

```
hBrush = GetStockObject (GRAY_BRUSH) ;
```

您可以呼叫 SelectObject 将它选进装置内容：

```
SelectObject (hdc, hBrush) ;
```

现在，如果您要画上表中的任一个图形，则其内部将为灰色。

如果您想画一个没有边界框的图形，可以将 NULL_PEN 选进装置内容：

```
SelectObject (hdc, GetStockObject (NULL_PEN)) ;
```

如果您想画出图形的边界框，但不填入内部，则将 NULL_BRUSH 选进装置内容：

```
SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
```

您也可以自订画刷，就如同您自订画笔一样。我们将马上谈到这个问题。

Polygon 函式和多边形填入方式

我已经讨论过了前五个区域填入函式，Polygon 是第六个画带边界框的填入图形的函式，该函式的呼叫与 Polyline 函式相似：

```
Polygon (hdc, apt, iCount) ;
```

其中，apt 参数是 POINT 结构的一个阵列，iCount 是点的数目。如果该阵列中的最後一个点与第一个点不同，则 Windows 将会再加一条线，将最後一个点与第一个点连起来（在 Polyline 函式中，Windows 不会这么做）。PolyPolygon 函式如下所示：

```
PolyPolygon (hdc, apt, aiCounts, iPolyCount) ;
```

该函式绘制多个多边形。最後一个参数给出了所画的多边形的个数。对于每个多边形，aiCounts 阵列给出了多边形的端点数。apt 阵列具有全部多边形的所有点。除传回值以外，PolyPolygon 在功能上与下面的代码相同：

```
for (i = 0, iAccum = 0 ; i < iPolyCount ; i++)
{
    Polygon (hdc, apt + iAccum, aiCounts[i]) ;
    iAccum += aiCounts[i] ;
}
```

对于 Polygon 和 PolyPolygon 函式，Windows 使用定义在装置内容中的目前

画刷来填入这个带边界的区域。至於填入内部的方式，则取决於多边形填入方式，您可以用 `SetPolyFillMode` 函式来设定：

```
SetPolyFillMode (hdc, iMode) ;
```

内定情况下，多边形填入方式是 `ALTERNATE`，但是您可以将它设定为 `WINDING`。两种方式的差别参见图 5-15 所示。

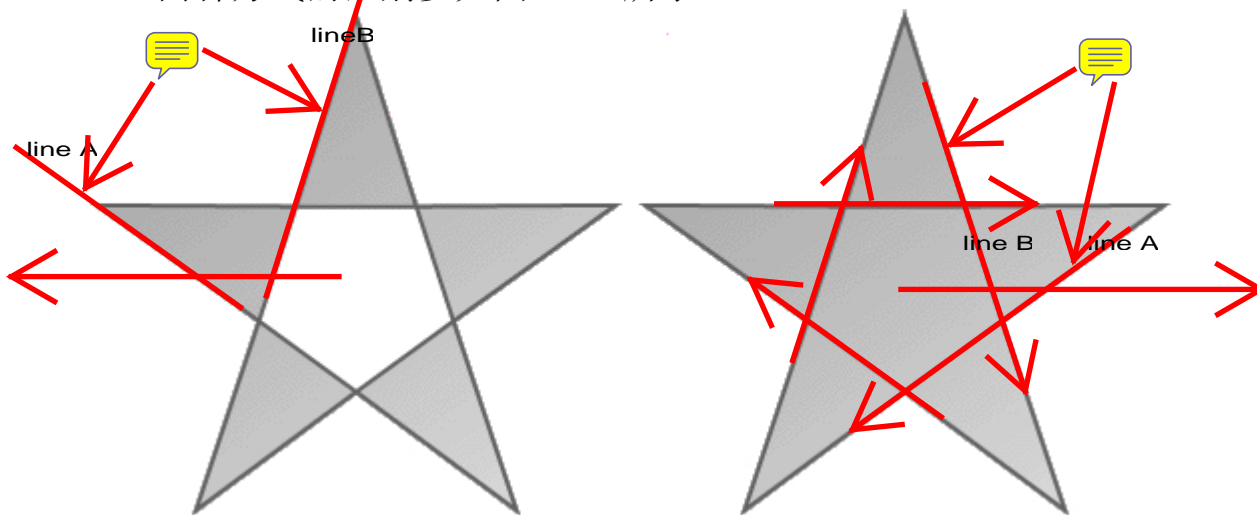


图 5-15 用两种多边形填入方式画出的图：`ALTERNATE`（左）和 `WINDING`（右）

首先，`ALTERNATE` 和 `WINDING` 方式之间的区别很容易察觉。對於 `ALTERNATE` 方式，您可以设想从一个无穷大的封闭区域内部的点画线，只有假想的线穿过了奇数条边界线时，才填入封闭区域。这就是填入了星的角而中心没被填入的原因。

五角星的例子使得 `WINDING` 方式看起来比实际上更简单一些。在绘制单个的多边形时，大多数情况下，`WINDING` 方式会填入所有封闭的区域。但是也有例外。

在 `WINDING` 方式下要确定一个封闭区域是否被填入，您仍旧可以设想从那个无穷大的区域画线。如果假想的线穿过了奇数条边界线，区域就被填入，这和 `ALTERNATE` 方式一样。如果假想的线穿过了偶数条边界线，则区域可能被填入也可能不被填入。如果一个方向（相对於假想线）的边界线数与另一个方向的边界线数不相等，就填入区域。

例如，考虑图 5-16 中的物体。线上的箭头指出了画线的方向。两种方式都会填入三个封闭的 L 形区域，号码从 1 到 3。号码为 4 和 5 的两个小内部区域，在 `ALTERNATE` 方式下不会被填入。但是，在 `WINDING` 方式下，号码为 5 的区域会被填入，因为从区域内必须穿过两条相同方向的线才能到达图形外部。号码为 4 的区域不会被填入，因为必须穿过两条方向相反的线。

如果您怀疑 Windows 没有这么聪明，那么程式 5-5 `ALTWIND` 会展示给您看。

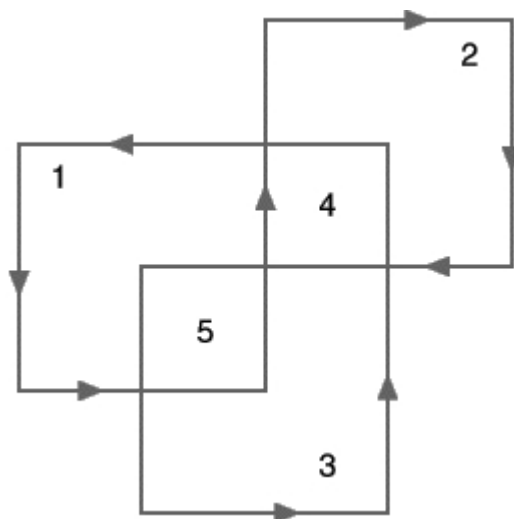


图 5-16 WINDING 方式不能填入所有内部区域的图形

程式 5-5 ALTWIND

```

ALTWIND.C
/*-----
    ALTWIND.C --      Alternate and Winding Fill Modes
                      (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("AltWind") ;
    HWND        hwnd ;
    MSG         msg ;
    WNDCLASS    wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;
        return 0 ;
    }
}

```

```
hWnd = CreateWindow (szAppName, TEXT ("Alternate and Winding Fill Modes"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hWnd, iCmdShow) ;
UpdateWindow (hWnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}

return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static POINT aptFigure [10] = {10,70, 50,70, 50,10, 90,10, 90,50,
                                    30,50, 30,90, 70,90, 70,30, 10,30 } ;
    static int  cxClient, cyClient ;
    HDC         hdc ;
    int         i ;
    PAINTSTRUCT ps ;
    POINT       apt[10] ;

    switch (message)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hWnd, &ps) ;

        SelectObject (hdc, GetStockObject (GRAY_BRUSH)) ;

        for (i = 0 ; i < 10 ; i++)
        {
            apt[i].x = cxClient * aptFigure[i].x / 200 ;
            apt[i].y = cyClient * aptFigure[i].y / 100 ;
        }

        SetPolyFillMode (hdc, ALTERNATE) ;
        Polygon (hdc, apt, 10) ;
    }
```

```
for (i = 0 ; i < 10 ; i++)
{
    apt[i].x += cxClient / 2 ;
}

SetPolyFillMode (hdc, WINDING) ;
Polygon (hdc, apt, 10) ;

EndPoint (hwnd, &ps) ;
return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图形的座标（划分为 100 100 个单位）储存在 aptFigure 阵列中。这些座标是依据显示区域的宽度和高度划分的。程式显示图形两次，一次使用 ALTERNATE 填入方式，另一次使用 WINDING 方式。结果见图 5-17。

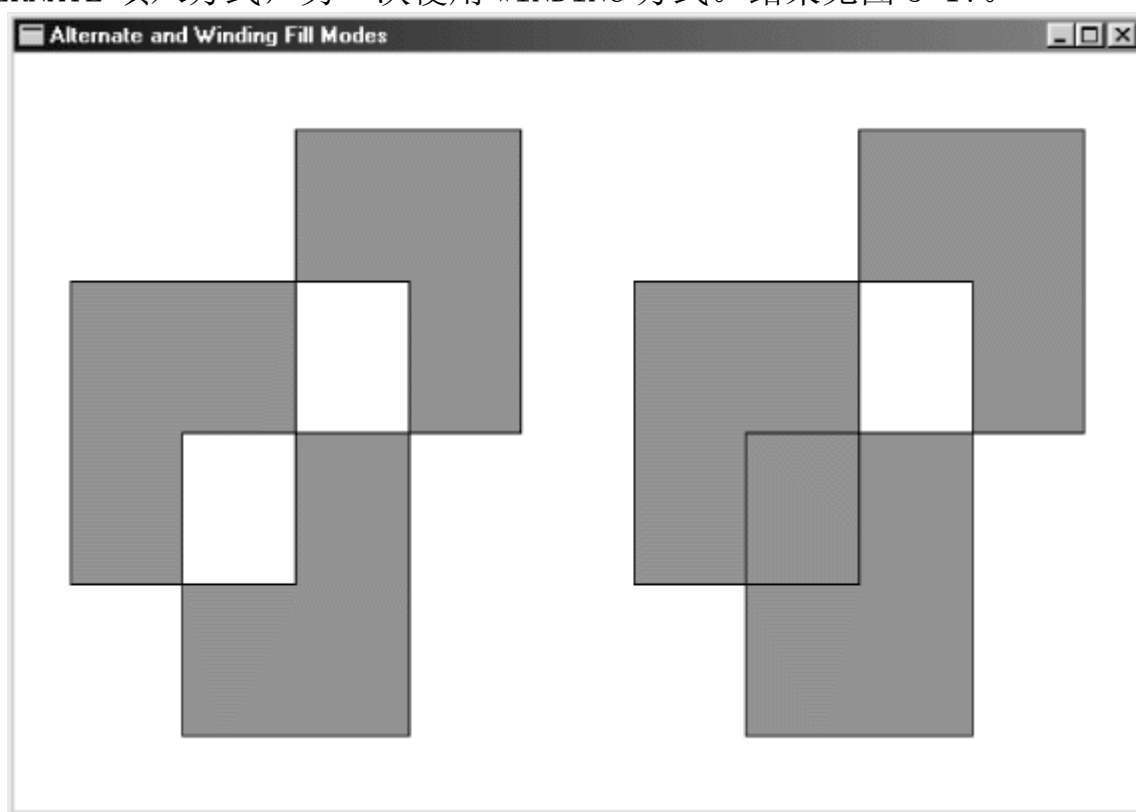


图 5-17 ALTWIND 的显示

用画刷填入内部

Rectangle、RoundRect、Ellipse、Chord、Pie、Polygon 和 PolyPolygon 图形的内部是用选进装置内容的目前画刷（也称为「图样」）来填入的。画刷

是一个 8 8 的点阵图，它水平和垂直地重复使用来填入内部区域。

当 Windows 用混色的方法来显示多於可从显示器上得到的色彩时，实际上是将画刷用於色彩。在单色系统上，Windows 能够使用黑色和白色图素的混色建立 64 种不同的灰色，更精确地说，Windows 能够建立 64 种不同的单色画刷。对於纯黑色，8 8 点阵图中的所有位元均为 0。第一种灰色有一位元为 1，第二种灰色有两位元为 1，以此类推，直到 8 8 点阵图中所有位元均为 1，这就是白色。在 16 色或 256 色显示系统上，混色也是点阵图，并且可以得到更多的色彩。

Windows 还有五个函式，可以让您建立逻辑画刷，然後就可使用 SelectObject 将画刷选进装置内容。与逻辑画笔一样，逻辑画刷也是 GDI 物件。您建立的所有画刷都必须被删除，但是当它还在装置内容中时不能将其删除。

下面是建立逻辑画刷的第一个函式：

```
hBrush = CreateSolidBrush (crColor) ;
```

函式中的 Solid 并不是指画刷为纯色。在将画刷选入装置内容中时，Windows 建立一个混色色的点阵图，并为画刷使用该点阵图。

您还可以使用由水平、垂直或者倾斜的线组成的「影线标记(hatch marks)」来建立画刷，这种风格的画刷对著色条形图的内部和在绘图机上进行绘图最有用。建立影线画刷的函式为：

```
hBrush = CreateHatchBrush (iHatchStyle, crColor) ;
```

iHatchStyle 参数描述影线标记的外观。图 5-18 显示了六种可用的影线标记风格。

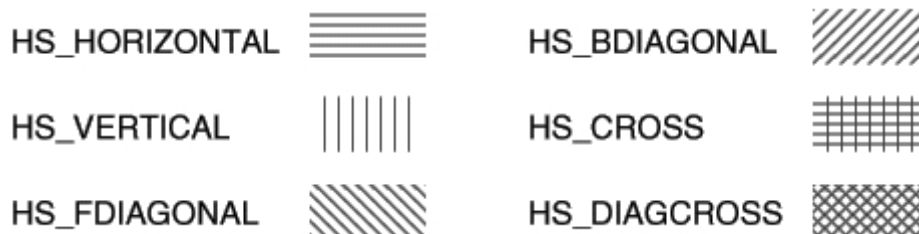


图 5-18 六种影线画刷风格

CreateHatchBrush 中的 crColor 参数是影线的色彩。在将画刷选进装置内容时，Windows 将这种色彩转换为与之最相近的纯色。影线之间的区域根据装置内容中定义的背景方式和背景色来著色。如果背景方式为 OPAQUE，则用背景色（它也被转换为纯色）来填入线之间的空间。在这种情况下，影线和填入色都不能是混色而成的颜色。如果背景方式为 TRANSPARENT，则 Windows 只画出影线，不填入它们之间的区域。

您也可以使用 CreatePatternBrush 和 CreateDIBPatternBrushPt 建立自己的点阵图画刷。

建立逻辑画刷的第五个函式包含其他四个函式：

```
hBrush = CreateBrushIndirect (&logbrush) ;
```


变数 logbrush 是一个型态为 LOGBRUSH (「逻辑画刷」) 的结构, 该结构的三个栏位如表 5-4 所示, lbStyle 栏位的值确定了 Windows 如何解释其他两个栏位的值:

表 5-4

lbStyle (UINT)	lbColor (COLORREF)	lbHatch (LONG)
BS_SOLID	画刷的色彩	忽略
BS_HOLLOW	忽略	忽略
BS_HATCHED	影线的色彩	影线画刷风格
BS_PATTERN	忽略	点阵图的代号
BS_DIBPATTERNPT	忽略	指向 DIB 的指标

前面我们用 SelectObject 将逻辑画笔选进装置内容, 用 DeleteObject 删除画笔, 用 GetObject 来取得逻辑画笔的资讯。对于画刷, 同样能使用这三个函式。一旦您取得了画刷代号, 就可以使用 SelectObject 将该画刷选进装置内容:

```
SelectObject (hdc, hBrush) ;
```

然後, 您可以使用 DeleteObject 函式删除所建立的画刷:

```
DeleteObject (hBrush) ;
```

但是, 不要删除目前选进装置内容的画刷。

如果您需要取得画刷的资讯, 可以呼叫 GetObject:

```
GetObject (hBrush, sizeof (LOGBRUSH), (LPVOID) &logbrush) ;
```

其中, logbrush 是一个型态为 LOGBRUSH 的结构。

GDI 映射方式

到目前为止, 所有的程式都是相对於显示区域的左上角, 以图素为单位绘图的。这是内定情况, 但不是唯一选择。事实上, 「映射方式」是一种几乎影响任何显示区域绘图的装置内容属性。另外有四种装置内容属性——视窗原点、视埠原点、视窗范围和视埠范围——与映射方式密切相关。

大多数 GDI 绘图函式需要座标值或大小。例如, 下面是 TextOut 函式:

```
TextOut (hdc, x, y, psText, iLength) ;
```

参数 x 和 y 分别表示文字的开始位置。参数 x 是在水平轴上的位置, 参数 y 是在垂直轴上的位置, 通常用 (x, y) 来表示这个点。

在 TextOut 中, 以及在几乎所有 GDI 函式中, 这些座标值使用的都是一种「逻辑单位」。Windows 必须将逻辑单位转换为「装置单位」, 即图素。这种转换是由映射方式、视窗和视埠的原点以及视窗和视埠的范围所控制的。映射方式还指示著 x 轴和 y 轴的方向 (orientation); 也就是说, 它确定了当您在向显

示器的左或者右移动时 x 的值是增大还是减小，以及在上下移动时 y 的值是增大还是减小。

Windows 定义了 8 种映射方式，它们在 WINGDI.H 中相应的识别字和含义如表 5-5 所示。

表 5-5

映射方式	逻辑单位	增加值	
		x 值	y 值
MM_TEXT	图素	右	下
MM_LOMETRIC	0.1 mm	右	上
MM_HIMETRIC	0.01 mm	右	上
MM_LOENGLISH	0.01 in.	右	上
MM_HIENGLISH	0.001 in.	右	上
MM_TWIPS	1/1440 in.	右	上
MM_ISOTROPIC	任意 (x = y)	可选	可选
MM_ANISOTROPIC	任意 (x != y)	可选	可选

METRIC 和 ENGLISH 指一般通行的度量衡系统，点是印刷的测量单位，约等於 1/72 英寸，但在图形程式设计中假定为正好 1/72 英寸。「Twip」等於 1/20 点，也就是 1/1440 英寸。「Isotropic」和「anisotropic」是真正的单字，意思是「等方性」（同方向）和「异方性」（不同方向）。

您可以使用下面的叙述来设定映射方式：

```
SetMapMode (hdc, iMapMode) ;
```

其中，iMapMode 是 8 个映射方式识别字之一。您可以通过以下呼叫取得目前的映射方式：

```
iMapMode = GetMapMode (hdc) ;
```

内定映射方式为 MM_TEXT。在这种映射方式下，逻辑单位与实际单位相同，这样我们可以直接以图素为单位进行操作。在 TextOut 呼叫中，它看起来像这样：

```
TextOut (hdc, 8, 16, TEXT ("Hello"), 5) ;
```

文字从距离显示区域左端 8 图素、上端 16 图素的位置处开始。

如果映射方式设定为 MM_LOENGLISH：

```
SetMapMode (hdc, MM_LOENGLISH) ;
```

则逻辑单位是百分之一。现在，TextOut 呼叫如下：

```
TextOut (hdc, 50, -100, TEXT ("Hello"), 5) ;
```

文字从距离显示区域左端 0.5 英寸、上端 1 英寸的位置处开始。至於 y 座标前面的负号，随著我们对映射方式更详细的讨论，将逐渐清楚。其他映射方

式允许程式按照毫米、印表机的点大小或者任意单位的座标轴来指定座标。

如果您认为使用图素进行工作很合适，那么就不要再使用内定的 MM_TEXT 方式外的任何映射方式。如果需要以英寸或者毫米尺寸显示图像，那么可以从 GetDeviceCaps 中取得所需要的资讯，自己再进行缩放。其他映射方式都是避免您自己进行缩放的一个方便途径而已。

虽然您在 GDI 函式中指定的座标是 32 位元的值，但是仅有 Windows NT 能够处理全 32 位元。在 Windows 98 中，座标被限制为 16 位元，范围从 -32,768 到 32,767。一些使用座标表示矩形的开始点和结束点的 Windows 函式也要求矩形的宽和高小於或者等於 32,767。

装置座标和逻辑座标

您也许会问：如果使用 MM_LOENGLISH 映射方式，是不是将会得到以百分之一英寸为单位的 WM_SIZE 讯息呢？绝对不会。Windows 对所有讯息（如 WM_MOVE、WM_SIZE 和 WM_MOUSEMOVE），对所有非 GDI 函式，甚至对一些 GDI 函式，永远使用装置座标。可以这样来考虑：由於映射方式是一种装置内容属性，所以，只有对需要装置内容代号作参数的 GDI 函式，映射方式才会起作用。GetSystemMetrics 不是 GDI 函式，所以它总是以装置单位（即图素）为量度来传回大小的。尽管 GetDeviceCaps 是 GDI 函式，需要一个装置内容代号作为参数，但是 Windows 仍然对 HORZRES 和 VERTRES 以装置单位作为传回值，因为该函式的目的之一就是给程式提供以图素为单位的设备大小。

不过，从 GetTextMetrics 呼叫中传回的 TEXTMETRIC 结构的值是使用逻辑单位的。如果在进行此呼叫时映射方式为 MM_LOENGLISH，则 GetTextMetrics 将以百分之一英寸为单位提供字元的宽度和高度。在呼叫 GetTextMetrics 以取得关于字元的宽度和高度资讯时，映射方式必须设定成根据这些资讯输出文字时所使用的映射方式，这样就可以简化工作。

装置座标系

Windows 将 GDI 函式中指定的逻辑座标映射为装置座标。在讨论以各种不同的映射方式使用逻辑座标系之前，我们先来看一下 Windows 为视讯显示器区域定义的不同的装置座标系。尽管我们大多数时间在视窗的显示区域内工作，但 Windows 在不同的时间使用另外两种装置座标区域。所有装置座标系都以图素为单位，水平轴（即 x 轴）上的值从左到右递增，垂直轴（即 y 轴）上的值从上到下递增。

当我们使用整个萤幕时，就根据「萤幕座标」进行操作。萤幕的左上角为

(0,0)点，萤幕座标用在 WM_MOVE 讯息（对于非子视窗）以及下列 Windows 函式中：CreateWindow 和 MoveWindow（都是对于非子视窗）、GetMessagePos、GetCursorPos、SetCursorPos、GetWindowRect 以及 WindowFromPoint（这不是全部函式的列表）。它们或者是与视窗无关的函式（如两个游标函式），或者是必须相对于某个萤幕点来移动（或者寻找）视窗的函式。如果以 DISPLAY 为参数呼叫 CreateDC，以取得整个萤幕的装置内容，则内定情况下 GDI 呼叫中指定的逻辑座标将被映射为萤幕座标。

「全视窗座标」以程式的整个视窗为基准，如标题列、功能表、卷动列和视窗框都包括在内。而对于普通视窗，点 (0,0) 是缩放边框的左上角。全视窗座标在 Windows 中极少使用，但是如果用 GetWindowDC 取得装置内容，GDI 函式中的逻辑座标就会转换为显示区域座标。

第三种坐标系是我们最常使用的「显示区域坐标系」。点 (0,0) 是显示区域的左上角。当使用 GetDC 或 BeginPaint 取得装置内容时，GDI 函式中的逻辑座标就会内定转换为显示区域座标。

用函式 ClientToScreen 和 ScreenToClient 可以将显示区域座标转换为萤幕座标，或者反过来，将萤幕座标转换为显示区域座标。也可以使用 GetWindowRect 函式取得萤幕座标下的整个视窗的位置和大小。这三个函式为一种装置座标转换为另一种提供了足够的资讯。

视埠和视窗

映射方式定义了 Windows 如何将 GDI 函式中指定的逻辑座标映射为装置座标，这里的装置座标系取决于您用哪个函式来取得装置内容。要继续讨论映射方式，我们需要一些术语：映射方式用于定义从「视窗」（逻辑座标）到「视埠」（装置座标）的映射。

「视窗」和「视埠」这两个词用得并不恰当。在其他图形介面语言中，视埠通常包含有剪裁区域的意思，并且，我们已经用视窗来指程式在萤幕上占据的区域。在这里的讨论中，我们必须把关于这些词的先入之见丢到一边。

「视埠」是依据装置座标（图素）的。通常，视埠和显示区域相同，但是，如果您已经用 GetWindowDC 或 CreateDC 取得了一个装置内容，则视埠也可以是指整视窗座标或者萤幕座标。点 (0,0) 是显示区域（或者整个视窗或萤幕）的左上角，x 的值向右增加，y 的值向下增加。

「视窗」是依据逻辑座标的，逻辑座标可以是图素、毫米、英寸或者您想要的任何其他单位。您在 GDI 绘图函式中指定逻辑视窗座标。

但是在真正的意义上，视埠和视窗仅是数学上的概念。对于所有的映射方

式, Windows 都用下面两个公式来将视窗 (逻辑) 座标转化为视埠 (设备) 座标:

$$xViewport = (xWindow - xWinOrg) \times \frac{xViewExt}{xWinExt} + xViewOrg$$

$$yViewport = (yWindow - yWinOrg) \times \frac{yViewExt}{yWinExt} + yViewOrg$$

其中, (xWindow, yWindow) 是待转换的逻辑点, (xViewport, yViewport) 是转换後的装置座标点, 一般情形下差不多就是显示区域座标了。

这两个公式使用了分别指定视窗和视埠「原点」的点: (xWinOrg, yWinOrg) 是逻辑座标的视窗原点; (xViewOrg, yViewOrg) 是装置座标的视埠原点。在内定的装置内容中, 这两个点均被设定为 (0, 0), 但是它们可以改变。此公式意味著, 逻辑点 (xWinOrg, yWinOrg) 总被映射为装置点 (xViewOrg, yViewOrg)。如果视窗和视埠的原点是预设值 (0, 0), 则公式简化为:

$$xViewport = xWindow \times \frac{xViewExt}{xWinExt}$$

$$yViewport = yWindow \times \frac{yViewExt}{yWinExt}$$

此公式还使用了两点来指定「范围」: (xWinExt, yWinExt) 是逻辑座标的视窗范围; (xViewExt, yViewExt) 是装置座标的视窗范围。在多数映射方式中, 范围是映射方式所隐含的, 不能够改变。每个范围自身没有什么意义, 但是视埠范围与视窗范围的比例是逻辑单位转换为装置单位的换算因数。

例如, 当您设定 MM_LOENGLISH 映射方式时, Windows 将 xViewExt 设定为某个图素数而将 xWinExt 设定为 xViewExt 图素占据的一英寸内有几百图素的长度。比值给出了一英寸内有几百个图素的数值。为了提高转换效能, 换算因数表示为整数比而不是浮点数。

范围可以为负, 也就是说, 逻辑 x 轴上的值不一定非得在向右时增加; 逻辑 y 轴上的值不一定非得在向下时增加。

Windows 也能将视埠 (设备) 座标转换为视窗 (逻辑) 座标:

$$xWindow = (xViewport - xViewOrg) \times \frac{xWinExt}{xViewExt} + xWinOrg$$

$$yWindow = (yViewport - yViewOrg) \times \frac{yWinExt}{yViewExt} + yWinOrg$$

Windows 提供了两个函式来让您将装置点转换为逻辑点以及将逻辑点转换为装置点。下面的函式将装置点转换为逻辑点：

```
DPToLP (hdc, pPoints, iNumber) ;
```

其中，pPoints 是一个指向 POINT 结构阵列的指标，而 iNumber 是要转换的点的个数。您会发现这个函式对于将 GetClientRect（它总是使用装置单位）取得的显示区域大小转换为逻辑座标很有用：

```
GetClientRect (hwnd, &rect) ;  
DPToLP (hdc, (PPOINT) &rect, 2) ;
```

下面的函式将逻辑点转换为装置点：

```
LPtoDP (hdc, pPoints, iNumber) ;
```

处理 MM_TEXT

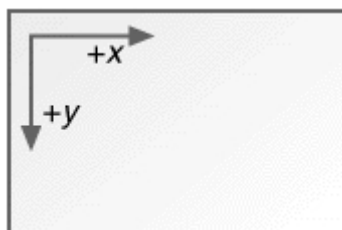
对于 MM_TEXT 映射方式，内定的原点和范围如下所示：

- 视窗原点：(0, 0) 可以改变
- 视埠原点：(0, 0) 可以改变
- 视窗范围：(1, 1) 不可改变
- 视埠范围：(1, 1) 不可改变

视埠范围与视窗范围的比例为 1，所以不用在逻辑座标与装置座标之间进行缩放。上面所给出的公式可以简化为：

$$\begin{aligned} x_{\text{Viewport}} &= x_{\text{Window}} - x_{\text{WinOrg}} + x_{\text{ViewOrg}} \\ y_{\text{Viewport}} &= y_{\text{Window}} - y_{\text{WinOrg}} + y_{\text{ViewOrg}} \end{aligned}$$

这种映射方式称为「文字」映射方式，不是因为它对于文字最适合，而是由于轴的方向。我们读文字是从左至右，从上至下的，而 MM_TEXT 以同样的方向定义轴上值的增长方向：



Windows 提供了函式 SetViewportOrgEx 和 SetWindowOrgEx，用来改变视埠和视窗的原点，这些函式都具有改变轴的效果，以致 (0, 0) 不再指左上角。一般来说，您会使用 SetViewportOrgEx 或 SetWindowOrgEx 之一，但不会同时使用二者。

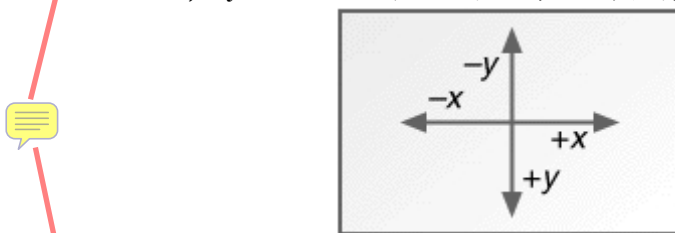
我们来看一看这些函式有何效果：如果将视埠原点改变为

$(xViewOrg, yViewOrg)$, 则逻辑点 $(0,0)$ 就会映射为装置点 $(xViewOrg, yViewOrg)$ 。如果将视窗原点改变为 $(xWinOrg, yWinOrg)$, 则逻辑点 $(xWinOrg, yWinOrg)$ 将会映射为装置点 $(0,0)$, 即左上角。不管对视窗和视埠原点作什么改变, 装置点 $(0,0)$ 始终是显示区域的左上角。

例如, 假设显示区域为 $cxClient$ 个图素宽和 $cyClient$ 个图素高。如果想将逻辑点 $(0,0)$ 定义为显示区域的中心, 可进行如下呼叫:

```
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

`SetViewportOrgEx` 的参数总是使用装置单位。现在, 逻辑点 $(0,0)$ 将映射为装置点 $(cxClient/2, cyClient/2)$, 而显示区域的座标系变成如下形状:



逻辑 x 轴的范围从 $-cxClient/2$ 到 $+cxClient/2$, 逻辑 y 轴的范围从 $-cyClient/2$ 到 $+cyClient/2$, 显示区域的右下角为逻辑点 $(cxClient/2, cyClient/2)$ 。如果您想从显示区域的左上角开始显示文字。则需要使用负座标:

```
TextOut (hdc, -cxClient / 2, -cyClient / 2, "Hello", 5) ;
```

用下面的 `SetWindowOrgEx` 叙述可以获得与上面使用 `SetViewportOrgEx` 同样的效果:

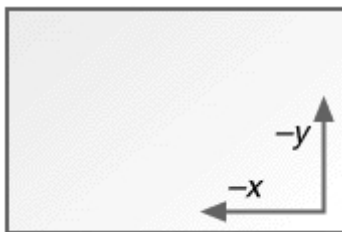
```
SetWindowOrgEx (hdc, -cxClient / 2, -cyClient / 2, NULL) ;
```

`SetWindowOrgEx` 的参数总是使用逻辑单位。在这个呼叫之後, 逻辑点 $(-cxClient / 2, -cyClient / 2)$ 映射为装置点 $(0,0)$, 即显示区域的左上角。

您不会将这两个函式一起用, 除非您知道这么做的结果:

```
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
SetWindowOrgEx (hdc, -cxClient / 2, -cyClient / 2, NULL) ;
```

这意味著逻辑点 $(-cxClient/2, -cyClient/2)$ 将映射为装置点 $(cxClient/2, cyClient/2)$, 结果是如下所示的座标系:



您可以使用下面两个函式取得目前视埠和视窗的原点:

```
GetViewportOrgEx (hdc, &pt) ;
GetWindowOrgEx (hdc, &pt) ;
```

其中 pt 是 `POINT` 结构。由 `GetViewportOrgEx` 传回的值是装置座标, 而由

GetWindowOrgEx 传回的值是逻辑坐标。

您可能想改变视埠或者视窗的原点，以改变视窗显示区域内的显示输出——例如，回应使用者在卷动列内的输入。但是，改变视埠和视窗原点并不能立即改变显示输出，而必须在改变原点之後更新输出。例如，在第四章的 SYSMETS2 程式中，我们使用了 iVscrollPos 值（垂直卷动列的目前位置）来调整显示输出的 y 坐标：

```
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    for (i = 0 ; i < NUMLINES ; i++)
    {
        y = cyChar * (i - iVscrollPos) ;
        // 显示文字
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
```

我们可以使用 SetWindowOrgEx 获得同样的效果：

```
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetWindowOrgEx (hdc, 0, cyChar * iVscrollPos) ;

    for (i = 0 ; i < NUMLINES ; i++)
    {
        y = cyChar * i ;
        // 显示文字
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
```

现在，TextOut 函式的 y 坐标的计算不需要 iVscrollPos 的值。这意味著您可以将文字输出函式放到一个常式中，不用将 iVscrollPos 值传给该常式，因为我们是通过改变视窗原点来调整文字显示的。

如果您有使用直角坐标系（即笛卡尔坐标系）的经验，那么将逻辑点 (0, 0) 移到显示区域的中央（像我们上面所说的那样）的确值得考虑。但是，对于 MM_TEXT 映射方式来说，还存在著一个小小的问题：笛卡尔坐标系中，y 值是随著上移而增加的，而 MM_TEXT 定义为下移时 y 值增加。从这一点来看，MM_TEXT 有点古怪，而下面这五种映射方式都使用通常的增值方法。

「度量」映射方式

Windows 包含五种以实际尺寸来表示逻辑坐标的映射方式。由於 x 轴和 y 轴

的逻辑座标映射为相同的实际单位，这些映射方式能使您画出不变形的圆和矩形。

这五种「度量」映射方式在表 5-6 中列出，按照从低精度到高精度的顺序排列。右边的两列分别给出了以英寸和毫米为单位时逻辑单位的大小，以便比较。

表 5-6

映射方式	逻辑单位	英寸	毫米
MM_LOENGLISH	0.01 in.	0.01	0.254
MM_LOMETRIC	0.1 mm.	0.00394	0.1
MM_HIENGLISH	0.001 in.	0.001	0.0254
MM_TWIPS	1/1400 in.	0.000694	0.0176
MM_HIMETRIC	0.01 mm.	0.000394	0.01

内定视窗及视埠的原点和范围如下所示：

- 视窗原点：(0, 0) 可以改变
- 视埠原点：(0, 0) 可以改变
- 视窗范围：(1, 1) 不可改变
- 视埠范围：(1, 1) 不可改变

问号表示视窗和视埠的范围依赖於映射方式和设备的解析度。前面已经提到过，这些范围本身并不重要，但是表示比例时就必须知道。下面是视窗座标到视埠座标的转换公式：

$$xViewport = (xWindow - xWinOrg) \times \frac{xViewExt}{xWinExt} + xViewOrg$$

$$yViewport = (yWindow - yWinOrg) \times \frac{yViewExt}{yWinExt} + yViewOrg$$

例如，對於 MM_LOENGLISH，Windows 计算的范围如下：

$$\frac{xViewExt}{xWinExt} = 0.01 \text{ in 中的水平圖素數}$$

$$\frac{-yViewExt}{yWinExt} = 0.01 \text{ in 中的垂直圖素數}$$

Windows 使用这些来自 GetDeviceCaps 的有用资讯设定范围。只是在 Windows 98 和 Windows NT 之间有一点差别。

首先，来看看 Windows 98 是如何做的：假设您使用「控制台」的「显示」

程式选择了 96 dpi 的系统字体。GetDeviceCaps 对於 LOGPIXELSX 和 LOGPIXELSY 索引都将传回值 96。Windows 为视埠范围使用这些值并以表 5-7 的方式设定视埠和视窗的范围。

表 5-7

映射方式	视埠范围 (x, y)	视窗范围 (x, y)
MM_LOMETRIC	(96, 96)	(254, -254)
MM_HIMETRIC	(96, 96)	(2540, -2540)
MM_LOENGLISH	(96, 96)	(100, -100)
MM_HIENGLISH	(96, 96)	(1000, -1000)
MM_TWIPS	(96, 96)	(1440, -1440)



这样，对 MM_LOENGLISH 来说，96 除以 100 的比值是 0.01 英寸中的图素数。对 MM_LOMETRIC 来说，96 除以 254 的比值是 0.1 毫米中的图素数。

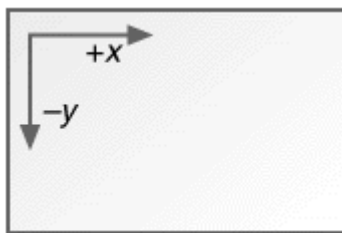
Windows NT 使用不同的方法设定视埠和视窗的范围（与早期 16 位元版本的 Windows 一致的方法）。视埠范围依据萤幕的图素尺寸。可以使用 HORZRES 和 VERTRES 索引从 GetDeviceCaps 取得这种资讯。视窗范围依据假定的显示大小，它是您使用 HORZSIZE 和 VERTSIZE 索引时由 GetDeviceCaps 传回的。我在前面提到过，这些值一般是 320 和 240 毫米。如果您将显示器的图素尺寸设定为 1024 768，则表 5-8 就是 Windows NT 报告的视埠和视窗范围的值。

表 5-8

映射方式	视埠范围 (x, y)	视窗范围 (x, y)
MM_LOMETRIC	(1024, -768)	(3, 200, 2, 400)
MM_HIMETRIC	(1024, -768)	(32, 000, 24, 000)
MM_LOENGLISH	(1024, -768)	(1, 260, 945)
MM_HIENGLISH	(1024, -768)	(12, 598, 9, 449)
MM_TWIPS	(1024, -768)	(18, 142, 13, 606)

这些视窗范围表示包含显示器全部宽度和高度的逻辑单位元数值。320 毫米宽的萤幕也为 1260 MM_LOENGLISH 单位或 12.6 英寸(320 除以 25.4 毫米/英寸)。

范围中，y 前面的负号表示改变了轴的方向。对於这五种映射方式，y 值随上升而增加，然而注意内定的视窗和视埠原点均为 (0, 0)。这个事实有一个有趣的结果。当一开始改变为五种映射方式之一时，座标系如下：



要想在显示区域显示任何东西，必须使用负的 y 值。例如下面的程式码：

```
SetMapMode (hdc, MM_LOENGLISH) ;
TextOut (hdc, 100, -100, "Hello", 5) ;
```

将把文字显示在距离显示区域左边和上边各一英寸的地方。

为了使自己保持头脑清醒，您可能想避免这样做。一种解决办法是将逻辑的 (0, 0) 点设为显示区域的左下角，您可以通过呼叫 SetViewportOrgEx 来完成（假设 cyClient 是以图素为单位的显示区域的高度）：

```
SetViewportOrgEx (hdc, 0, cyClient, NULL) ;
```

此时的坐标系如下：

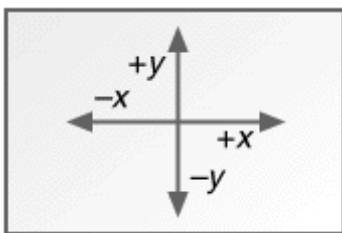


这是直角坐标系的右上象限。

另一种方法是将逻辑 (0, 0) 点设为显示区域的中心：

```
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

此时的坐标系如下所示：



现在，我们有了一个真正的 4 象限笛卡尔坐标系，在 x 轴和 y 轴上有相等的按英寸、毫米或 twip 计算的逻辑单位。

您还可以使用 SetWindowOrgEx 函式来改变逻辑 (0, 0) 点，但是这稍微困难一些，因为 SetWindowOrgEx 的参数必须使用逻辑单位，先要将 (cxClient, cyClient) 用 DpToLP 函式转换为逻辑坐标。假设变数 pt 是型态为 POINT 的结构，下面的代码将逻辑 (0, 0) 点改变到显示区域的中央：

```
pt.x = cxClient ;
pt.y = cyClient ;
DpToLP (hdc, &pt, 1) ;
SetWindowOrgEx (hdc, -pt.x / 2, -pt.y / 2, NULL) ;
```

「自行决定」的映射方式

剩下的两种映射方式为 `MM_ISOTROPIC` 和 `MM_ANISOTROPIC`。只有这两种映射方式可以让您改变视埠和视窗范围，也就是说可以改变 Windows 用来转换逻辑和装置座标的换算因数。「isotropic」的意思是「同方向性」；「anisotropic」的意思是「异方向性」。与上面所讨论的度量映射方式相似，`MM_ISOTROPIC` 使用相同的轴，x 轴上的逻辑单位与 y 轴上的逻辑单位的实际尺寸相等。这对您建立纵横比与显示比无关的图像是有帮助的。

`MM_ISOTROPIC` 与度量映射方式之间的区别是，使用 `MM_ISOTROPIC`，您可以控制逻辑单位的实际尺寸。如果愿意，您可以根据显示区域的大小来调整逻辑单位的实际尺寸，从而使所画的图像总是包含在显示区域内，并相应地放大或缩小。例如，第八章的两个时钟程式就是方向同性的例子。在您改变视窗大小时，时钟也相应地调整。

Windows 程式完全可以通过调整视窗和视埠范围来处理图像大小的变化。因此，不管视窗尺寸怎样变，程式都可以在绘图函数中使用相同的逻辑单位。

有时候 `MM_TEXT` 和度量映射方式称为「完全局限性」映射方式，这就是说，您不能改变视窗和视埠的范围以及 Windows 将逻辑座标换算为装置座标的方法。`MM_ISOTROPIC` 是一种「半局限性」的映射方式，Windows 允许您改变视窗和视埠范围，但只是调整它们，以便 x 和 y 逻辑单位代表同样的实际尺寸。`MM_ANISOTROPIC` 映射方式是「非局限性」的，您可以改变视窗和视埠范围，但是 Windows 不调整这些值。

MM_ISOTROPIC 映射方式

如果想要在使用任意的轴时都保证两个轴上的逻辑单位相同，则 `MM_ISOTROPIC` 映射方式就是理想的映射方式。这时，具有相同逻辑宽度和高度的矩形显示为正方形，具有相同逻辑宽度和高度的椭圆显示为圆。

当您刚开始将映射方式设定为 `MM_ISOTROPIC` 时，Windows 使用与 `MM_LOMETRIC` 同样的视窗和视埠范围（但是，不要对此有所依赖）。区别在於，您现在可以呼叫 `SetWindowExtEx` 和 `SetViewportExtEx` 来根据自己的偏好改变范围了，然後，Windows 将调整范围的值，以便两条轴上的逻辑单位有相同的实际距离。

一般说来，您可以用所期望的逻辑视窗的逻辑尺寸作为 `SetWindowExtEx` 的参数，用显示区域的实际宽和高作为 `SetViewportExtEx` 的参数。Windows 在调整这些范围时，必须让逻辑视窗适应实际视窗，这就有可能导致显示区域的一

段落到了逻辑视窗的外面。必须在呼叫 `SetViewportExtEx` 之前呼叫 `SetWindowExtEx`，以便最有效地使用显示区域中的空间。

例如，假设您想要一个「传统的」单象限虚拟坐标系，其中 (0, 0) 在显示区域的左下角，宽度和高度的范围都是从 0 到 32,767，并且希望 x 和 y 轴的单位具有同样的实际尺寸。以下就是所需的程式：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetViewportOrgEx (hdc, 0, cyClient, NULL) ;
```

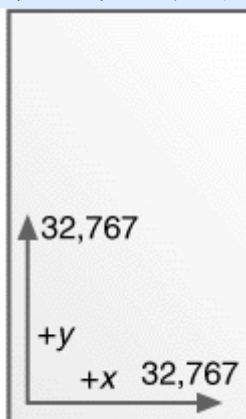
如果其後用 `GetWindowExtEx` 和 `GetViewportExtEx` 函式获得了视窗和视埠的范围，可以发现，它们并不是先前指定的值。Windows 将根据显示设备的纵横比来调整范围，以便两条轴上的逻辑单位表示相同的实际尺寸。

如果显示区域的宽度大於高度（以实际尺寸为准），Windows 将调整 x 的范围，以便逻辑视窗比显示区域视埠窄。这样，逻辑视窗将放置在显示区域的左边：



Windows 98 不允许在显示区域的右边超越 x 轴的范围之外显示任何东西，因为这需要一个大於 16 位元所能表示的坐标。Windows NT 使用全 32 位元坐标，您可以在超出右边显示一些东西。

如果显示区域的高度大於宽度（以实际尺寸为准），那么 Windows 将调整 y 的范围。这样，逻辑视窗将放置在显示区域的下边：



Windows 98 不允许在显示区域的顶部显示任何东西。

如果您希望逻辑视窗总是放在显示区域的左上部，那么将前面给出的程式码改为：

```
SetMapMode (MM_ISOTROPIC) ;
```



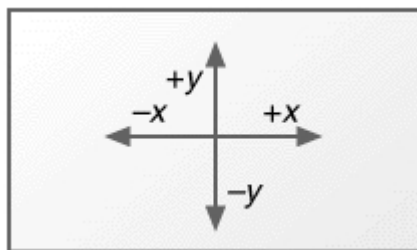
```
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetWindowOrgEx (hdc, 0, 32767, NULL) ;
```

在呼叫 SetWindowOrgEx 中，我们要求将逻辑点 (0, 32767) 映射为装置点 (0, 0)。现在，如果显示区域的高大於宽，则座标系将安排为：

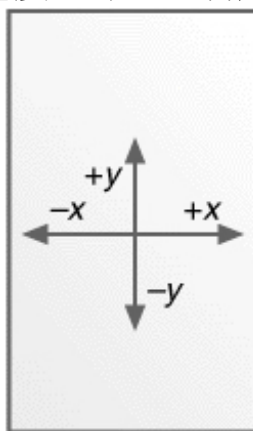
对於时钟程式，您也许想要使用一个四象限的笛卡尔座标系，四个方向的座标尺度可以任意指定，(0, 0) 必须居於显示区域的中央。如果您想要每条轴的范围从 0 到 1000，则可以使用以下程式码：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 1000, 1000, NULL) ;
SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

如果显示区域的宽度大於高度，则逻辑座标系形如：



如果显示区域的高度大於宽度，那么逻辑座标也会居中：



记住，视窗或者视埠范围并不意味着要进行剪裁。在呼叫 GDI 函式时，您仍然对以随便地使用小於-1000 和大於 1000 的 x 和 y 值。根据显示区域的外形，这些点可能看得见，也可能看不见。

在 MM_ISOTROPIC 映射方式下，可以使逻辑单位大於图素。例如，假设您想要一种映射方式，使点 (0, 0) 显示在萤幕的左上角，y 的值向下增长（和 MM_TEXT 相似），但是逻辑座标单位为 1/16 英寸。以下是一种方法：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 16, 16, NULL) ;
SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
                  GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
```

SetWindowExtEx 函式的参数指出了每一英寸中逻辑单位数。

SetViewportExtEx 函式的参数指出了每一英寸中实际单位数 (图素)。

然而, 这种方法与 Windows NT 中的度量映射方式不一致。这些映射方式使用显示器的图素大小和公制大小。要与度量映射方式保持一致, 可以这样做:

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 160 * GetDeviceCaps (hdc, HORZSIZE) / 254,
    160 * GetDeviceCaps (hdc, VERTSIZE) / 254, NULL) ;
SetViewportExtEx (hdc, GetDeviceCaps (hdc, HORZRES),
    GetDeviceCaps (hdc, VERTRES), NULL) ;
```

在这个程式码中, 视埠范围设定为按图素计算的整个萤幕的大小, 视窗范围则必须设定为以 1/16 英寸为单位的整个萤幕的大小。GetDeviceCaps 以 HORZRES 和 VERTRES 为参数, 传回以毫米为单位的装置尺寸。如果我们使用浮点数, 将把毫米数除以 25.4, 转换为英寸, 然後, 再乘以 16 以转换为 1/16 英寸。但是, 由於我们使用的是整数, 所以先乘以 160, 再除以 254。

当然, 这种座标系会使逻辑单位大於实际单位。在设备上输出的所有东西都将映射为按 1/16 英寸增量的座标值。当然, 这样就不能画两条间隔 1/32 英寸的水平直线, 因为这样将需要小数逻辑座标。

MM_ANISOTROPIC: 根据需要放缩图像

在 MM_ISOTROPIC 映射方式下设定视窗和视埠范围时, Windows 会调整范围, 以便两条轴上的逻辑单位具有相同的实际尺度。在 MM_ANISOTROPIC 映射方式下, Windows 不对您所设定的值进行调整, 这就是说, MM_ANISOTROPIC 不需要维持正确的纵横比。

使用 MM_ANISOTROPIC 的一种方法是对显示区域使用任意座标, 就像我们对 MM_ISOTROPIC 所做的一样。下面的程式码将点 (0, 0) 设定为显示区域的左下角, x 轴和 y 轴都从 0 到 32,767:

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetViewportOrgEx (hdc, 0, cyClient, NULL) ;
```

在 MM_ISOTROPIC 方式下, 相似的程式码导致显示区域的一部分在轴的范围之外。但是对於 MM_ANISOTROPIC, 不论其尺度多大, 显示区域的右上角总是 (32767, 32767)。如果显示区域不是正方形的, 则逻辑 x 和 y 的单位具有不同的实际尺度。

前一节在 MM_ISOTROPIC 映射方式下, 我们讨论了在显示区域中画一个类似时钟的图像, x 和 y 轴的范围都是从 -1000 到 +1000。对於 MM_ANISOTROPIC, 也可以写出类似的程式:

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
```

```
SetWindowExtEx (hdc, 1000, 1000, NULL) ;
SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

与 MM_ANISOTROPIC 方式不同的是，这个时钟一般是椭圆形的，而不是圆形的。

另一种使用 MM_ANISOTROPIC 的方法是将 x 和 y 轴的单位固定，但其值不相等。例如，如果有一个只显示文字的程式，您可能想根据单个字元的高度和宽度设定一种粗刻度的座标：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 1, 1, NULL) ;
SetViewportExtEx (hdc, cxChar, cyChar, NULL) ;
```

当然，这里假设 cxChar 和 cyChar 分别是那种字体的字元宽度和高度。现在，您可以按字元行和列指定座标。下面的叙述在距离显示区域左边三个字元，上边二个字元处显示文字：

```
TextOut (hdc, 3, 2, TEXT ("Hello"), 5) ;
```

如果您使用固定大小的字体时会更加方便，就像下面的 WHATSIZE 程式所示的那样。

当您第一次设定 MM_ANISOTROPIC 映射方式时，它总是继承前面所设定的映射方式的范围，这会很方便。可以认为 MM_ANISOTROPIC 不「锁定」范围；也就是说，它允许您任意改变视窗范围。例如，假设您想用 MM_LOENGLISH 映射方式，因为希望逻辑单位为 0.01 英寸，但您不希望 y 轴的值向上增加，喜欢如 MM_TEXT 那样的方向，即 y 轴的值向下增加，可以使用如下的代码：

```
SIZE size ;
```

其他行程式

```
SetMapMode (hdc, MM_LOENGLISH) ;
SetMapMode (hdc, MM_ANISOTROPIC) ;
GetViewportExtEx (hdc, &size) ;
SetViewportExtEx (hdc, size.cx, -size.cy, NULL) ;
```

我们首先将映射方式设定为 MM_LOENGLISH，然後，通过将映射方式设定为 MM_ANISOTROPIC 让范围可以自由改变。GetViewportExtEx 取得视埠范围并放到一个 SIZE 结构中，然後，我们使用范围来呼叫 SetViewportExtEx，只是要将 y 范围取反。

WHATSIZE 程式

Windows 的小历史：第一篇如何写作 Windows 程式的介绍文章出现在《Microsoft Systems Journal》1986 年 12 月号上。在那篇文章中，范例程式叫做 WSZ（「what size：什么尺寸」），它以图素、英寸和毫米为单位显示了

显示区域的大小。那个程式的更简易版本是 WHATSIZE，如程式 5-6 所示。程式显示了以五种度量映射方式显示的视窗显示区域的大小。

程式 5-6 WHATSIZE

```

WHATSIZE.C
/*-----
   WHATSIZE.C -- What Size is the Window?
   (c) Charles Petzold, 1998
   -----*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("WhatSize") ;
    HWND        hwnd ;
    MSG         msg ;
    WNDCLASS    wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor= LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
            szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("What Size is the Window?"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
    }
}

```

```

        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void Show (HWND hwnd, HDC hdc, int xText, int yText, int iMapMode,
          TCHAR * szMapMode)
{
    TCHAR szBuffer [60] ;
    RECT rect ;

    SaveDC (hdc) ;
    SetMapMode (hdc, iMapMode) ;
    GetClientRect (hwnd, &rect) ;
    DPTOLP (hdc, (PPOINT) &rect, 2) ;

    RestoreDC (hdc, -1) ;
    TextOut (   hdc, xText, yText, szBuffer,
              wsprintf (szBuffer, TEXT ("%20s %7d %7d %7d %7d"), szMapMode,
              rect.left, rect.right, rect.top, rect.bottom)) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static TCHAR szHeading [] =
        TEXT ("Mapping Mode    Left  Right Top  Bottom") ;
    static TCHAR szUndLine [] =
        TEXT ("-----  ----  -----  ---  -----") ;
    static int   cxChar, cyChar ;
    HDC          hdc ;
    PAINTSTRUCT ps ;
    TEXTMETRIC  tm ;

    switch (message)
    {
    case WM_CREATE:
        hdc = GetDC (hwnd) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

```

```

SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 1, 1, NULL) ;
SetViewportExtEx (hdc, cxChar, cyChar, NULL) ;

TextOut (hdc, 1, 1, szHeading, lstrlen (szHeading)) ;
TextOut (hdc, 1, 2, szUndLine, lstrlen (szUndLine)) ;

Show (hwnd, hdc, 1, 3, MM_TEXT, TEXT ("TEXT (pixels)")) ;
Show (hwnd, hdc, 1, 4, MM_LOMETRIC, TEXT ("LOMETRIC (.1mm)")) ;
Show (hwnd, hdc, 1, 5, MM_HIMETRIC, TEXT ("HIMETRIC (.01
mm)")) ;

Show (hwnd, hdc, 1, 6, MM_LOENGLISH, TEXT ("LOENGLISH (.01 in)")) ;
Show (hwnd, hdc, 1, 7, MM_HIENGLISH, TEXT ("HIENGLISH (.001 in)")) ;
Show (hwnd, hdc, 1, 8, MM_TWIPS, TEXT ("TWIPS (1/1440 in)")) ;

EndPaint (hwnd, &ps) ;
return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

为了便於用 TextOut 函式显示资讯, WHA_SIZE 使用了一种固定间距的字体。下面一条简单的叙述就可以切换为固定间距的字体 (在 Windows 3.0 中它是优先使用的):

```
SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
```

有两个同样的函式用於选取画笔和画刷。像前面提到的, WHA_SIZE 也使用 MM_ANISTROPIC 映射方式将逻辑单位设定为字元大小。

当 WHA_SIZE 需要取得六种映射方式之一的显示区域的大小时, 它保存目前的装置内容, 设定一种新的映射方式, 取得显示区域座标, 将它们转换为逻辑座标, 然後在显示资讯之前, 恢复原映射方式。底下这些程式码在 WHA_SIZE 的 Show 函式里:

```

SaveDC (hdc) ;
SetMapMode (hdc, iMapMode) ;
GetClientRect (hwnd, &rect) ;
DptolP (hdc, (PPOINT) &rect, 2) ;
RestoreDC (hdc, -1) ;

```

图 5-19 显示了 WHA_SIZE 的典型输出。

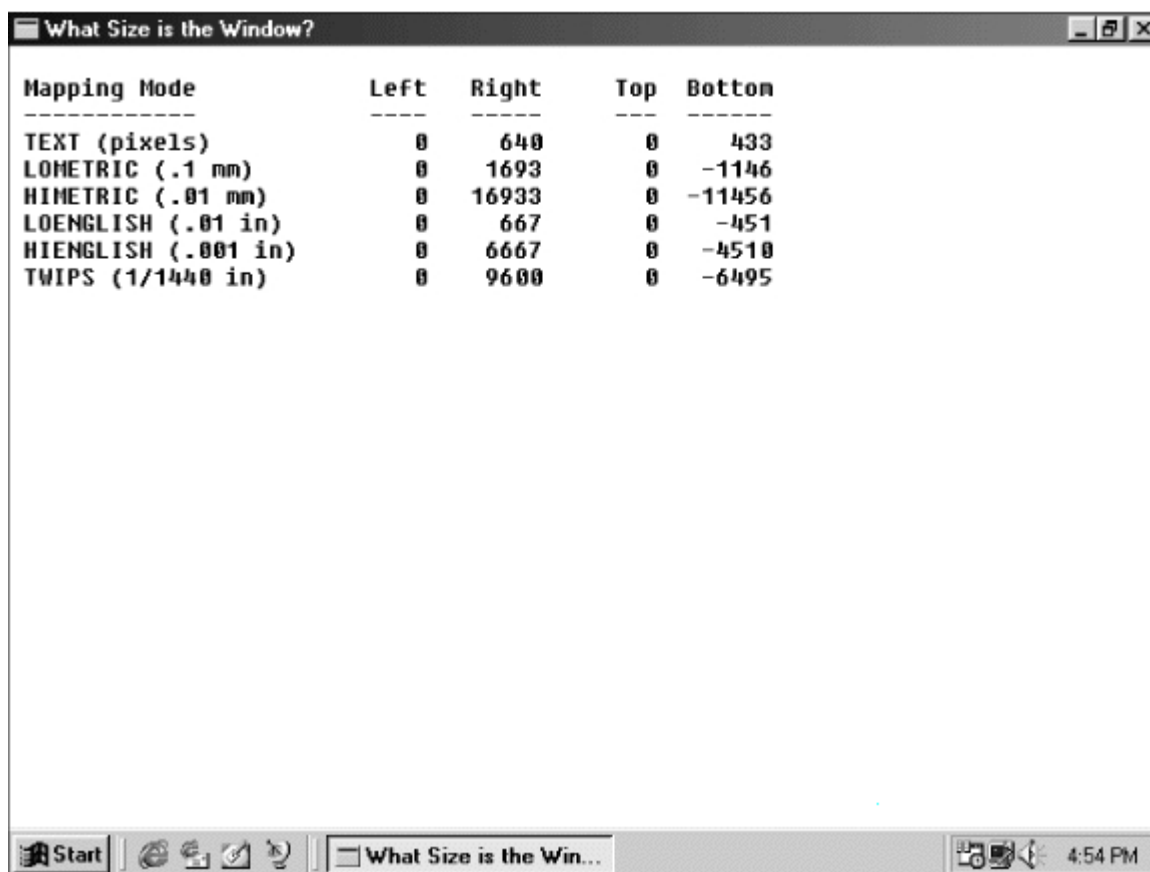


图 5-19 典型的 WHATSIZE 显示

矩形、区域和剪裁

Windows 包含了几种使用 RECT (矩形) 结构和「区域」的绘图函式。区域就是萤幕上的一块地方，它是矩形、多边形和椭圆的组合。

矩形函式

下面三个绘图函式需要一个指向矩形结构的指标：

```
FillRect (hdc, &rect, hBrush) ;
FrameRect (hdc, &rect, hBrush) ;
InvertRect (hdc, &rect) ;
```

在这些函式中，rect 参数是一个 RECT 型态的结构，它包含有 4 个栏位：left、top、right 和 bottom。这个结构中的座标被当作逻辑座标。

FillRect 用指定画刷来填入矩形（直到但不包含 right 和 bottom 座标），该函式不需要先将画刷选进装置内容。

FrameRect 使用画刷画矩形框，但是不填入矩形。使用画刷画矩形看起来有点奇怪，因为对于我们所介绍过的函式（如 Rectangle），其边线都是用目前画笔绘制的。FrameRect 允许使用者画一个不一定为纯色的矩形框。该边界框为一个逻辑单位元宽。如果逻辑单位大於装置单位，则边界框将会为 2 个图素宽或

者更宽。

InvertRect 将矩形中所有图素翻转，1 转换成 0，0 转换为 1，该函数将白色区域转变成黑色，黑色区域转变为白色，绿色区域转变成洋红色。

Windows 还提供了 9 个函数，使您可以更容易、更清楚地操作 RECT 结构。例如，要将 RECT 结构的四个栏位设定为特定值，通常使用如下的程式段：

```
rect.left      = xLeft ;
rect.top       = xTop  ;
rect.right     = xRight ;
rect.bottom    = xBottom ;
```

但是，通过呼叫 SetRect 函数，只需要一道叙述就可以得到同样的结果：

```
SetRect (&rect, xLeft, yTop, xRight, yBottom) ;
```

在您想要做以下事情之一时，可以很方便地选用其他 8 个函数：

将矩形沿 x 轴和 y 轴移动几个单元	OffsetRect (&rect, x, y) ;
增减矩形的尺寸	InflateRect (&rect, x, y) ;
矩形各栏位设定为 0	SetRectEmpty (&rect) ;
将矩形复制给另一个矩形	CopyRect (&DestRect, &SrcRect) ;
取得两个矩形的交集	IntersectRect (&DestRect, &SrcRect1, &SrcRect2) ;
取得两个矩形的联集	UnionRect (&DestRect, &SrcRect1, &SrcRect2) ;
确定矩形是否为空	bEmpty = IsRectEmpty (&rect) ;
确定点是否在矩形内	bInRect = PtInRect (&rect, point) ;

大多数情况下，与这些函数相同作用的程式码很简单。例如，您可以用下列叙述来替代 CopyRect 函数呼叫：

```
DestRect = SrcRect ;
```

随机矩形

在图形系统中，有这么一个「永远」有人执行的有趣程式，它简单地使用随机的大小和色彩绘制一系列矩形。您可以在 Windows 中建立一个这样的程式，但是它并不像乍看起来那样容易编写。我希望您能认识到，您不能简单地在 WM_PAINT 讯息中使用一个 while(TRUE) 回圈。当然，它能够执行，但是程式将停止对其他讯息的处理，同时，这个程式不能中止或者最小化。

一种可以接受的方法是设定一个 Windows 计时器，给视窗程序发送 WM_TIMER 讯息（我将在第八章中讨论计时器）。对于每条 WM_TIMER 讯息，您使用 GetDC 取得一个装置内容，画一个随机的矩形，然后用 ReleaseDC 释放装置内容。但是这样又降低了程式的趣味性，因为程式不能尽可能快地画随机矩形，它必须等待 WM_TIMER 讯息，而这又依赖于系统时钟的解析度。

在 Windows 中一定有很多「闲置时间」，在这个时间内，所有讯息伫列为

空, Windows 只停在一个小回圈中等待键盘或者滑鼠输入。我们能否在闲置时间内获得控制, 绘制矩形, 并且只在有讯息加入程式的讯息伫列之後才释放控制呢? 这就是 PeekMessage 函式的目的之一。下面是 PeekMessage 呼叫的一个例子:

```
PeekMessage (&msg, NULL, 0, 0, PM_REMOVE) ;
```

前面的四个参数 (一个指向 MSG 结构的指标、一个视窗代号、两个值指示讯息范围) 与 GetMessage 的参数相同。将第二、三、四个参数设定为 NULL 或 0 时, 表明我们想让 PeekMessage 传回程式中所有视窗的所有讯息。如果要将讯息从讯息伫列中删除, 则将 PeekMessage 的最後一个参数设定为 PM_REMOVE。如果您不希望删除讯息, 那么您可以将这个参数设定为 PM_NOREMOVE。这就是为什么 Peek_Message 是「偷看」而不是「取得」的原因, 它使得程式可以检查程式的伫列中的下一个讯息, 而不实际删除它。

GetMessage 不将控制传回给程式, 直到从程式的讯息伫列中取得讯息, 但是 PeekMessage 总是立刻传回, 而不论一个讯息是否出现。当讯息伫列中有一个讯息时, PeekMessage 的传回值为 TRUE (非 0), 并且将按通常方式处理讯息。当伫列中没有讯息时, PeekMessage 传回 FALSE (0)。

这使得我们可以改写普通的讯息回圈。我们可以将如下所示的回圈:

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
```

替换为下面的回圈:

```
while (TRUE)
{
    if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (msg.message == WM_QUIT)
            break ;
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    else
    {
        // 完成某些工作的其他行程式
    }
}
return msg.wParam ;
```

注意, WM_QUIT 讯息被另外挑出来检查。在普通的讯息回圈中您不必这么作,

因为如果 GetMessage 接收到一个 WM_QUIT 讯息，它将传回 0，但是 PeekMessage 用它的传回值来指示是否得到一个讯息，所以需要对 WM_QUIT 进行检查。

如果 PeekMessage 的传回值为 TRUE，则讯息按通常方式进行处理。如果传回值为 FALSE，则在将控制传回给 Windows 之前，还可以作一点工作（如显示另一个随机矩形）。

（尽管 Windows 文件上说，您不能用 PeekMessage 从讯息伫列中删除 WM_PAINT 讯息，但是这并不是什么大不了的问题。毕竟，GetMessage 并不从讯息伫列中删除 WM_PAINT 讯息。从伫列中删除 WM_PAINT 讯息的唯一方法是令视窗显示区域的失效区域变得有效，这可以用 ValidateRect 和 ValidateRgn 或者 BeginPaint 和 EndPaint 对来完成。如果您在使用 PeekMessage 从伫列中取出 WM_PAINT 讯息後，同平常一样处理它，那么就不会有问题了。所不能作的是使用如下所示的程式码来清除讯息伫列中的所有讯息：

```
while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE)) ;
```

这行叙述从讯息伫列中删除 WM_PAINT 之外的所有讯息。如果伫列中有一个 WM_PAINT 讯息，程式就会永远地陷在 while 回圈中。）

PeekMessage 在 Windows 的早期版本中比在 Windows 98 中要重要得多。这是因为 Windows 的 16 位元版本使用的是非优先权式的多工（我将在第二十章中讨论这一点）。Windows 的 Terminal 程式在从通讯埠接收输入後，使用一个 PeekMessage 回圈。列印管理器程式使用这个技术来进行列印，其他的 Windows 列印应用程式通常都会使用一个 PeekMessage 回圈。在 Windows 98 优先权式的多工环境下，程式可以建立多个执行绪，我们将第二十章看到这一点。

不管怎样，有了 PeekMessage 函式，我们就可以编写一个不停地显示随机矩形的程式。这个 RANDRECT 如程式 5-7 中所示。

程式 5-7 RANDRECT

```
RANDRECT.C
/*-----
   RANDRECT.C --  Displays Random Rectangles
                  (c) Charles Petzold, 1998
   -----*/

#include <windows.h>
#include <stdlib.h>    // for the rand function

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
void DrawRectangle (HWND) ;

int cxClient, cyClient ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
```

```

{
    static TCHAR szAppName[] = TEXT ("RandRect") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc= WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Random Rectangles"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (TRUE)
    {
        if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
        {
            if (msg.message == WM_QUIT)
                break ;
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
        else
            DrawRectangle (hwnd) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```



```

{
    switch (iMsg)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;

    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

void DrawRectangle (HWND hwnd)
{
    HBRUSH      hBrush ;
    HDC          hdc ;
    RECT         rect ;

    if (cxClient == 0 || cyClient == 0)
        return ;
    SetRect (&rect, rand () % cxClient, rand () % cyClient,
              rand () % cxClient, rand () % cyClient) ;
    hBrush = CreateSolidBrush (
        RGB (rand () % 256, rand () % 256, rand () % 256)) ;
    hdc = GetDC (hwnd) ;
    FillRect (hdc, &rect, hBrush) ;
    ReleaseDC (hwnd, hdc) ;
    DeleteObject (hBrush) ;
}

```

这个程式在现在的电脑上执行得非常快，看起来都不像是一系列随机矩形了。程式使用我在上面讨论过的 SetRect 和 FillRect 函式，根据由 C 的 rand 函式得到的乱数决定矩形座标和实心画刷的色彩。我将在第二十章中提供这个程式的多执行绪版本。

建立和绘制剪裁区域

剪裁区域是对显示器上一个范围的描述，这个范围是矩形、多边形和椭圆的组合。剪裁区域可以用於绘制和剪裁，通过将剪裁区域选进装置内容，就可以用剪裁区域来进行剪裁（就是说，将可以绘图的范围限制为显示区域的一部分）。与画笔、画刷和点阵图一样，剪裁区域是 GDI 物件，您应该呼叫 DeleteObject 来删除您所建立的剪裁区域。

当您建立一个剪裁区域时，Windows 传回一个该剪裁区域的代号，型态为

HRGN。最简单的剪裁区域是矩形，有两种建立矩形的方法：

```
hRgn = CreateRectRgn (xLeft, yTop, xRight, yBottom) ;
```

或者

```
hRgn = CreateRectRgnIndirect (&rect) ;
```

您也可以建立椭圆剪裁区域：

```
hRgn = CreateEllipticRgn (xLeft, yTop, xRight, yBottom) ;
```

或者

```
hRgn = CreateEllipticRgnIndirect (&rect) ;
```

CreateRoundRectRgn 建立圆角的矩形剪裁区域。

建立多边形剪裁区域的函式类似於 Polygon 函式：

```
hRgn = CreatePolygonRgn (&point, iCount, iPolyFillMode) ;
```

point 参数是一个 POINT 型态的结构阵列，iCount 是点的数目，iPolyFillMode 是 ALTERNATE 或者 WINDING。您还可以用 CreatePolyPolygonRgn 来建立多个多边形剪裁区域。

那么，您会问，剪裁区域究竟有什么特别之处？下面这个函式才真正显示出了剪裁区域的作用：

```
iRgnType = CombineRgn (hDestRgn, hSrcRgn1, hSrcRgn2, iCombine) ;
```

这一函式将两个剪裁区域（hSrcRgn1 和 hSrcRgn2）组合起来并用代号 hDestRgn 指向组合成的剪裁区域。这三个剪裁区域代号都必须都是有效的，但是 hDestRgn 原来所指向的剪裁区域被破坏掉了（当您使用这个函式时，您可能要让 hDestRgn 在初始时指向一个小的矩形剪裁区域）。

iCombine 参数说明 hSrcRgn1 和 hSrcRgn2 如何组合，见表 5-9。

表 5-9

iCombine 值	新剪裁区域
RGN_AND	两个剪裁区域的公共部分
RGN_OR	两个剪裁区域的全部
RGN_XOR	两个剪裁区域的全部除去公共部分
RGN_DIFF	hSrcRgn1 不在 hSrcRgn2 中的部分
RGN_COPY	hSrcRgn1 的全部（忽略 hSrcRgn2）

从 CombineRgn 传回的 iRgnType 值是下列之一：NULLREGION，表示得到一个空剪裁区域；SIMPLEREGION，表示得到一个简单的矩形、椭圆或者多边形；COMPLEXREGION，表示多个矩形、椭圆或多边形的组合；ERROR，表示出错了。

剪裁区域的代号可以用於四个绘图函式：

```
FillRgn (hdc, hRgn, hBrush) ;
FrameRgn (hdc, hRgn, hBrush, xFrame, yFrame) ;
InvertRgn (hdc, hRgn) ;
PaintRgn (hdc, hRgn) ;
```

FillRgn、FrameRgn 和 InvertRgn 类似於 FillRect、FrameRect 和 InvertRect。FrameRgn 的 xFrame 和 yFrame 参数是画在区域周围的边框的宽度和高度。PaintRgn 函式用装置内容中目前画刷填入所指定的区域。所有这些函式都假定区域是用逻辑坐标定义的。

在您用完一个区域後，可以像删除其他 GDI 物件那样删除它：

```
DeleteObject (hRgn) ;
```

矩形与区域的剪裁

区域也在剪裁中扮演了一个角色。InvalidateRect 函式使显示的一个矩形区域失效，并产生一个 WM_PAINT 讯息。例如，您可以使用 InvalidateRect 函式来清除显示区域并产生一个 WM_PAINT 讯息：

```
InvalidateRect (hwnd, NULL, TRUE) ;
```

您可以通过呼叫 GetUpdateRect 来取得失效矩形的座标，并且可以使用 ValidateRect 函式使显示区域的矩形有效。当您接收到一个 WM_PAINT 讯息时，无效矩形的座标可以从 PAINTSTRUCT 结构中得到，该结构是用 BeginPaint 函式填入的。这个无效矩形还定义了一个「剪裁区域」，您不能在剪裁区域外绘图。

Windows 有两个作用於剪裁区域而不是矩形的函式，它们类似於 InvalidateRect 和 ValidateRect：

```
InvalidateRgn (hwnd, hRgn, bErase) ;
```

和

```
ValidateRgn (hwnd, hRgn) ;
```

当您接收到一个由无效区域引起的 WM_PAINT 讯息时，剪裁区域不一定是矩形。

您可以使用以下两个函式之一：

```
SelectObject (hdc, hRgn) ;
```

或

```
SelectClipRgn (hdc, hRgn) ;
```

通过将一个剪裁区域选进装置内容来建立自己的剪裁区域，这个剪裁区域使用装置座标。

GDI 为剪裁区域建立一份副本，所以在将它选进装置内容之後，使用者可以删除它。Windows 还提供了几个对剪裁区域进行操作的函式，如 ExcludeClipRect 用於将一个矩形从剪裁区域里排除掉，IntersectClipRect 用於建立一个新的剪裁区域，它是前一个剪裁区域与一个矩形的交，OffsetClipRgn 用於将剪裁区域移动到显示区域的另一部分。

CLOVER 程式

CLOVER 程式用四个椭圆组成一个剪裁区域，将这个剪裁区域选进装置内容中，然後画出从视窗显示区域的中心出发的一系列直线，这些直线只出现在剪裁区域所限定的范围，结果显示如图 5-20 所示。

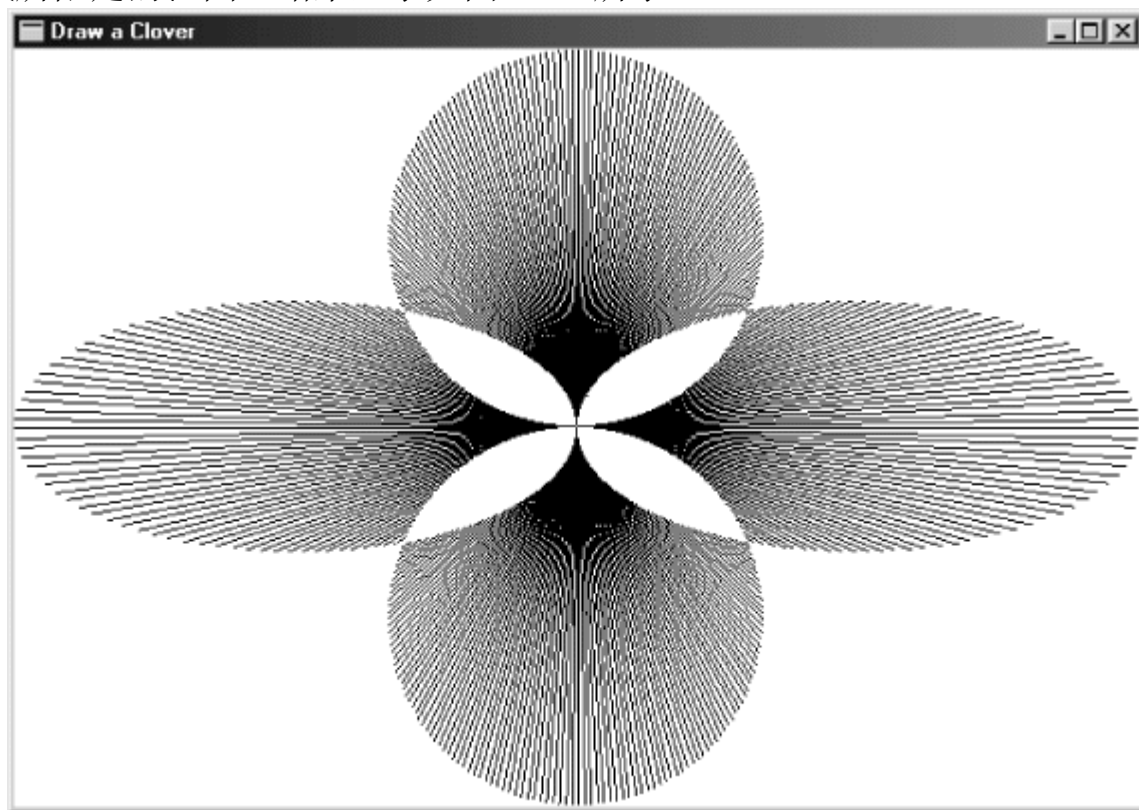


图 5-20 CLOVER 利用复杂的剪裁区域画出的图像

要用常规的方法画出这个图形，就必须根据椭圆的边线公式计算出每条直线的端点。利用复杂的剪裁区域，可以直接画出这些线条，而让 Windows 确定其端点。CLOVER 如程式 5-8 所示。

程式 5-8 CLOVER

```
CLOVER.C
/*-----
    CLOVER.C -- Clover Drawing Program Using Regions
    (c) Charles Petzold, 1998
    -----*/
#include <windows.h>
#include <math.h>

#define TWO_PI (2.0 * 3.14159)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Clover") ;
    HWND          hwnd ;
```

```
MSG          msg ;
WNDCLASS     wndclass ;

wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc= WndProc ;
wndclass.cbClsExtra  = 0 ;
wndclass.cbWndExtra  = 0 ;
wndclass.hInstance  = hInstance ;
wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName = NULL ;
wndclass.lpszClassName = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Draw a Clover"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static HRGN hRgnClip ;
    static int  cxClient, cyClient ;
    double      fAngle, fRadius ;
    HCURSOR     hCursor ;
    HDC          hdc ;
    HRGN        hRgnTemp[6] ;
    int          i ;
    PAINTSTRUCT ps ;
    switch (iMsg)
```

```

{
case WM_SIZE:
    cxClient    = LOWORD (lParam) ;
    cyClient    = HIWORD (lParam) ;
    hCursor     = SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    if (hRgnClip)
        DeleteObject (hRgnClip) ;

    hRgnTemp[0] = CreateEllipticRgn (0, cyClient / 3,
                                    cxClient / 2, 2 * cyClient / 3) ;
    hRgnTemp[1] = CreateEllipticRgn (cxClient / 2, cyClient / 3,
                                    cxClient, 2 * cyClient / 3) ;
    hRgnTemp[2] = CreateEllipticRgn (cxClient / 3, 0,
                                    2 * cxClient / 3, cyClient / 2) ;
    hRgnTemp[3] = CreateEllipticRgn (cxClient / 3, cyClient / 2,
2 * cxClient / 3, cyClient) ;
    hRgnTemp[4] = CreateRectRgn (0, 0, 1, 1) ;
    hRgnTemp[5] = CreateRectRgn (0, 0, 1, 1) ;
    hRgnClip     = CreateRectRgn (0, 0, 1, 1) ;

    CombineRgn (hRgnTemp[4], hRgnTemp[0], hRgnTemp[1], RGN_OR) ;
    CombineRgn (hRgnTemp[5], hRgnTemp[2], hRgnTemp[3], RGN_OR) ;
    CombineRgn (hRgnClip, hRgnTemp[4], hRgnTemp[5], RGN_XOR) ;

    for (i = 0 ; i < 6 ; i++)
        DeleteObject (hRgnTemp[i]) ;

    SetCursor (hCursor) ;
    ShowCursor (FALSE) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
    SelectClipRgn (hdc, hRgnClip) ;
    fRadius = _hypot (cxClient / 2.0, cyClient / 2.0) ;
    for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)
    {
        MoveToEx (hdc, 0, 0, NULL) ;
        LineTo (hdc, (int) (fRadius * cos (fAngle) + 0.5),
                (int) (-fRadius * sin (fAngle) + 0.5)) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

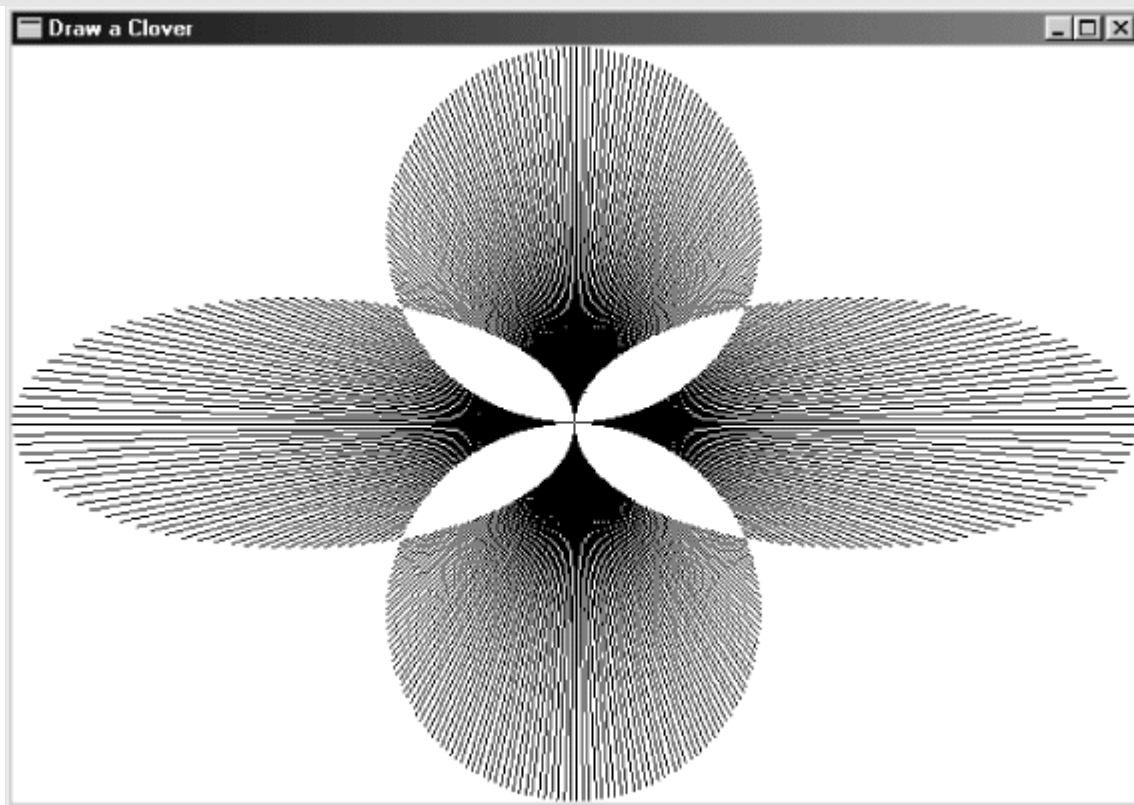
case WM_DESTROY:

```

```

DeleteObject (hRgnClip) ;
PostQuitMessage (0) ;
return 0 ;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

```



由於剪裁区域总是使用装置座标，CLOVER 程式必须在每次接收到 WM_SIZE 讯息时重新建立剪裁区域。几年前，这可能需要几秒钟。现在的快速机器在一瞬间就可以画出来。

CLOVER 从建立四个椭圆剪裁区域开始，这四个椭圆存放在 hRgnTemp 阵列的头四个元素中，然後建立三个「空」剪裁区域：

```

hRgnTemp [4]      = CreateRectRgn (0, 0, 1, 1) ;
hRgnTemp [5]      = CreateRectRgn (0, 0, 1, 1) ;
hRgnClip          = CreateRectRgn (0, 0, 1, 1) ;

```

显示区域左右的两个椭圆区域组合起来：

```
CombineRgn (hRgnTemp [4], hRgnTemp [0], hRgnTemp [1], RGN_OR) ;
```

同样，显示区域上下两个椭圆区域组合起来：

```
CombineRgn (hRgnTemp [5], hRgnTemp [2], hRgnTemp [3], RGN_OR) ;
```

最後，两个组合後的区域再组合到 hRgnClip 中：

```
CombineRgn (hRgnClip, hRgnTemp [4], hRgnTemp [5], RGN_XOR) ;
```

RGN_XOR 识别字用於从结果区域中排除重叠部分。最後，删除 6 个临时区域：

```

for (i = 0 ; i < 6 ; i++)
    DeleteObject (hRgnTemp [i]) ;

```

与画出的图形比起来，WM_PAINT 的处理很简单。视埠原点设定为显示区域

的中心（使画直线更容易一些），在 WM_SIZE 讯息处理期间建立的区域选择为装置内容的剪裁区域：

```
SetViewportOrg (hdc, xClient / 2, yClient / 2) ;  
SelectClipRgn (hdc, hRgnClip) ;
```

现在，剩下的就是画直线了，共 360 条，每隔一度画一条。每条线的长度为变数 fRadius，这是从中心到显示区域的角落的距离：

```
fRadius = hypot (xClient / 2.0, yClient / 2.0) ;  
for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)  
{  
    MoveToEx (hdc, 0, 0, NULL) ;  
    LineTo (hdc, (int) ( fRadius * cos (fAngle) + 0.5),  
           (int) (-fRadius * sin (fAngle) + 0.5)) ;  
}
```

在处理 WM_DESTROY 讯息时，删除该剪裁区域：

```
DeleteObject (hRgnClip) ;
```

这不是本书关于图形程式设计的最後内容第十三章讨论列印，第十四章和十五章讨论点阵图，第十七章讨论文字和字体，第十八章讨论 metafile。

第六章 键盘

在 Microsoft Windows 98 中，键盘和滑鼠是两个标准的使用者输入来源，在一些连贯操作中常产生互补作用。当然，滑鼠在今天的應用程式中比十年前使用得更为广泛。甚至在一些應用程式中，我们更习惯於使用滑鼠，例如在游戏、画图程式、音乐程式以及 Web 浏览器等程式中就是这样。然而，我们可以不使用滑鼠，但绝对不能从一般的 PC 中把键盘拆掉。

相对於个人电脑的其他元件，键盘有非常悠久的历史，它起源於 1874 年的第一台 Remington 打字机。早期的电脑程式员用键盘在 Hollerith 卡片上打孔，後来在终端机上用键盘直接与大型主机沟通。PC 上的键盘在某些方面进行了扩充，加上了功能键、游标移动键和单独的数字键盘，但它们的输入原理基本相同。

键盘基础

您大概已经猜到 Windows 程式是如何获得键盘输入的：键盘输入以讯息的形式传递给程式的视窗讯息处理程式。实际上，第一次学习讯息时，键盘事件就是一个讯息如何将不同型态资讯传递给應用程式的显例。

Windows 用八种不同的讯息来传递不同的键盘事件。这好像太多了，但是（就像我们所看到的一样）程式可以忽略其中至少一半的讯息而不会有任何问题。并且，在大多数情况下，这些讯息中包含的键盘资讯会多於程式所需要的。处理键盘的部分工作就是识别出哪些讯息是重要的，哪些是不重要的。

忽略键盘

虽然键盘是 Windows 程式中使用者的主要来源，但是程式不必对它接收的所有讯息都作出回应。Windows 本身也能处理许多键盘功能。

例如，您可以忽略那些属於系统功能的按键，它们通常用到 Alt 键。程式不必监视这些按键，因为 Windows 会将按键的作用通知程式（当然，如果程式想这么做，它也能监视这些按键）。虽然呼叫程式功能表的按键将通过视窗的视窗讯息处理程式，但通常内定的处理方式是将按键传递给 DefWindowProc。最终，视窗讯息处理程式将获得一个讯息，表示一个功能表项被选择了。通常，这是所有视窗讯息处理程式需要知道的（在第十章将介绍功能表）。

有些 Windows 程式使用「键盘加速键」来启动通用功能表项。加速键通常是功能键或字母同 Ctrl 键的组合（例如，Ctrl-S 用於保存档案）。这些键盘加

速键与程式功能表一起在程式的资源描述档案中定义（我们可以在第十章看到）。Windows 将这些键盘加速键转换为功能表命令讯息，您不必自己去进行转换。

对话方块也有键盘介面，但是当对话方块处于活动状态时，應用程式通常不必监视键盘。键盘介面由 Windows 处理，Windows 把关于按键作用的讯息发送给程式。对话方块可以包含用于输入文字的编辑控制项。它们一般是小方框，使用者可以在框中键入字符串。Windows 处理所有编辑控制项逻辑，并在输入完毕后，将编辑控制项的最终内容传送给程式。关于对话方块的详细资讯，请参见第十一章。

编辑控制项不必局限于单独一行，而且也不限于只在对话方块中。一个在程式主视窗内的多行编辑控制项就能够作为一个简单的文字编辑器了（参见第九、十、十一和十三章的 POPPAD 程式）。Windows 甚至有一个 Rich Text 文字编辑控制项，允许您编辑和显示格式化的文字（请参见/Platform SDK/User Interface Services/Controls/Rich Edit Controls）。

您将会发现，在开发 Windows 程式时，可以使用处理键盘和鼠标输入的子视窗控制项来将较高层的资讯传递回父视窗。只要这样的控制项用得够多，您就不会因处理键盘讯息而烦恼了。

谁获得了焦点

与所有的个人电脑硬体一样，键盘必须由在 Windows 下执行的所有應用程式共用。有些應用程式可能有多个视窗，键盘必须由该應用程式内的所有视窗共用。

回想一下，程式用来从讯息伫列中检索讯息的 MSG 结构包括 hwnd 栏位。此栏位指出接收讯息的视窗控制项码。讯息回圈中的 DispatchMessage 函式向视窗讯息处理程式发送该讯息，此视窗讯息处理程式与需要讯息的视窗相联系。在按下键盘上的键时，只有一个视窗讯息处理程式接收键盘讯息，并且此讯息包括接收讯息的视窗控制项码。

接收特定键盘事件的视窗具有输入焦点。输入焦点的概念与活动视窗的概念很相近。有输入焦点的视窗是活动视窗或活动视窗的衍生视窗（活动视窗的子视窗，或者活动视窗子视窗的子视窗等等）。

通常很容易辨别活动视窗。它通常是顶层视窗——也就是说，它的父视窗代号是 NULL。如果活动视窗有标题列，Windows 将突出显示标题列。如果活动视窗具有对话方块架（对话方块中很常见的格式）而不是标题列，Windows 将突出显示框架。如果活动视窗目前是最小化的，Windows 将在工作列中突出显示该

项，其显示就像一个按下的按钮。

如果活动视窗有子视窗，那么有输入焦点的视窗既可以是活动视窗也可以是其子视窗。最常见的子视窗有类似以下控制项：出现在对话方块中的下压按钮、单选钮、核取方块、卷动列、编辑方块和清单方块。子视窗不能自己成为活动视窗。只有当它是活动视窗的衍生视窗时，子视窗才能有输入焦点。子视窗控制项一般通过显示一个闪烁的插入符号或虚线来表示它具有输入焦点。

有时输入焦点不在任何视窗中。这种情况发生在所有程式都是最小化的时候。这时，Windows 将继续向活动视窗发送键盘讯息，但是这些讯息与发送给非最小化的活动视窗的键盘讯息有不同的形式。

视窗讯息处理程式通过拦截 WM_SETFOCUS 和 WM_KILLFOCUS 讯息来判定它的视窗何时拥有输入焦点。WM_SETFOCUS 指示视窗正在得到输入焦点，WM_KILLFOCUS 表示视窗正在失去输入焦点。我将在本章的后面详细说明这些讯息。

伫列和同步

当使用者按下并释放键盘上的键时，Windows 和键盘驱动程序将硬体扫描码转换为格式讯息。然而，这些讯息并不保存在讯息伫列中。实际上，Windows 在所谓的「系统讯息伫列」中保存这些讯息。系统讯息伫列是独立的讯息伫列，它由 Windows 维护，用於初步保存使用者从键盘和滑鼠输入的资讯。只有当 Windows 應用程式处理完前一个使用者输入讯息时，Windows 才会从系统讯息伫列中取出下一个讯息，并将其放入應用程式的讯息伫列中。

此过程分为两步：首先在系统讯息伫列中保存讯息，然後将它们放入應用程式的讯息伫列，其原因是需要同步。就像我们刚才所学的，假定接收键盘输入的视窗就是有输入焦点的视窗。使用者的输入速度可能比應用程式处理按键的速度快，并且特定的按键可能会使焦点从一个视窗切换到另一个视窗，後来的按键就输入到了另一个视窗。但如果後来的按键已经记下了目标视窗的位址，并放入了應用程式讯息伫列，那么後来的按键就不能输入到另一个视窗。

按键和字元

應用程式从 Windows 接收的关于键盘事件的讯息可以分为按键和字元两类，这与您看待键盘的两种方式一致。

首先，您可以将键盘看作是键的集合。键盘只有唯一的 A 键，按下该键是一次按键，释放该键也是一次按键。但是键盘也是能产生可显示字元或控制字元的输入设备。根据 Ctrl、Shift 和 Caps Lock 键的状态，A 键能产生几个字

元。通常情况下，此字元为小写 a。如果按下 Shift 键或者打开了 Caps Lock，则该字元就变成大写 A。如果按下了 Ctrl，则该字元为 Ctrl-A（它在 ASCII 中有意义，但在 Windows 中可能是某事件的键盘加速键）。在一些键盘上，A 按键之前可能有「死字元键(dead-character key)」或者 Shift、Ctrl 或者 Alt 的不同组合，这些组合可以产生带有音调标记的小写或者大写，例如，à、á、â、Ä、或 Å。

对产生可显示字元的按键组合，Windows 不仅给程式发送按键讯息，而且还发送字元讯息。有些键不产生字元，这些键包括 shift 键、功能键、光标移动键和特殊字元键如 Insert 和 Delete。对于这些键，Windows 只产生按键讯息。

按键讯息

当您按下一个键时，Windows 把 WM_KEYDOWN 或者 WM_SYSKEYDOWN 讯息放入有输入焦点的视窗的讯息伫列；当您释放一个键时，Windows 把 WM_KEYUP 或者 WM_SYSKEYUP 讯息放入讯息伫列中。

表 6-1

	键按下	键释放
非系统键	WM_KEYDOWN	WM_KEYUP
系统键	WM_SYSKEYDOWN	WM_SYSKEYUP

通常「down（按下）」和「up（放开）」讯息是成对出现的。不过，如果您按住一个键使得自动重复功能生效，那么当该键最后被释放时，Windows 会给视窗讯息处理程式发送一系列 WM_KEYDOWN（或者 WM_SYSKEYDOWN）讯息和一个 WM_KEYUP（或者 WM_SYSKEYUP）讯息。像所有放入伫列的讯息一样，按键讯息也有时间资讯。通过呼叫 GetMessageTime，您可以获得按下或者释放键的相对时间。

系统按键与非系统按键

WM_SYSKEYDOWN 和 WM_SYSKEYUP 中的「SYS」代表「系统」，它表示该按键对 Windows 比对 Windows 應用程式更加重要。WM_SYSKEYDOWN 和 WM_SYSKEYUP 讯息经常由与 Alt 相组合的按键产生，这些按键启动程式功能表或者系统功能表上的选项，或者用于切换活动视窗等系统功能（Alt-Tab 或者 Alt-Esc），也可以用作系统功能表加速键（Alt 键与一个功能键相结合，例如 Alt-F4 用于关闭應用程式）。程式通常忽略 WM_SYSKEYUP 和 WM_SYSKEYDOWN 讯息，并将它们传送到 DefWindowProc。由于 Windows 要处理所有 Alt 键的功能，所以您无需拦截

这些讯息。您的视窗讯息处理程式将最後收到关于这些按键结果（如功能表选择）的其他讯息。如果您想在自己的视窗讯息处理程式中加上拦截系统按键的程式码（如本章后面的 KEYVIEW1 和 KEYVIEW2 程式所作的那样），那么在处理这些讯息之后再传送到 DefWindowProc，Windows 就仍然可以将它们用于通常的目的。

但是，请再考虑一下，几乎所有会影响使用者程式视窗的讯息都会先通过使用者视窗讯息处理程式。只有使用者把讯息传送到 DefWindowProc，Windows 才会对讯息进行处理。例如，如果您将下面几行叙述：

```
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
    return 0 ;
```

加入到一个视窗讯息处理程式中，那么当您的程式主视窗拥有输入焦点时，就可以有效地阻止所有 Alt 键操作（我将在本章的后面讨论 WM_SYSCHAR），其中包括 Alt-Tab、Alt-Esc 以及功能表操作。虽然我怀疑您会这么做，但是，我相信您会感到视窗讯息处理程式的强大功能。

WM_KEYDOWN 和 WM_KEYUP 讯息通常是在按下或者释放不带 Alt 键的键时产生的，您的程式可以使用或者忽略这些讯息，Windows 本身并不处理这些讯息。

对所有四类按键讯息，wParam 是虚拟键代码，表示按下或释放的键，而 lParam 则包含属于按键的其他资料。

虚拟键码

虚拟键码保存在 WM_KEYDOWN、WM_KEYUP、WM_SYSKEYDOWN 和 WM_SYSKEYUP 讯息的 wParam 参数中。此代码标识按下或释放的键。

哈，又是「虚拟」，您喜欢这个词吗？虚拟指的是假定存在于思想中而不是现实世界中的一些事物，也只有熟练使用 DOS 组合语言编写应用程式的程式写作者才有可能指出，为什么对 Windows 键盘处理如此基本的键码是虚拟的而不是真实的。

对于早期的程式写作者来说，真实的键码由实际键盘硬体产生。在 Windows 文件中将这些键码称为「扫描码(scan codes)」。在 IBM 相容机种上，扫描码 16 是 Q 键，17 是 W 键，18 是 E、19 是 R，20 是 T，21 是 Y 等等。这时您会发现，扫描码是依据键盘的实际布局的。Windows 开发者认为这些代码过于与设备相关了，于是他们试图通过定义所谓的虚拟键码，以便经由与装置无关的方式处理键盘。其中一些虚拟键码不能在 IBM 相容机种上产生，但可能会在其他制造商生产的键盘中找到，或者在未来的键盘上找到。

您使用的大多数虚拟键码的名称在 WINUSER.H 表头档案中都定义为以 VK_ 开头。表 6-2 列出了这些名称和数值（十进位和十六进位），以及与虚拟键相对应的 IBM 相容机种键盘上的键。下表也标出了 Windows 执行时是否需要这些键。下表还按数位顺序列出了虚拟键码。

前四个虚拟键码中有三个指的是滑鼠键：

表 6-2

十进位	十六进位	WINUSER.H 识别字	必需?	IBM 相容键盘
1	01	VK_LBUTTON		滑鼠左键
2	02	VK_RBUTTON		滑鼠右键
3	03	VK_CANCEL	√	Ctrl-Break
4	04	VK_MBUTTON		滑鼠中键

您永远都不会从键盘讯息中获得这些滑鼠键代码。在下一章可以看到，我们能够从滑鼠讯息中获得它们。VK_CANCEL 代码是一个虚拟键码，它包括同时按下两个键 (Ctrl-Break)。Windows 應用程式通常不使用此键。

表 6-3 中的键——Backspace、Tab、Enter、Escape 和 Spacebar——通常用於 Windows 程式。不过，Windows 一般用字元讯息（而不是键盘讯息）来处理这些键。

表 6-3

十进位	十六进位	WINUSER.H 识别字	必需?	IBM 相容键盘
8	08	VK_BACK	√	Backspace
9	09	VK_TAB	√	Tab
12	0C	VK_CLEAR		Num Lock 关闭时的数字键盘 5
13	0D	VK_RETURN	√	Enter （或者另一个）
16	10	VK_SHIFT	√	Shift （或者另一个）
17	11	VK_CONTROL	√	Ctrl （或者另一个）
18	12	VK_MENU	√	Alt （或者另一个）
19	13	VK_PAUSE		Pause
20	14	VK_CAPITAL	√	Caps Lock
27	1B	VK_ESCAPE	√	Esc
32	20	VK_SPACE	√	Spacebar

另外，Windows 程式通常不需要监视 Shift、Ctrl 或 Alt 键的状态。

表 6-4 列出的前八个码可能是与 VK_INSERT 和 VK_DELETE 一起最常用的虚拟键码：

表 6-4

十进位	十六进位	WINUSER.H 识别字	必需?	IBM 相容键盘
33	21	VK_PRIOR	√	Page Up
34	22	VK_NEXT	√	Page Down
35	23	VK_END	√	End
36	24	VK_HOME	√	Home
37	25	VK_LEFT	√	左箭头
38	26	VK_UP	√	上箭头
39	27	VK_RIGHT	√	右箭头
40	28	VK_DOWN	√	下箭头
41	29	VK_SELECT		
42	2A	VK_PRINT		
43	2B	VK_EXECUTE		
44	2C	VK_SNAPSHOT		Print Screen
45	2D	VK_INSERT	√	Insert
46	2E	VK_DELETE	√	Delete
47	2F	VK_HELP		

注意，许多名称（例如 VK_PRIOR 和 VK_NEXT）都与键上的标志不同，而且也与卷动列中的识别字不统一。Print Screen 键在平时都被 Windows 應用程式所忽略。Windows 本身回应此键时会将视讯显示的点阵图影本存放到剪贴板中。假使有键盘提供了 VK_SELECT、VK_PRINT、VK_EXECUTE 和 VK_HELP，大概也没几个人看过那样的键盘。

Windows 也包括在主键盘上的字母和数位键的虚拟键码（数字键盘将单独处理）。

表 6-5

十进位	十六进位	WINUSER.H 识别字	必需?	IBM 相容键盘
48-57	30-39	无	√	主键盘上的 0 到 9
65-90	41-5A	无	√	A 到 Z

注意，数字和字母的虚拟键码是 ASCII 码。Windows 程式几乎从不使用这些虚拟键码；实际上，程式使用的是 ASCII 码字元的字元讯息。

表 6-6 所示的代码是由 Microsoft Natural Keyboard 及其相容键盘产生的：

表 6-6

十进位	十六进位	WINUSER.H 识别字	必需?	IBM 相容键盘
91	5B	VK_LWIN		左 Windows 键

92	5C	VK_RWIN		右 Windows 键
93	5D	VK_APPS		Applications 键

Windows 用 VK_LWIN 和 VK_RWIN 键打开「开始」功能表或者（在以前的版本中）启动「工作管理员程式」。这两个都可以用於登录或登出 Windows（只在 Microsoft Windows NT 中有效），或者登录或登出网路（在 Windows for Applications 中）。应用程式能够通过显示辅助资讯或者当成捷径键看待来处理 application 键。

表 6-7 所示的代码用於数字键盘上的键（如果有的话）：

表 6-7

十进位	十六进位	WINUSER. H 识别字	必需?	IBM 相容键盘
96-105	60-69	VK_NUMPAD0 到 VK_NUMPAD9		NumLock 打开时 数字键盘上的 0 到 9
106	6A	VK_MULTIPLY		数字键盘上的*
107	6B	VK_ADD		数字键盘上的+
108	6C	VK_SEPARATOR		
109	6D	VK_SUBTRACT		数字键盘上的-
110	6E	VK_DECIMAL		数字键盘上的.
111	6F	VK_DIVIDE		数字键盘上的/

最後，虽然多数的键盘都有 12 个功能键，但 Windows 只需要 10 个，而位元旗标却有 24 个。另外，程式通常用功能键作为键盘加速键，这样，它们通常不处理表 6-8 所示的按键：

表 6-8

十进位	十六进位	WINUSER. H 识别字	必需?	IBM 相容键盘
112-121	70-79	VK_F1 到 VK_F10	√	功能键 F1 到 F10
122-135	7A-87	VK_F11 到 VK_F24		功能键 F11 到 F24
144	90	VK_NUMLOCK		Num Lock
145	91	VK_SCROLL		Scroll Lock

另外，还定义了一些其他虚拟键码，但它们只用於非标准键盘上的键，或者通常在大型主机终端机上使用的键。查看/ Platform SDK / User Interface Services / User Input / Virtual-Key Codes，可得到完整的列表。

lParam 资讯

在四个按键讯息 (WM_KEYDOWN、WM_KEYUP、WM_SYSKEYDOWN 和 WM_SYSKEYUP) 中, wParam 讯息参数含有上面所讨论的虚拟键码, 而 lParam 讯息参数则含有对了解按键非常有用的其他资讯。lParam 的 32 位分为 6 个栏位, 如图 6-1 所示。

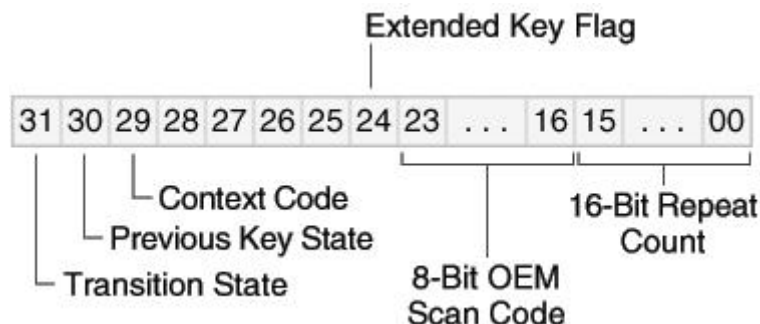


图 6-1 lParam 变数的 6 个按键讯息栏位

重复计数 (Repeat Count)

重复计数是该讯息所表示的按键次数, 大多数情况下, 重复计数设定为 1。不过, 如果按下一个键之後, 您的视窗讯息处理程式不够快, 以致不能处理自动重复速率 (您可以在「控制台」的「键盘」中进行设定) 下的按键讯息, Windows 就把几个 WM_KEYDOWN 或者 WM_SYSKEYDOWN 讯息组合到单个讯息中, 并相应地增加重复计数。WM_KEYUP 或 WM_SYSKEYUP 讯息的重复计数总是为 1。

因为重复计数大於 1 指示按键速率大於您程式的处理能力, 所以您也可能想在处理键盘讯息时忽略重复计数。几乎每个人都有文书处理或执行試算表时画面卷过头的经验, 因为多余的按键堆满了键盘缓冲区, 所以当程式用一些时间来处理每一次按键时, 如果忽略您程式中的重复计数, 就能够解决此问题。不过, 有时可能也会用到重复计数, 您应该尝试使用两种方法执行程式, 并从中找出一种较好的方法。

OEM 扫描码 (OEM Scan Code)

OEM 扫描码是由硬体 (键盘) 产生的代码。这对中古时代的组合语言程式写作者来说应该很熟悉, 它是从 PC 相容机种的 ROM BIOS 服务中所获得的值 (OEM 指的是 PC 的原始设备制造商 (Original Equipment Manufacturer) 及其与「IBM 标准」同步的内容)。在此我们不需要更多的资讯。除非需要依赖实际键盘布局的样貌, 不然 Windows 程式可以忽略掉几乎所有的 OEM 扫描码资讯, 参见第二十二章的程式 KBMIDI。

扩充键旗标 (Extended Key Flag)

如果按键结果来自 IBM 增强键盘的附加键之一，那么扩充键旗标为 1（IBM 增强型键盘有 101 或 102 个键。功能键在键盘顶端，游标移动键从数字键盘中分离出来，但在数字键盘上还保留有游标移动键的功能）。对键盘右端的 Alt 和 Ctrl 键，以及不是数字键盘那部分的游标移动键（包括 Insert 和 Delete 键）、数字键盘上的斜线（/）和 Enter 键以及 Num Lock 键等，此旗标均被设定为 1。Windows 程式通常忽略扩充键旗标。

内容代码 (Context Code)

右按键时，假如同时压下 ALT 键，那么内容代码为 1。对 WM_SYSKEYUP 与 WM_SYSKEYDOWN 而言，此位元总视为 1；而对 WM_KEYUP 与 WM_KEYDOWN 讯息而言，此位元为 0。除了两个之外：

如果活动视窗最小化了，则它没有输入焦点。这时候所有的按键都会产生 WM_SYSKEYUP 和 WM_SYSKEYDOWN 讯息。如果 Alt 键未被按下，则内容代码栏位被设定为 0。Windows 使用 WM_SYSKEYUP 和 WM_SYSKEYDOWN 讯息，从而使最小化了的
活动视窗不处理这些按键。

对于一些外国语文（非英文）键盘，有些字元是通过 Shift、Ctrl 或者 Alt 键与其他键相组合而产生的。这时内容代码为 1，但是此讯息并非系统按键讯息。

键的先前状态 (Previous Key State)

如果在此之前键是释放的，则键的先前状态为 0，否则为 1。对 WM_KEYUP 或者 WM_SYSKEYUP 讯息，它总是设定为 1；但是对 WM_KEYDOWN 或者 WM_SYSKEYDOWN 讯息，此位元可以为 0，也可以为 1。如果为 1，则表示该键是自动重复功能所产生的第二个或者后续讯息。

转换状态 (Transition State)

如果键正被按下，则转换状态为 0；如果键正被释放，则转换状态为 1。对 WM_KEYDOWN 或者 WM_SYSKEYDOWN 讯息，此栏位为 0；对 WM_KEYUP 或者 WM_SYSKEYUP 讯息，此栏位为 1。

位移状态

在处理按键讯息时，您可能需要知道是否按下了位移键（Shift、Ctrl 和 Alt）或开关键（Caps Lock、Num Lock 和 Scroll Lock）。通过呼叫 GetKeyState

函式，您就能获得此资讯。例如：

```
iState = GetKeyState (VK_SHIFT) ;
```

如果按下了 Shift，则 iState 值为负（即设定了最高位置位元）。如果 Caps Lock 键打开，则从

```
iState = GetKeyState (VK_CAPITAL) ;
```

传回的值低位元被设为 1。此位元与键盘上的小灯保持一致。

通常，您在使用 GetKeyState 时，会带有虚拟键码 VK_SHIFT、VK_CONTROL 和 VK_MENU（在说明 Alt 键时呼叫）。使用 GetKeyState 时，您也可以使用下面的识别字来确定按下的 Shift、Ctrl 或 Alt 键是左边的还是右边的：VK_LSHIFT、VK_RSHIFT、VK_LCONTROL、VK_RCONTROL、VK_LMENU、VK_RMENU。这些识别字只用于 GetKeyState 和 GetAsyncKeyState（下面将详细说明）。

使用虚拟键码 VK_LBUTTON、VK_RBUTTON 和 VK_MBUTTON，您也可以获得鼠标键的状态。不过，大多数需要监视鼠标键与按键相组合的 Windows 应用程序都使用其他方法来做到这一点——即在接收到鼠标讯息时检查按键。实际上，位移状态资讯包含在鼠标资讯中，正如您在下一章中将看到的一样。

请注意 GetKeyState 的使用，它并非即时检查键盘状态，而只是检查直到目前为止正在处理的讯息的键盘状态。多数情况下，这正符合您的要求。如果您需要确定使用者是否按下了 Shift-Tab，请在处理 Tab 键的 WM_KEYDOWN 讯息时呼叫 GetKeyState，带有参数 VK_SHIFT。如果 GetKeyState 传回的值负，那么您就知道在按下 Tab 键之前按下了 Shift 键。并且，如果在您开始处理 Tab 键之前，已经释放了 Shift 键也没有关系。您知道，在按下 Tab 键的时候 Shift 键是按下的。

GetKeyState 不会让您获得独立于普通键盘讯息的键盘资讯。例如，您或许想暂停视窗讯息处理程式的处理，直到您按下 F1 功能键为止：

```
while (GetKeyState (VK_F1) >= 0) ; // WRONG !!!
```

不要这么做！这将让程式当死（除非在执行此叙述之前早就从讯息伫列中接收到了 F1 的 WM_KEYDOWN）。如果您确实需要知道目前某键的状态，那么您可以使用 GetAsyncKeyState。

使用按键讯息

如果程式能够获得每个按键的资讯，这当然很理想，但是大多数 Windows 程式忽略了几乎所有的按键，而只处理部分的按键讯息。WM_SYSKEYDOWN 和 WM_SYSKEYUP 讯息是由 Windows 系统函式使用的，您不必为此费心，就算您要处理 WM_KEYDOWN 讯息，通常也可以忽略 WM_KEYUP 讯息。

Windows 程式通常为不产生字元的按键使用 WM_KEYDOWN 讯息。虽然您可能

认为借助按键讯息和位移键状态资讯能将按键讯息转换为字元讯息，但是不要这么做，因为您将遇到国际键盘间的差异所带来的问题。例如，如果您得到 wParam 等於 0x33 的 WM_KEYDOWN 讯息，您就可以知道使用者按下了键 3，到此为止一切正常。这时，如果用 GetKeyState 发现 Shift 键被按下，您就可能会认为使用者输入了#号，这可不一定。比如英国使用者就是在输入 £。

对于光标移动键、功能键、Insert 和 Delete 键，WM_KEYDOWN 讯息是最有用的。不过，Insert、Delete 和功能键经常作为功能表加速键。因为 Windows 能把功能表加速键翻译为功能表命令讯息，所以您就不必自己来处理按键。

在 Windows 之前的 MS-DOS 應用程式中大量使用功能键与 Shift、Ctrl 和 Alt 键的组合，同样地，您也可以 Windows 程式中使用（实际上，Microsoft Word 将大量的功能键用作命令快捷方式），但并不推荐这样做。如果您确实希望使用功能键，那么这些键应该是重复功能表命令。Windows 的目标之一就是提供不需要记忆或者使用复杂命令流程的使用者介面。

因此，可以归纳如下：多数情况下，您将只为光标移动键（有时也为 Insert 和 Delete 键）处理 WM_KEYDOWN 讯息。在使用这些键的时候，您可以通过 GetKeyState 来检查 Shift 键和 Ctrl 键的状态。例如，Windows 程式经常使用 Shift 与光标键的组合键来扩大文书处理里选中的范围。Ctrl 键常用于修改光标键的意义。例如，Ctrl 与右箭头键相组合可以表示光标右移一个字。

决定您的程式中使用键盘方式的最好方法之一是了解现有的 Windows 程式使用键盘的方式。如果您不喜欢那些定义，当然可以对其加以修改，但是这样做不利于其他人很快地学会使用您的程式。

为 SYSMETS 加上键盘处理功能

在编写第四章中三个版本的 SYSMETS 程式时，我们还不了解键盘，只能使用滚动列和滑鼠来滚动文字。现在我们知道了处理键盘讯息的方法，那么不妨在程式中加入键盘介面。显然，这是处理光标移动键的工作。我们将大多数光标键（Home、End、Page Up、Page Down、Up Arrow 和 Down Arrow）用于垂直滚动，左箭头键和右箭头键用于不太重要的水平滚动。

建立键盘介面的一种简单方法是在视窗讯息处理程式中加入与 WM_VSCROLL 和 WM_HSCROLL 处理方式相仿，而且本质上相同的 WM_KEYDOWN 处理方法。不过这样子做是不聪明的，因为如果要修改滚动列的做法，就必须相对地修改 WM_KEYDOWN。

为什么不简单地将每一种 WM_KEYDOWN 讯息都翻译成同等效用的 WM_VSCROLL 或者 WM_HSCROLL 讯息呢？通过向视窗讯息处理程式发送假冒讯息，我们可能会

让 WndProc 认为它获得了滚动资讯。

在 Windows 中，这种方法是可行的。发送讯息的函式叫做 SendMessage，它所用的参数与传递到视窗讯息处理程式的参数是相同的：

```
SendMessage (hwnd, message, wParam, lParam) ;
```

在呼叫 SendMessage 时，Windows 呼叫视窗代号为 hwnd 的视窗讯息处理程式，并把这四个参数传给它。当视窗讯息处理程式完成讯息处理之後，Windows 把控制传回到 SendMessage 呼叫之後的下一道叙述。您发送讯息过去的视窗讯息处理程式，可以是同一个视窗讯息处理程式、同一程式中的其他视窗讯息处理程式或者其他应用程式，中的视窗讯息处理程式。

下面说明在 SYSMETS 程式中使用 SendMessage 处理 WM_KEYDOWN 代码的方法：

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_HOME:
            SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
            break ;

        case VK_END:
            SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
            break ;

        case VK_PRIOR:
            SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
            break ;
```

至此，您已经有了大概观念了吧。我们的目标是为滚动列添加键盘介面，并且也正在这么做。通过把滚动讯息发送到视窗讯息处理程式，我们实作了用游标移动键进行滚动列的功能。现在您知道在 SYSMETS3 中为 WM_VSCROLL 讯息加上 SB_TOP 和 SB_BOTTOM 处理码的原因了吧。在那里并没有用到它，但是现在处理 Home 和 End 键时就有用了。如程式 6-1 所示的 SYSENTS4 就加上了这些变化。编译这个程式时还需要用到第四章的 SYSMETS.H 档案。

程式 6-1 SYSMETS4

```
SYSMETS4.C
/*-----
   SYSMETS4.C -- System Metrics Display Program No. 4
               (c) Charles Petzold, 1998
   -----*/

#include <windows.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```

        PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[]      = TEXT ("SysMets4") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
            szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 4"),
        WS_OVERLAPPEDWINDOW | WS_VSCROLL
| WS_HSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }

    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int  cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
    HDC         hdc ;
    int         i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
    PAINTSTRUCT ps ;

```



```
SCROLLINFO si ;
TCHAR          szBuffer[10] ;
TEXTMETRIC tm ;

switch (message)
{
case WM_CREATE:
    hdc = GetDC (hwnd) ;

    GetTextMetrics (hdc, &tm) ;
    cxChar      = tm.tmAveCharWidth ;
    cxCaps      = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar      = tm.tmHeight + tm.tmExternalLeading ;

    ReleaseDC (hwnd, hdc) ;

    // Save the width of the three columns

    iMaxWidth = 40 * cxChar + 22 * cxCaps ;
    return 0 ;

case WM_SIZE:
    cxClient      = LOWORD (lParam) ;
    cyClient      = HIWORD (lParam) ;

    // Set vertical scroll bar range and page size

    si.cbSize     = sizeof (si) ;
    si.fMask      = SIF_RANGE | SIF_PAGE ;
    si.nMin       = 0 ;
    si.nMax       = NUMLINES - 1 ;
    si.nPage      = cyClient / cyChar ;
    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;

    // Set horizontal scroll bar range and page size

    si.cbSize     = sizeof (si) ;
    si.fMask      = SIF_RANGE | SIF_PAGE ;
    si.nMin       = 0 ;
    si.nMax       = 2 + iMaxWidth / cxChar ;
    si.nPage      = cxClient / cxChar ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    return 0 ;

case WM_VSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize     = sizeof (si) ;
    si.fMask      = SIF_ALL ;
```

```
GetScrollInfo (hwnd, SB_VERT, &si) ;

        // Save the position for comparison later on

        iVertPos = si.nPos ;

        switch (LOWORD (wParam))
        {
case SB_TOP:
        si.nPos = si.nMin ;
        break ;

case SB_BOTTOM:
        si.nPos = si.nMax ;
        break ;

case SB_LINEUP:
        si.nPos -= 1 ;
        break ;

case SB_LINEDOWN:
        si.nPos += 1 ;
        break ;

case SB_PAGEUP:
        si.nPos -= si.nPage ;
        break ;

case SB_PAGEDOWN:
        si.nPos += si.nPage ;
        break ;

case SB_THUMBTRACK:
        si.nPos = si.nTrackPos ;
        break ;

        default:
        break ;
        }

        // Set the position and then retrieve it. Due to adjustments
        // by Windows it might not be the same as the value set.

        si.fMask = SIF_POS ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;

        // If the position has changed, scroll the window and update
it
```



```
    if (si.nPos != iVertPos)
    {
        ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
                                                                NULL, NULL) ;

        UpdateWindow (hwnd) ;
    }
    return 0 ;

case WM_HSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize      = sizeof (si) ;
    si.fMask       = SIF_ALL ;

    // Save the position for comparison later on

    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos      = si.nPos ;

    switch (LOWORD (wParam))
    {
case SB_LINELEFT:
    si.nPos -= 1 ;
    break ;

case SB_LINERIGHT:
    si.nPos += 1 ;
    break ;

case SB_PAGELEFT:
    si.nPos -= si.nPage ;
    break ;

case SB_PAGERIGHT:
    si.nPos += si.nPage ;
    break ;

case SB_THUMBPOSITION:
    si.nPos = si.nTrackPos ;
    break ;

    default:
    break ;
    }

    // Set the position and then retrieve it.  Due to
adjustments
    // by Windows it might not be the same as the value set.
```

```
    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

    // If the position has changed, scroll the window

    if (si.nPos != iHorzPos)
    {
        ScrollWindow (hwnd, cxChar * (iHorzPos - si.nPos), 0,
            NULL, NULL) ;
    }
    return 0 ;

case WM_KEYDOWN:
    switch (wParam)
    {
case VK_HOME:
        SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
        break ;

case VK_END:
        SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
        break ;

case VK_PRIOR:
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
        break ;

case VK_NEXT:
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ;
        break ;

case VK_UP:
        SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
        break ;

case VK_DOWN:
        SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
        break ;

case VK_LEFT:
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEUP, 0) ;
        break ;

case VK_RIGHT:
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEDOWN, 0) ;
        break ;
    }
}
```

```
        return 0 ;
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
        // Get vertical scroll bar position
    si.cbSize      = sizeof (si) ;
    si.fMask       = SIF_POS ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    iVertPos      = si.nPos ;

        // Get horizontal scroll bar position

    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos      = si.nPos ;

        // Find painting limits

    iPaintBeg      = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
    iPaintEnd      = min (NUMLINES - 1,
        iVertPos + ps.rcPaint.bottom / cyChar) ;

    for (i = iPaintBeg ; i <= iPaintEnd ; i++)
    {
        x = cxChar * (1 - iHorzPos) ;
        y = cyChar * (i - iVertPos) ;

        TextOut (hdc, x, y,
            sysmetrics[i].szLabel,
            lstrlen (sysmetrics[i].szLabel)) ;

        TextOut (hdc, x + 22 * cxCaps, y,
            sysmetrics[i].szDesc,
            lstrlen (sysmetrics[i].szDesc)) ;

        SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;

        TextOut (hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
            wsprintf (szBuffer, TEXT ("%5d"),
                GetSystemMetrics (sysmetrics[i].iIndex))) ;

        SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

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

```
    }  
    return DefWindowProc (hwnd, message, wParam, lParam) ;  
}
```

字元讯息

前面讨论了利用位移状态资讯把按键讯息翻译为字元讯息的方法，并且提到，仅利用转换状态资讯还不够，因为还需要知道与国家/地区有关的键盘配置。由於这个原因，您不应该试图把按键讯息翻译为字元代码。Windows 会为您完成这一工作，在前面我们曾看到过以下的程式码：

```
while (GetMessage (&msg, NULL, 0, 0))  
{  
    TranslateMessage (&msg) ;  
    DispatchMessage (&msg) ;  
}
```

这是 WinMain 中典型的讯息回圈。GetMessage 函式用伫列中的下一个讯息填入 msg 结构的栏位。DispatchMessage 以此讯息为参数呼叫适当的视窗讯息处理程式。

在这两个函式之间是 TranslateMessage 函式，它将按键讯息转换为字元讯息。如果讯息为 WM_KEYDOWN 或者 WM_SYSKEYDOWN，并且按键与位移状态相组合产生一个字元，则 TranslateMessage 把字元讯息放入讯息伫列中。此字元讯息将是 GetMessage 从讯息伫列中得到的按键讯息之後的下一个讯息。

四类字元讯息

字元讯息可以分为四类，如表 6-9 所示。

表 6-9

	字元	死字元
非系统字元	WM_CHAR	WM_DEADCHAR
系统字元	WM_SYSCHAR	WM_SYSDEADCHAR

WM_CHAR 和 WM_DEADCHAR 讯息是从 WM_KEYDOWN 得到的；而 WM_SYSCHAR 和 WM_SYSDEADCHAR 讯息是从 WM_SYSKEYDOWN 讯息得到的（我将简要地讨论一下什么是死字元）。

有一个好消息：在大多数情况下，Windows 程式会忽略除 WM_CHAR 之外的任何讯息。伴随四个字元讯息的 lParam 参数与产生字元代码讯息的按键讯息之 lParam 参数相同。不过，参数 wParam 不是虚拟键码。实际上，它是 ANSI 或 Unicode 字元代码。

这些字元讯息是我们将文字传递给视窗讯息处理程式时遇到的第一个讯

息。它们不是唯一的讯息，其他讯息伴随以 0 结尾的整个字串。视窗讯息处理程式是如何知道该字元是 8 位元的 ANSI 字元还是 16 位元的 Unicode 宽字元呢？很简单：任何与您用 RegisterClassA (RegisterClass 的 ANSI 版) 注册的视窗类别相联系的视窗讯息处理程式，都会获得含有 ANSI 字元代码的讯息。如果视窗讯息处理程式用 RegisterClassW (RegisterClass 的宽字元版) 注册，那么传递给视窗讯息处理程式的讯息就带有 Unicode 字元代码。如果程式用 RegisterClass 注册视窗类别，那么在 UNICODE 识别字被定义时就呼叫 RegisterClassW，否则呼叫 RegisterClassA。

除非在程式写作的时候混合了 ANSI 和 Unicode 的函式与视窗讯息处理程式，用 WM_CHAR 讯息（及其他三种字元讯息）说明的字元代码将是：

```
(TCHAR) wParam
```

同一个视窗讯息处理程式可能会用到两个视窗类别，一个用 RegisterClassA 注册，而另一个用 RegisterClassW 注册。也就是说，视窗讯息处理程式可能会获得一些 ANSI 字元代码讯息和一些 Unicode 字元代码讯息。如果您的视窗讯息处理程式需要晓得目前视窗是否处理 Unicode 讯息，则它可以呼叫：

```
fUnicode = IsWindowUnicode (hwnd) ;
```

如果 hwnd 的视窗讯息处理程式获得 Unicode 讯息，那么变数 fUnicode 将为 TRUE，这表示视窗是用 RegisterClassW 注册的视窗类别。

讯息顺序

因为 TranslateMessage 函式从 WM_KEYDOWN 和 WM_SYSKEYDOWN 讯息产生了字元讯息，所以字元讯息是夹在按键讯息之间传递给视窗讯息处理程式的。例如，如果 Caps Lock 未打开，而使用者按下再释放 A 键，则视窗讯息处理程式将接收到如表 6-10 所示的三个讯息：

表 6-10

讯息	按键或者代码
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「a」的字元代码 (0x61)
WM_KEYUP	「A」的虚拟键码 (0x41)

如果您按下 Shift 键，再按下 A 键，然後释放 A 键，再释放 Shift 键，就会输入大写的 A，而视窗讯息处理程式会接收到五个讯息，如表 6-11 所示：

表 6-11

讯息	按键或者代码
----	--------

WM_KEYDOWN	虚拟键码 VK_SHIFT (0x10)
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「A」的字元代码 (0x41)
WM_KEYUP	「A」的虚拟键码 (0x41)
WM_KEYUP	虚拟键码 VK_SHIFT (0x10)

Shift 键本身不产生字元讯息。

如果使用者按住 A 键，以使自动重复产生一系列的按键，那么对每条 WM_KEYDOWN 讯息，都会得到一条字元讯息，如表 6-12 所示：

表 6-12

讯息	按键或者代码
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「a」的字元代码 (0x61)
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「a」的字元代码 (0x61)
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「a」的字元代码 (0x61)
WM_KEYDOWN	「A」的虚拟键码 (0x41)
WM_CHAR	「a」的字元代码 (0x61)
WM_KEYUP	「A」的虚拟键码 (0x41)

如果某些 WM_KEYDOWN 讯息的重复计数大於 1，那么相应的 WM_CHAR 讯息将具有同样的重复计数。

组合使用 Ctrl 键与字母键会产生从 0x01 (Ctrl-A) 到 0x1A (Ctrl-Z) 的 ASCII 控制代码，其中的某些控制代码也可以由表 6-13 列出的键产生：

表 6-13

按键	字元代码	产生方法	ANSI C 控制字元
Backspace	0x08	Ctrl-H	\b
Tab	0x09	Ctrl-I	\t
Ctrl-Enter	0x0A	Ctrl-J	\n
Enter	0x0D	Ctrl-M	\r
Esc	0x1B	Ctrl-[

最右列给出了在 ANSI C 中定义的控制字元，它们用於描述这些键的字元代码。

有时 Windows 程式将 Ctrl 与字母键的组合用作功能表加速键（我将在第十章讨论），此时，不会将字母键转换成字元讯息。

处理控制字元

处理按键和字元讯息的基本规则是：如果需要读取输入到视窗的键盘字元，那么您可以处理 WM_CHAR 讯息。如果需要读取游标键、功能键、Delete、Insert、Shift、Ctrl 以及 Alt 键，那么您可以处理 WM_KEYDOWN 讯息。

但是 Tab 键怎么办？Enter、Backspace 和 Escape 键又怎么办？传统上，这些键都产生表 6-13 列出的 ASCII 控制字元。但是在 Windows 中，它们也产生虚拟键码。这些键应该在处理 WM_CHAR 或者在处理 WM_KEYDOWN 期间处理吗？

经过 10 年的考虑（回顾这些年来我写过的 Windows 程式码），我更喜欢将 Tab、Enter、Backspace 和 Escape 键处理成控制字元，而不是虚拟键。我通常这样处理 WM_CHAR：

```
case WM_CHAR:
    //其他行程式
    switch (wParam)
    {
    case '\b':           // backspace
        //其他行程式
        break ;

    case '\t':           // tab
        //其他行程式
        break ;

    case '\n':           // linefeed
        //其他行程式
        break ;

    case '\r':           // carriage return
        //其他行程式
        break ;

    default:             // character codes
        //其他行程式
        break ;
    }
    return 0 ;
```

死字元讯息

Windows 程式经常忽略 WM_DEADCHAR 和 WM_SYSDEADCHAR 讯息，但您应该明确地知道死字元是什么，以及它们工作的方式。

在某些非 U.S. 英语键盘上，有些键用於给字母加上音调。因为它们本身不产生字元，所以称之为「死键」。例如，使用德语键盘时，对於 U.S. 键盘上的

+/=键，德语键盘的对应位置就是一个死键，未按下 Shift 键时它用於标识锐音，按下 Shift 键时则用於标识抑音。

当使用者按下这个死键时，视窗讯息处理程式接收到一个 wParam 等於音调本身的 ASCII 或者 Unicode 代码的 WM_DEADCHAR 讯息。当使用者再按下可以带有此音调的字母键（例如 A 键）时，视窗讯息处理程式会接收到 WM_CHAR 讯息，其中 wParam 等於带有音调的字母「a」的 ANSI 代码。

因此，使用者程式不需要处理 WM_DEADCHAR 讯息，原因是 WM_CHAR 讯息已含有程式所需要的所有资讯。Windows 的做法甚至还设计了内部错误处理。如果在死键之後跟有不能带此音调符号的字母（例如「s」），那么视窗讯息处理程式将在一行接收到两条 WM_CHAR 讯息——前一个讯息的 wParam 等於音调符号本身的 ASCII 代码（与传递到 WM_DEADCHAR 讯息的 wParam 值相同），第二个讯息的 wParam 等於字母 s 的 ASCII 代码。

当然，要感受这种做法的运作方式，最好的方法就是实际操作。您必须载入使用死键的外语键盘，例如前面讲过的德语键盘。您可以这样设定：在「控制台」中选择「键盘」，然後选择「语系」页面标签。然後您需要一个應用程式，该程式可以显示它接收的每一个键盘讯息的详细资讯。下面的 KEYVIEW1 就是这样的程式。

键盘讯息和字元集

本章剩下的范例程式有缺陷。它们不能在所有版本的 Windows 下都正常执行。这些缺陷不是特意引过程式码中的；事实上，您也许永远不会遇到这些缺陷。只有在不同的键盘语言和键盘布局间切换，以及在中位元组字元集的远东版 Windows 下执行程式时，这些问题才会出现——所以我不愿将它们称为「错误」。

不过，如果程式使用 Unicode 编译并在 Windows NT 下执行，那么程式会执行得更好。我在第二章提到过这个问题，并且展示了 Unicode 对简化棘手的国际化问题的重要性。

KEYVIEW1 程式

了解键盘国际化问题的第一步，就是检查 Windows 传递给视窗讯息处理程式的键盘内容和字元讯息。程式 6-2 所示的 KEYVIEW1 会对此有所帮助。该程式在显示区域显示 Windows 向视窗讯息处理程式发送的 8 种不同键盘讯息的全部资讯。

程式 6-2 KEYVIEW1


```
KEYVIEW1.C
/*-----
    KEYVIEW1.C --      Displays Keyboard and Character Messages
                      (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("KeyView1") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Keyboard Message Viewer #1"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
```

```

        return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int    cxClientMax, cyClientMax, cxClient, cyClient, cxChar, cyChar ;
    static int    cLinesMax, cLines ;
    static PMSG   pmsg ;
    static RECT   rectScroll ;
    static TCHAR  szTop[] = TEXT ("Message Key   Char ")
                                TEXT ("Repeat Scan Ext ALT Prev
Tran") ;
    static TCHAR  szUnd[] = TEXT ("_____")
                                TEXT ("_____") ;

    static TCHAR * szFormat[2] = {
        TEXT ("% -13s %3d %-15s%c%6u %4d %3s %3s %4s %4s"),
        TEXT ("% -13s 0x%04X%1s%c %6u %4d %3s %3s %4s %4s") } ;

    static TCHAR * szYes  = TEXT ("Yes") ;
    static TCHAR * szNo   = TEXT ("No") ;
    static TCHAR * szDown = TEXT ("Down") ;
    static TCHAR * szUp   = TEXT ("Up") ;

    static TCHAR * szMessage [] = {
        TEXT ("WM_KEYDOWN"), TEXT ("WM_KEYUP"),
        TEXT ("WM_CHAR"), TEXT ("WM_DEADCHAR"),
        TEXT ("WM_SYSKEYDOWN"), TEXT ("WM_SYSKEYUP"),
        TEXT ("WM_SYSCHAR"), TEXT ("WM_SYSDEADCHAR") } ;
    HDC          hdc ;
    int          i, iType ;
    PAINTSTRUCT  ps ;
    TCHAR          szBuffer[128], szKeyName [32] ;
    TEXTMETRIC    tm ;

    switch (message)
    {
    case WM_CREATE:
    case WM_DISPLAYCHANGE:
        // Get maximum size of client area
        cxClientMax = GetSystemMetrics (SM_CXMAXIMIZED) ;
        cyClientMax = GetSystemMetrics (SM_CYMAXIMIZED) ;

        // Get character size for fixed-pitch font
        hdc = GetDC (hwnd) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
    }
}

```

```
cyChar = tm.tmHeight ;

ReleaseDC (hwnd, hdc) ;
        // Allocate memory for display lines
if (pmsg)
    free (pmsg) ;
    cLinesMax = cyClientMax / cyChar ;
    pmsg = malloc (cLinesMax * sizeof (MSG)) ;
    cLines = 0 ;
    // fall through
case WM_SIZE:
    if (message == WM_SIZE)
    {
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
    }
        // Calculate scrolling rectangle
rectScroll.left      = 0 ;
rectScroll.right = cxClient ;
rectScroll.top       = cyChar ;
rectScroll.bottom= cyChar * (cyClient / cyChar) ;

InvalidateRect (hwnd, NULL, TRUE) ;
return 0 ;

case WM_KEYDOWN:
case WM_KEYUP:
case WM_CHAR:
case WM_DEADCHAR:
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
case WM_SYSDEADCHAR:
        // Rearrange storage array
for (i = cLinesMax - 1 ; i > 0 ; i--)
{
    pmsg[i] = pmsg[i - 1] ;
}
        // Store new message
pmsg[0].hwnd = hwnd ;
pmsg[0].message = message ;
pmsg[0].wParam = wParam ;
pmsg[0].lParam = lParam ;

cLines = min (cLines + 1, cLinesMax) ;
        // Scroll up the display
ScrollWindow (hwnd, 0, -cyChar, &rectScroll, &rectScroll) ;
break ;          // i.e., call DefWindowProc so Sys messages work
```

```

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    SetBkMode (hdc, TRANSPARENT) ;
    TextOut (hdc, 0, 0, szTop, lstrlen (szTop)) ;
    TextOut (hdc, 0, 0, szUnd, lstrlen (szUnd)) ;

    for (i = 0 ; i < min (cLines, cyClient / cyChar - 1) ; i++)
    {
        iType =      pmsg[i].message == WM_CHAR ||
                    pmsg[i].message == WM_SYSCHAR ||
                    pmsg[i].message == WM_DEADCHAR ||
                    pmsg[i].message == WM_SYSDEADCHAR ;

        GetKeyNameText (pmsg[i].lParam, szKeyName,
                        sizeof (szKeyName) / sizeof (TCHAR)) ;

        TextOut (hdc, 0, (cyClient / cyChar - 1 - i) * cyChar, szBuffer,
        wsprintf (szBuffer, szFormat [iType],
        szMessage [pmsg[i].message - WM_KEYFIRST],
        pmsg[i].wParam,
        (PTSTR) (iType ? TEXT (" ") : szKeyName),
        (TCHAR) (iType ? pmsg[i].wParam : ' '),
        LOWORD (pmsg[i].lParam),
        HIWORD (pmsg[i].lParam) & 0xFF,
                0x01000000 & pmsg[i].lParam ? szYes : szNo,
                0x20000000 & pmsg[i].lParam ? szYes : szNo,
                0x40000000 & pmsg[i].lParam ? szDown : szUp,
                0x80000000 & pmsg[i].lParam ? szUp : szDown)) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}

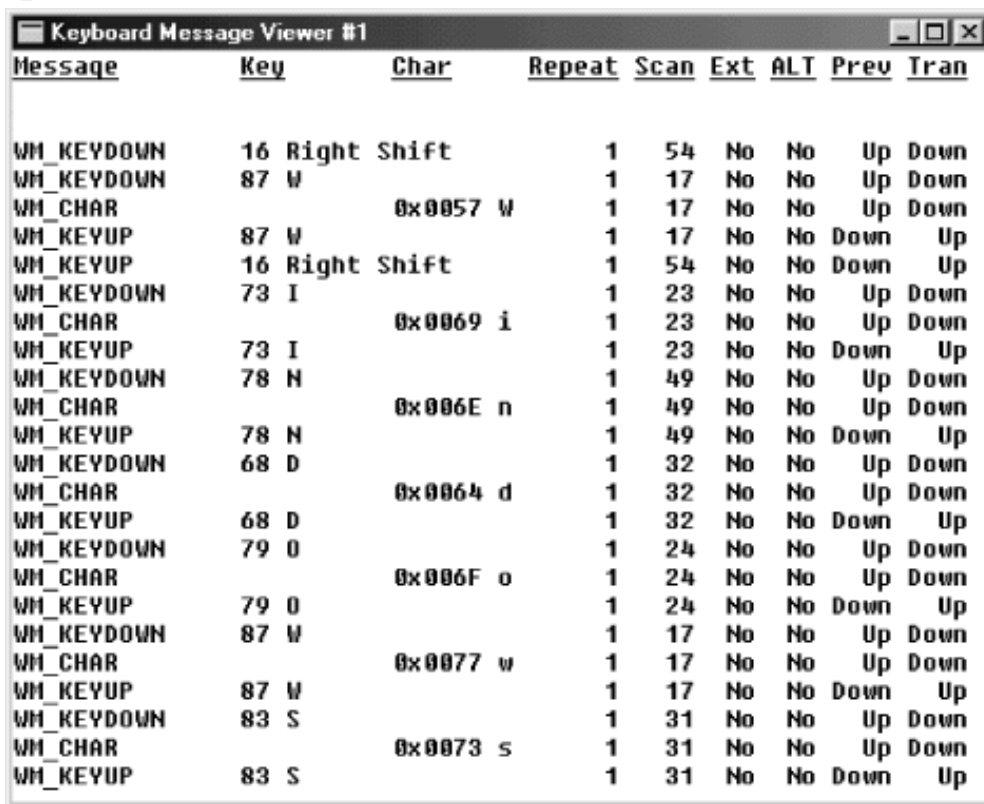
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

KEYVIEW1 显示视窗讯息处理程式接收到的每次按键和字元讯息的内容，并将这些讯息储存在一个 MSG 结构的阵列中。该阵列的大小依据最大化视窗的大小和等宽的系统字体。如果使用者在程式执行时调整了视讯显示的大小（在这种情况下 KEYVIEW1 接收 WM_DISPLAYCHANGE 讯息），将重新分配此阵列。KEYVIEW1 使用标准 C 的 malloc 函式为阵列配置记忆体。

图 6-2 给出了在键入「Windows」之後 KEYVIEW1 的萤幕显示。第一列显示

了键盘讯息；第二列在键名称的前面显示了按键讯息的虚拟键代码，此代码是经由 GetKeyNameText 函式取得的；第三列（标注为「Char」）在字元本身的後面显示字元讯息的十六进位字元代码。其余六列显示了 lParam 讯息参数中六个栏位的状态。



Message	Key	Char	Repeat	Scan	Ext	ALT	Prev	Tran
WM_KEYDOWN	16 Right Shift		1	54	No	No	Up	Down
WM_KEYDOWN	87 W		1	17	No	No	Up	Down
WM_CHAR		0x0057 W	1	17	No	No	Up	Down
WM_KEYUP	87 W		1	17	No	No	Down	Up
WM_KEYUP	16 Right Shift		1	54	No	No	Down	Up
WM_KEYDOWN	73 I		1	23	No	No	Up	Down
WM_CHAR		0x0069 i	1	23	No	No	Up	Down
WM_KEYUP	73 I		1	23	No	No	Down	Up
WM_KEYDOWN	78 N		1	49	No	No	Up	Down
WM_CHAR		0x006E n	1	49	No	No	Up	Down
WM_KEYUP	78 N		1	49	No	No	Down	Up
WM_KEYDOWN	68 D		1	32	No	No	Up	Down
WM_CHAR		0x0064 d	1	32	No	No	Up	Down
WM_KEYUP	68 D		1	32	No	No	Down	Up
WM_KEYDOWN	79 O		1	24	No	No	Up	Down
WM_CHAR		0x006F o	1	24	No	No	Up	Down
WM_KEYUP	79 O		1	24	No	No	Down	Up
WM_KEYDOWN	87 W		1	17	No	No	Up	Down
WM_CHAR		0x0077 w	1	17	No	No	Up	Down
WM_KEYUP	87 W		1	17	No	No	Down	Up
WM_KEYDOWN	83 S		1	31	No	No	Up	Down
WM_CHAR		0x0073 s	1	31	No	No	Up	Down
WM_KEYUP	83 S		1	31	No	No	Down	Up

图 6-2 KEYVIEW1 的萤幕显示

为便於以分行的方式显示此资讯，KEYVIEW1 使用了等宽字体。与前一章所讨论的一样，这需要呼叫 GetStockObject 和 SelectObject：

```
SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
```

KEYVIEW1 在显示区域上部画了一个标题以确定分成九行。此列文字带有底线。虽然可以建立一种带底线的字体，但这里使用了另一种方法。我定义了两个字串变数 szTop（有文字）和 szUnd（有底线），并在 WM_PAINT 讯息处理期间将它们同时显示在视窗顶部的同一位置。通常，Windows 以一种「不透明」的方式显示文字，也就是说显示字元时 Windows 将擦除字元背景区。这将导致第二个字串（szUnd）擦除掉前一个（szTop）。要防止这一现象的发生，可将装置内容切换到「透明」模式：

```
SetBkMode (hdc, TRANSPARENT) ;
```

这种加底线的方法只有在使用等宽字体时才可行。否则，底线字元将无法与显现在底线上面的字元等宽。

外语键盘问题

如果您执行美国英语版本的 Windows, 那么您可安装不同的键盘布局, 并输入外语。可以在 **控制台** 的 **键盘** 中安装外语键盘布局。选择 **语系** 页面标签, 按下 **新增** 键。要查看死键的工作方式, 您可能想安装「德语」键盘。此外, 我还要讨论「俄语」和「希腊语」的键盘布局, 因此您也可安装这些键盘布局。如果在「键盘」显示的列表中找到「俄语」和「希腊语」的键盘布局, 则需要安装多语系支援: 从「控制台」中选择 **新增/删除** 程式, 然後选择 **Windows 安装程式** 页面标签, 确认选中 **多语系支援** 核取方块。在任何情况下, 这些变更都需要原始的 Windows 光碟。

安装完其他键盘布局後, 您将在工作列右侧的通知区看到一个带有两个字母代码的蓝色框。如果内定的是英语, 那么这两个字母是「EN」。单击此图示, 将得到所有已安装键盘布局的列表。从中单击需要的键盘布局即可更改目前活动程式的键盘。此改变只影响目前活动的程式。

现在开始进行实验。不使用 UNICODE 识别字定义来编译 KEYVIEW1 程式 (在本书附带的光碟中, 非 Unicode 版本的 KEYVIEW1 程式位於 RELEASE 子目录)。在美国英语版本的 Windows 下执行该程式, 并输入字元『abcde』。WM_CHAR 讯息与您所期望的一样: ASCII 字元代码 0x61、0x62、0x63、0x64 和 0x65 以及字母 a、b、c、d 和 e。

现在, KEYVIEW1 还在执行, 选择德语键盘布局。按下=键然後输入一个母音 (a、e、i、o 或者 u)。=键将产生一个 WM_DEADCHAR 讯息, 母音产生一个 WM_CHAR 讯息和 (单独的) 字元代码 0xE1、0xE9、0xED、0xF3、0xFA 和字元 á、é、í、ó 或 ú。这就是死键的工作方式。

现在选择希腊键盘布局。输入『abcde』, 您会得到什么? 您将得到 WM_CHAR 讯息和字元代码 0xE1、0xE2、0xF8、0xE4、0xE5 和字元 á、â、ç、ä 和 å。在这里有些字元不能正确显示。难道您不应该得到希腊字母表中的字母吗?

现在切换到俄语键盘并重新输入『abcde』。现在您得到 WM_CHAR 讯息和字元代码 0xF4、0xE8、0xF1、0xE2 和 0xF3, 以及字元 ô、è、ñ、â 和 ó。而且, 还是有些字母不能正常显示。您应从斯拉夫字母表中得到这些字母。

问题在於: 您已经切换键盘以产生不同的字元代码, 但您还没有将此切换通知 GDI, 好让 GDI 能选择适当的符号来显示解释这些字元代码。

如果您非常勇敢, 还有可用的备用 PC, 并且是专业或全球版 Microsoft Developer Network (MSDN) 的订阅户, 那么您也许想安装 (例如) 希腊版的 Windows, 您还可以把那四种键盘布局 (英语、希腊语、德语和俄语) 安装上去。现在执行 KEYLOOK1, 切换到英语键盘布局, 然後输入『abcde』。您应得到 ASCII

字元代码 0x61、0x62、0x63、0x64 和 0x65 以及字元 a、b、c、d 和 e（并且您可以放心：即使在希腊版，ASCII 还是正常通行的）。

在希腊版的 Windows 中，切换到希腊键盘布局并输入『abcde』。您将得到 WM_CHAR 讯息和字元代码 0xE1、0xE2、0xF8、0xE4 和 0xE5。这与您在安装希腊键盘布局的英语版 Windows 中得到的字元代码相同。但现在显示的字元是 τ 、 β 、 ψ 、 δ 和 ϵ 。这些确实是小写的希腊字母 alpha、beta、psi、delta 和 epsilon（gamma 怎么了？是这样，如果使用希腊版的 Windows，那么您将使用键帽上带有希腊字母的键盘。与英语 c 相对应的键正好是 psi。gamma 由与英语 g 相对应的键产生。您可在 Nadine Kano 编写的《Developing International Software for Windows 95 and Windows NT》的第 587 页看到完整的希腊字母表）。

继续在希腊版的 Windows 下运行 KEYVIEW1，切换到德语键盘布局。输入『=』键，然後依次输入 a、e、i、o 和 u。您将得到 WM_CHAR 讯息和字元代码 0xE1、0xE9、0xED、0xF3 和 0xFA。这些字元代码与安装德语键盘布局的英语版 Windows 中的一样。不过，显示的字元却是 α 、 ι 、 ν 、 σ 和 \omicron ，而不是正确的 \acute{a} 、 \acute{e} 、 \acute{i} 、 \acute{o} 和 \acute{u} 。

现在切换到俄语键盘并输入『abcde』。您会得到字元代码 0xF4、0xE8、0xF1、0xE2 和 0xF3，这与安装俄语键盘的英语版 Windows 中得到的一样。不过，显示的字元是 τ 、 θ 、 ρ 、 β 和 σ ，而不是斯拉夫字母表中的字母。

您还可安装俄语版的 Windows。现在您可以猜到，英语和俄语键盘都可以工作，而德语和希腊语则不行。

现在，如果您真的很勇敢，您还可安装日语版的 Windows 并执行 KEYVIEW1。如果再依美国键盘输入，那么您将输入英语文字，一切似乎都正常。不过，如果切换到德语、希腊语或者俄语键盘布局，并且试著作上述介绍的任何练习，您将看到以点显示的字元。如果输入大写的字母——无论是带重音符号的德语字母、希腊语字母还是俄语字母——您将看到这些字母显示为日语中用於拼写外来语的片假名。您也许对输入片假名感兴趣，但那不是德语、希腊语或者俄语。

远东版本的 Windows 包括一个称作「输入法编辑器」（IME）的实用程式，该程式显示为浮动的工具列，它允许您用标准键盘输入象形文字，即汉语、日语和朝鲜语中使用的复杂字元。一般来说，输入一组字母後，组成的字元将显示在另一个浮动视窗内。然後按 **Enter** 键，合成的字元代码就发送到了活动视窗（即 KEYVIEW1）。KEYVIEW1 几乎没什么回应——WM_CHAR 讯息带来的字元代码大於 128，但这些代码没有意义（Nadine Kano 的书中有许多关於使用 IME 的内容）。

这时，我们已经看到了许多 KEYLOOK1 显示错误字元的例子——当执行安装了俄语或希腊语键盘布局的英语版 Windows 时，当执行安装了俄语或德语键盘布局的希腊版 Windows 时，以及执行安装了德语、俄语或者希腊语键盘布局的俄语版 Windows 时，都是这样。我们也看到了从日语版 Windows 的输入法编辑器输入字元时的错误显示。

字元集和字体

KEYLOOK1 的问题是字体问题。用於在萤幕上显示字元的字体和键盘接收的字元代码不一致。因此，让我们看一下字体。

我将在第十七章进行详细讨论，Windows 支援三类字体——点阵字体、向量字体和（从 Windows 3.1 开始的）TrueType 字体。

事实上向量字体已经过时了。这些字体中的字元由简单的线段组成，但这些线段没有定义填入区域。向量字体可以较好地缩放到任意大小，但字元通常看上去有些单薄。

TrueType 字体是定义了填入区域的文字轮廓字体。TrueType 字体可缩放；而且该字元的定义包括「提示」，以消除可能带来的文字不可见或者不可读的圆整问题。使用 TrueType 字体，Windows 就真正实现了 WYSIWYG（「所见即所得」），即文字在视讯显示器显示与印表机输出完全一致。

在点阵字体中，每个字元都定义为与视讯显示器上的图素对应的位元点阵。点阵字体可拉伸到较大的尺寸，但看上去带有锯齿。点阵字体通常被设计成方便在视讯显示器上阅读的字體。因此，Windows 中的标题列、功能表、按钮和对话方块的显示文字都使用点阵字体。

在内定的装置内容下获得的点阵字体称为系统字体。您可通过呼叫带有 SYSTEM_FONT 识别字的 GetStockObject 函式来获得字体代号。KEYVIEW1 程式选择使用 SYSTEM_FIXED_FONT 表示的等宽系统字体。GetStockObject 函式的另一个选项是 OEM_FIXED_FONT。

这三种字体有（各自的）字体名称——System、FixedSys 和 Terminal。程式可以在 CreateFont 或者 CreateFontIndirect 函式呼叫中使用字体名称来指定字体。这三种字体储存在两组放在 Windows 目录内的 FONTS 子目录下的三个档案中。Windows 使用哪一组档案取决於「控制台」里的「显示器」是选择显示「小字体」还是「大字体」（亦即，您希望 Windows 假定视讯显示器是 96 dpi 的解析度还是 120 dpi 的解析度）。表 6-14 总结了所有的情况：

表 6-14

GetStockObject 识别字	字体名称	小字体档案	大字体档案
SYSTEM_FONT	System	VGASYS.FON	8514SYS.FON
SYSTEM_FIXED_FONT	FixedSys	VGAFIX.FON	8514FIX.FON
OEM_FIXED_FONT	Terminal	VGAOEM.FON	8514OEM.FON

在档案名称中,「VGA」指的是视频图形阵列 (Video Graphics Array), IBM 在 1987 年推出的显示卡。这是 IBM 第一块可显示 640 480 图素大小的 PC 显示卡。如果在「控制台」的「显示器」中选择了「小字体」(表示您希望 Windows 假定视讯显示的解析度为 96 dpi), 则 Windows 使用的这三种字体档案名将以「VGA」开头。如果选择了「大字体」(表示您希望解析度为 120 dpi), Windows 使用的档案名将以「8514」开头。8514 是 IBM 在 1987 年推出的另一种显示卡, 它的最大显示尺寸为 1024 768。

Windows 不希望您看到这些档案。这些档案的属性设定为系统和隐藏, 如果用 Windows Explorer 来查看 FONTS 子目录的内容, 您是不会看到它们的, 即使选择了查看系统和隐藏档案也不行。从开始功能表选择「寻找」选项来寻找档名满足 *.FON 限定条件的档案。这时, 您可以双击档案名来查看字体字元是些什么。

对于许多标准控制项和使用者介面元件, Windows 不使用系统字体。相反地, 使用名称为 MS Sans Serif 的字体 (「MS」代表 Microsoft)。这也是一种点阵字体。档案 (名为 SSERIFE.FON) 包含依据 96 dpi 视讯显示器的字体, 点值为 8、10、12、14、18 和 24。您可在 GetStockObject 函式中使用 DEFAULT_GUI_FONT 识别字来得到该字体。Windows 使用的点值取决于「控制台」的「显示」中选择的显示解析度。

到目前为止, 我已提到四种识别字, 利用这四种识别字, 您可以用 GetStockObject 来获得用于装置内容的字体。还有三种其他字体识别字: ANSI_FIXED_FONT、ANSI_VAR_FONT 和 DEVICE_DEFAULT_FONT。为了开始处理键盘和字元显示问题, 让我们先看一下 Windows 中的所有备用字体。显示这些字体的程式是 STOKFONT, 如程式 6-3 所示。

程式 6-3 STOKFONT

```
STOKFONT.C
/*-----
    STOKFONT.C --      Stock Font Objects
                      (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
```

```

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("StokFont") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Stock Fonts"),
                          WS_OVERLAPPEDWINDOW
WS_VSCROLL,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static struct
    {

```

```

        int      idStockFont ;
        TCHAR *  szStockFont ;
    }
    stockfont [] = { OEM_FIXED_FONT,          "OEM_FIXED_FONT",
                    ANSI_FIXED_FONT, "ANSI_FIXED_FONT",
                    ANSI_VAR_FONT,         "ANSI_VAR_FONT",
                    SYSTEM_FONT,           "SYSTEM_FONT",
                    DEVICE_DEFAULT_FONT, "DEVICE_DEFAULT_FONT",
                    SYSTEM_FIXED_FONT,
                    "SYSTEM_FIXED_FONT",
                    DEFAULT_GUI_FONT,
                    "DEFAULT_GUI_FONT" } ;

    static int  iFont, cFonts = sizeof stockfont / sizeof stockfont[0] ;
    HDC          hdc ;
    int          i, x, y, cxGrid, cyGrid ;
    PAINTSTRUCT ps ;
    TCHAR          szFaceName [LF_FACESIZE], szBuffer [LF_FACESIZE +
64] ;
    TEXTMETRIC  tm ;
    switch (message)
    {
    case WM_CREATE:
        SetScrollRange (hwnd, SB_VERT, 0, cFonts - 1, TRUE) ;
        return 0 ;

    case WM_DISPLAYCHANGE:
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case WM_VSCROLL:
        switch (LOWORD (wParam))
        {
        case SB_TOP:          iFont = 0 ;          break ;
        case SB_BOTTOM:       iFont = cFonts - 1 ; break ;
        case SB_LINEUP:       case SB_PAGEUP:      iFont -= 1 ;          break ;
        case SB_LINEDOWN:
        case SB_PAGEDOWN:      iFont += 1 ;          break ;
        case SB_THUMBPOSITION: iFont = HIWORD (wParam) ; break ;
        }
        iFont = max (0, min (cFonts - 1, iFont)) ;
        SetScrollPos (hwnd, SB_VERT, iFont, TRUE) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case WM_KEYDOWN:
        switch (wParam)

```

```

    {
    case VK_HOME: SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
break ;

    case VK_END:  SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;   break ;
    case VK_PRIOR:
    case VK_LEFT:
    case VK_UP:    SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;   break ;
    case VK_NEXT:
    case VK_RIGHT:
    case VK_DOWN:  SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ; break ;
    }
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectObject (hdc, GetStockObject (stockfont[iFont].idStockFont)) ;
    GetTextFace (hdc, LF_FACESIZE, szFaceName) ;
    GetTextMetrics (hdc, &tm) ;
    cxGrid = max (3 * tm.tmAveCharWidth, 2 * tm.tmMaxCharWidth) ;
    cyGrid = tm.tmHeight + 3 ;

    TextOut (hdc, 0, 0, szBuffer,
    wsprintf ( szBuffer, TEXT (" %s: Face Name = %s, CharSet = %i"),
                stockfont[iFont].szStockFont,
                szFaceName, tm.tmCharSet)) ;

    SetTextAlign (hdc, TA_TOP | TA_CENTER) ;
    // vertical and horizontal lines
    for (i = 0 ; i < 17 ; i++)
    {
        MoveToEx (hdc, (i + 2) * cxGrid, 2 * cyGrid, NULL) ;
        LineTo   (hdc, (i + 2) * cxGrid, 19 * cyGrid) ;

        MoveToEx (hdc, cxGrid, (i + 3) * cyGrid, NULL) ;
        LineTo   (hdc, 18 * cxGrid, (i + 3) * cyGrid) ;
    }

    // vertical and horizontal headings

    for (i = 0 ; i < 16 ; i++)
    {
        TextOut (hdc, (2 * i + 5) * cxGrid / 2, 2 * cyGrid + 2, szBuffer,
        wsprintf (szBuffer, TEXT ("%X-"), i)) ;

        TextOut (hdc, 3 * cxGrid / 2, (i + 3) * cyGrid + 2, szBuffer,
        wsprintf (szBuffer, TEXT ("-%X"), i)) ;
    }

    // characters

```

```

for (y = 0 ; y < 16 ; y++)
for (x = 0 ; x < 16 ; x++)
{
    TextOut (hdc, (2 * x + 5) * cxGrid / 2,
              (y + 3) * cyGrid + 2, szBuffer,
              wsprintf (szBuffer, TEXT ("%c"), 16 * x + y)) ;
}

EndPaint (hwnd, &ps) ;
return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}

return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

这个程式相当简单。它使用卷动列和游标移动键让您选择显示七种备用字体之一。该程式在一个网格中显示一种字体的 256 个字元。顶部的标题和网格的左侧显示字元代码的十六进位值。

在显示区域的顶部, STOKFONT 用 GetStockObject 函式显示用於选择字体的识别字。它还显示由 GetTextFace 函式得到的字体样式名称和 TEXTMETRIC 结构的 tmCharSet 栏位。这个「字元集识别字」对理解 Windows 如何处理外语版本的 Windows 是非常重要的。

如果在美国英语版本的 Windows 中执行 STOKFONT, 那么您看到的第一个画面将显示使用 OEM_FIXED_FONT 识别字呼叫 GetStockObject 函式得到的字体。如图 6-3 所示。

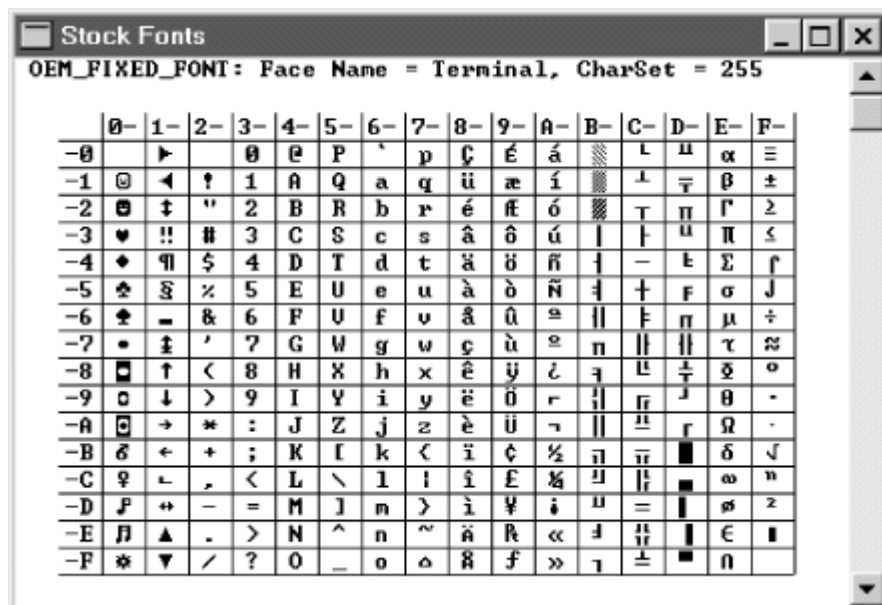


图 6-3 美国版 Windows 中的 OEM_FIXED_FONT

在本字元集中（与本章其他部分一样），您将看到一些 ASCII。但请记住 ASCII 是 7 位元代码，它定义了从代码 0x20 到 0x7E 的可显示字元。到 IBM 开发出 IBM PC 原型机时，8 位元位元组代码已被稳固地建立起来，因此可使用全 8 位元代码作为字元代码。IBM 决定使用一系列由线和方块组成的字元、带重音字母、希腊字母、数学符号和一些其他字元来扩展 ASCII 字元集。许多文字模式的 MS-DOS 程式在其萤幕显示中都使用绘图字元，并且许多 MS-DOS 程式都在档案中使用了一些扩展字元。

这个特殊的字元集给 Windows 最初的开发者带来了一个问题。一方面，因为 Windows 有完整的图形程式设计语言，所以线和方块字元在 Windows 中不需要。因此，这些字元使用的 48 个代码最好用於许多西欧语言所需要的附带重音字母。另一方面，IBM 字元集定义了一个无法完全忽略的标准。

因此，Windows 最初的开发者决定支援 IBM 字元集，但将其重要性降低到第二位——它们大多用於在视窗中执行的旧 MS-DOS 应用程式，和需要使用由 MS-DOS 应用程式建立档案的 Windows 程式。Windows 应用程式不使用 IBM 字元集，并且随著时间的推移，其重要性日渐衰退。然而，如果需要，您还是可以使用。在此环境下，「OEM」指的就是「IBM」。

（您应知道外语版本的 Windows 不必支援与美国英语版相同的 OEM 字元集。其他国家有其自己的 MS-DOS 字元集。这是个独立的问题，就不在本书中讨论了。）

因为 IBM 字元集被认为不适合 Windows，於是选择了另一种扩展字元集。此字元集称作「ANSI 字元集」，由美国国家标准协会(American National Standards Institute)制定，但它实际上是 ISO (International Standards Organization, 国际标准化组织) 标准，也就是 ISO 标准 8859。它还称为 Latin 1、Western European、或者内码表 1252。图 6-4 显示了 ANSI 字元集的一个版本——美国英语版 Windows 的系统字体。

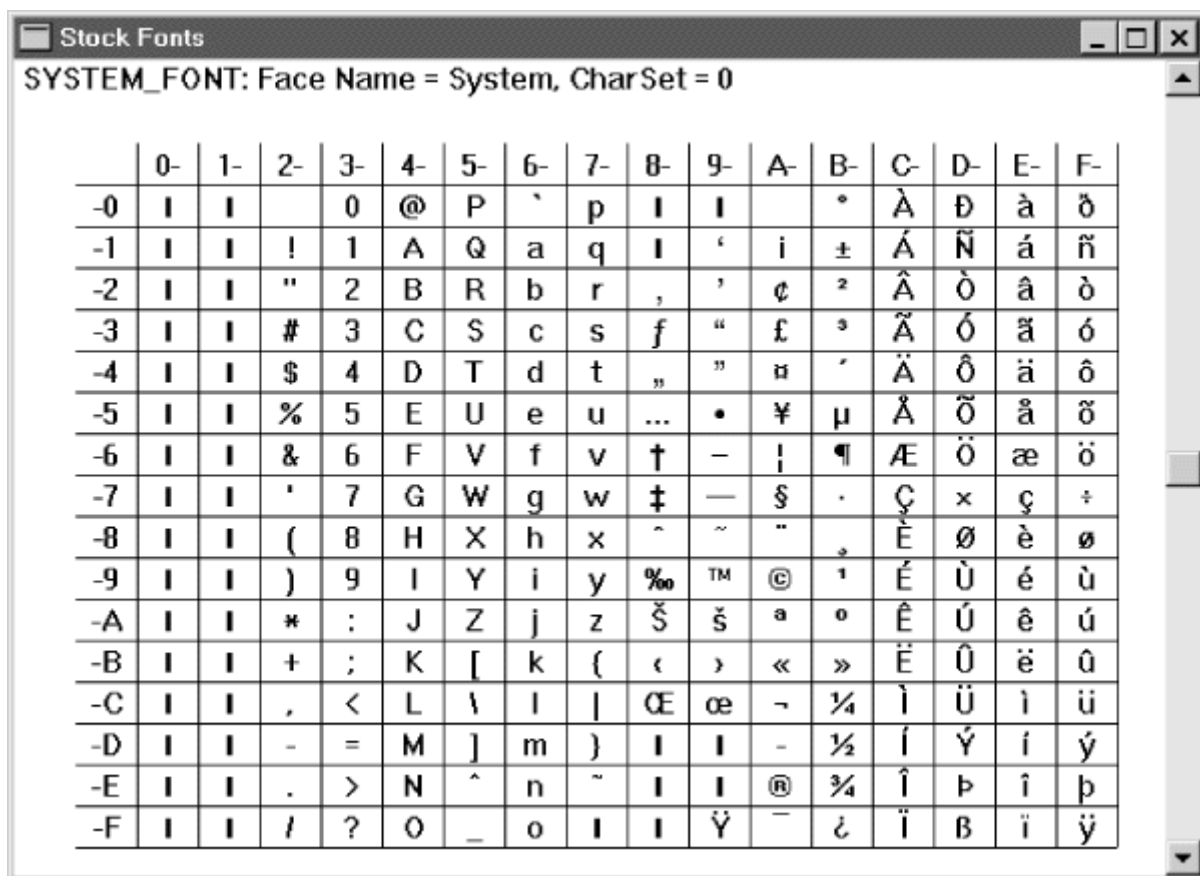


图 6-4 美国版 Windows 中的 SYSTEM_FONT

粗的垂直条表示这些字元代码没有定义。注意，代码 0x20 到 0x7E 还是 ASCII。此外，ASCII 控制字元（0x00 到 0x1F 以及 0x7F）并不是可显示字元。它们本应如此。

代码 0xC0 到 0xFF 使得 ANSI 字元集对外语版 Windows 来说非常重要。这些代码提供 64 个在西欧语言中普遍使用的字元。字元 0xA0，看起来像空格，但实际上定义为非断开空格，例如「WW II」中的空格。

之所以说这是 ANSI 字元集的「一个版本」，是因为存在代码 0x80 到 0x9F 的字元。等宽的系统字体只包括其中的两个字元，如图 6-5 所示。

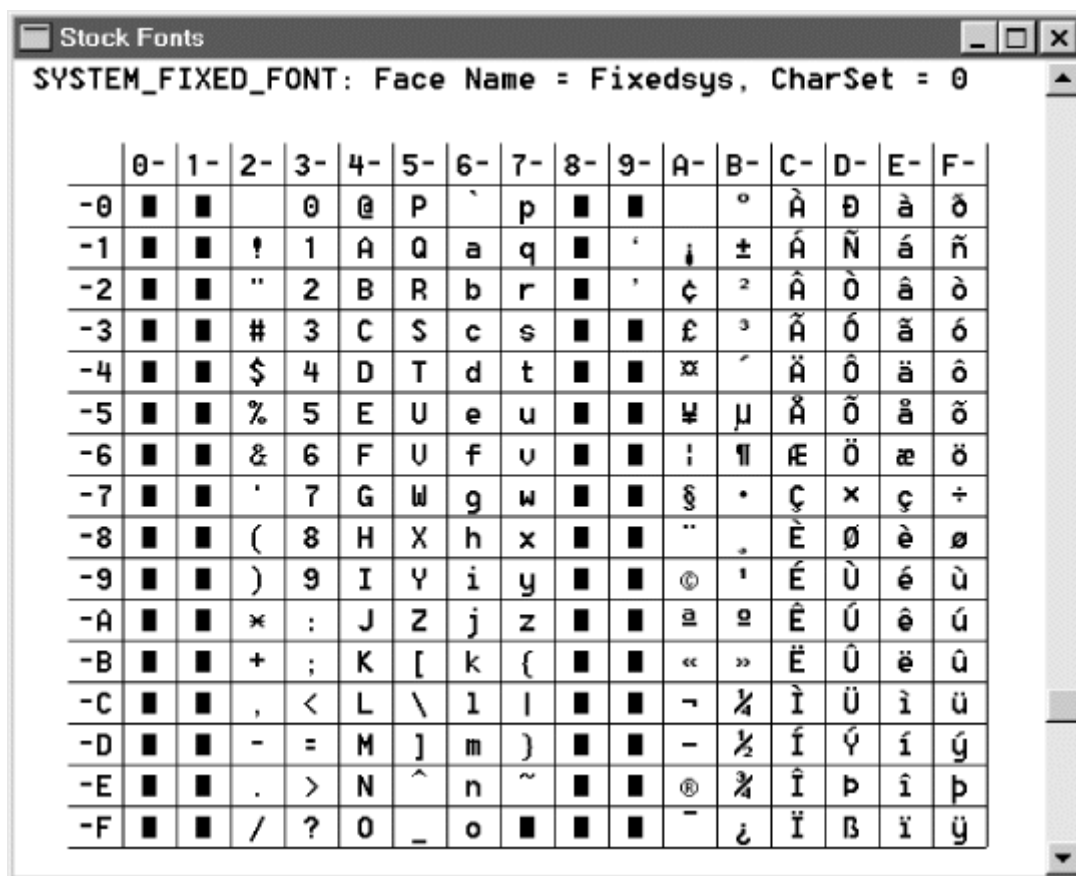


图 6-5 美国版 Windows 中的 SYSTEM_FIXED_FONT

在 Unicode 中,代码 0x0000 到 0x007F 与 ASCII 相同,代码 0x0080 到 0x009F 复制了 0x0000 到 0x001F 的控制字元,代码 0x00A0 到 0x00FF 与 Windows 中使用的 ANSI 字元集相同。

如果执行德语版的 Windows,那么当您用 SYSTEM_FONT 或者 SYSTEM_FIXED_FONT 识别字来呼叫 GetStockObject 函数时会得到同样的 ANSI 字元集。其他西欧版 Windows 也是如此。ANSI 字元集中含有这些语言所需要的所有字元。

不过,当您执行希腊版的 Windows 时,内定的字元集就改变了。相反地,SYSTEM_FONT 如图 6-6 所示。

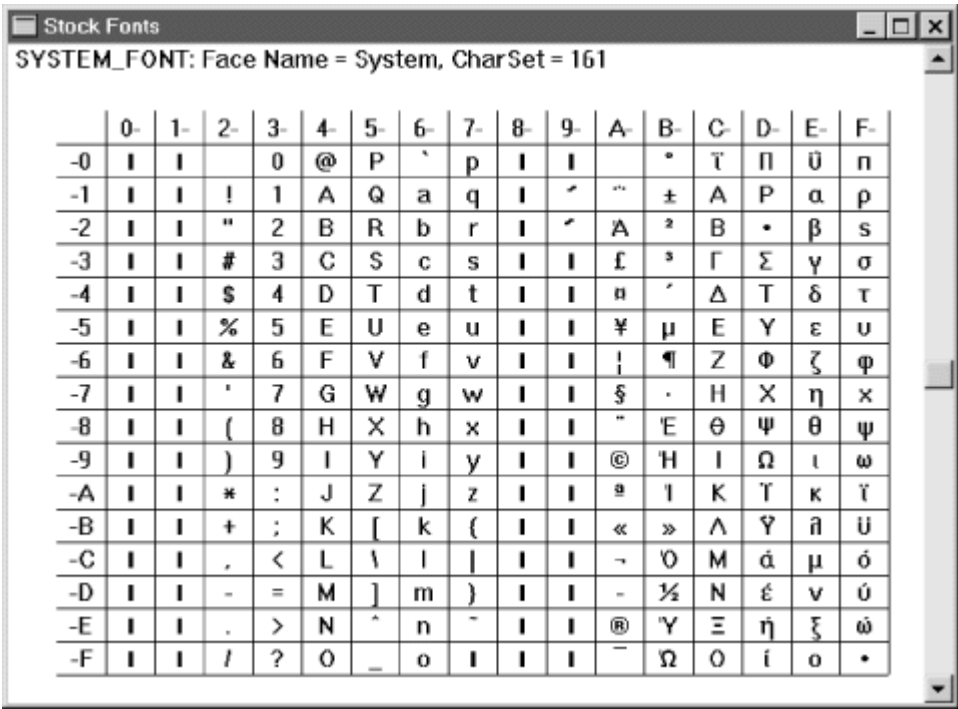


图 6-6 希腊版 Windows 中的 SYSTEM_FONT

SYSTEM_FIXED_FONT 有同样的字元。注意从 0xC0 到 0xFF 的代码。这些代码包含希腊字母表中的大写字母和小写字母。当您执行俄语版 Windows 时，内定的字元集如图 6-7 所示。

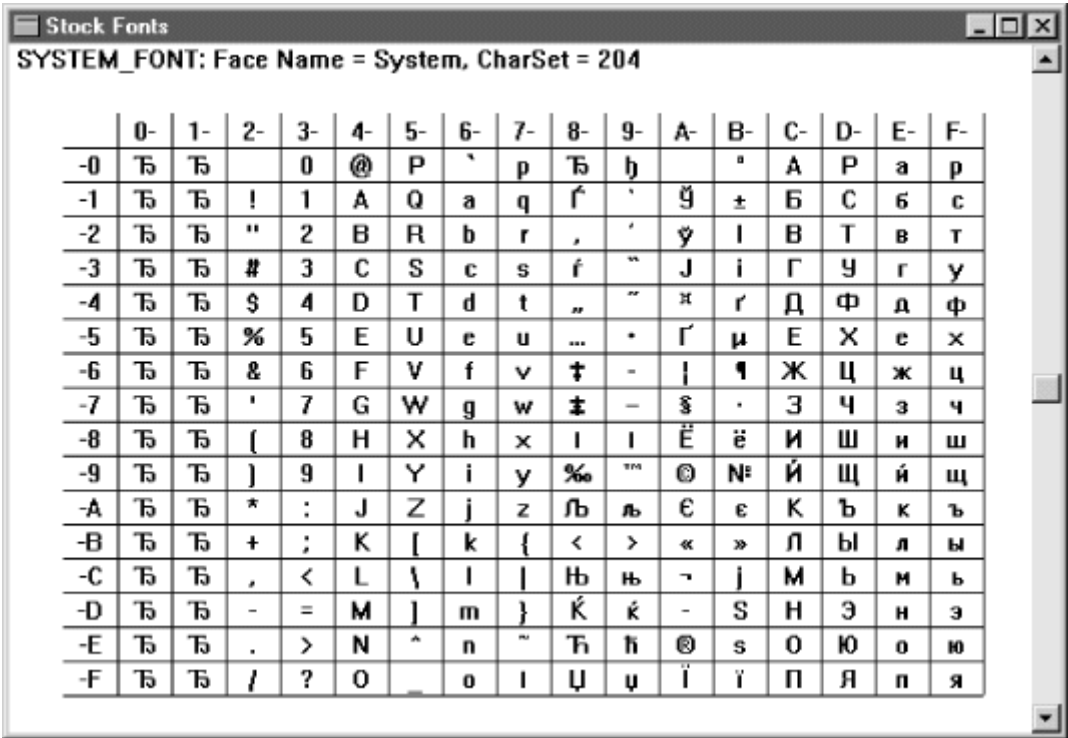


图 6-7 俄语版 Windows 中的 SYSTEM_FONT

此外， 注意斯拉夫字母表中的大写和小写字母占用了代码 0xC0 和 0xFF。

图 6-8 显示了日语版 Windows 的 SYSTEM_FONT。从 0xA5 到 0xDF 的字元都是片假名字母表的一部分。

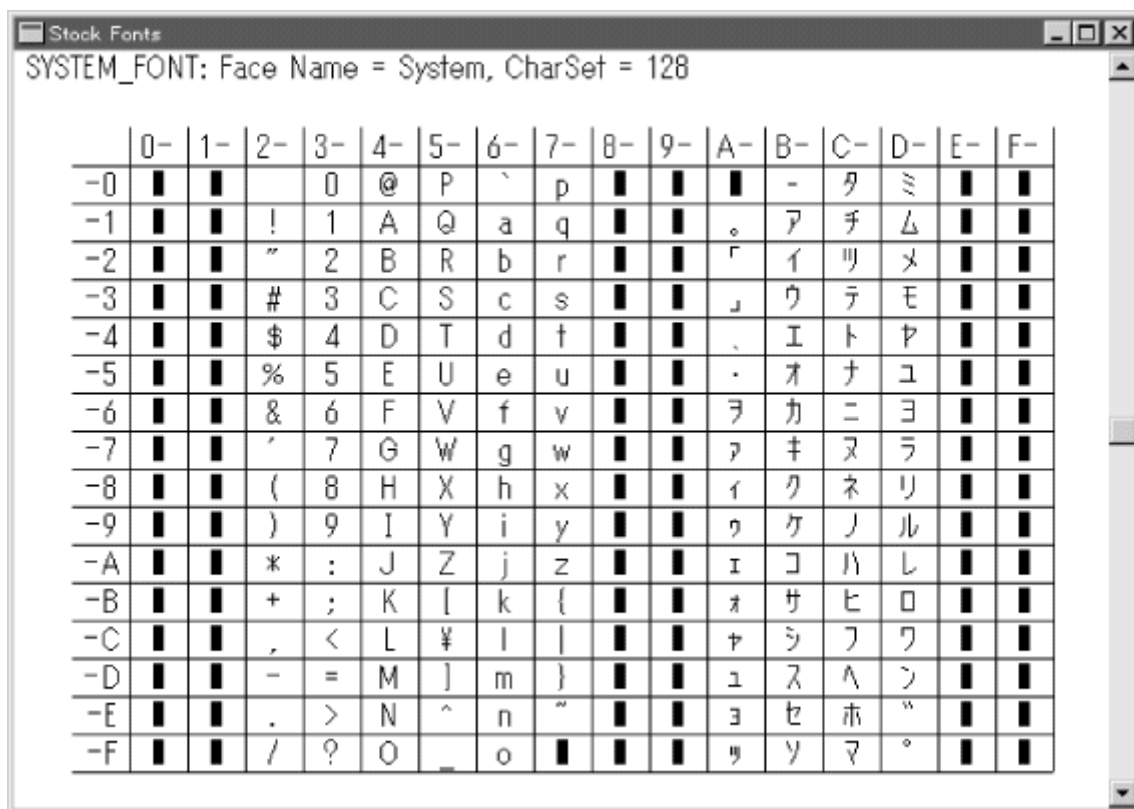


图 6-8 日语版 Windows 中的 SYSTEM_FONT

图 6-8 所示的日文系统字体不同於前面显示的那些，因为它实际上是双位元组字元集 (DBCS)，称为「Shift-JIS」（「JIS」代表日本工业标准，Japanese Industrial Standard）。从 0x81 到 0x9F 以及从 0xE0 到 0xFF 的大多数字元代码实际上只是双位元组代码的第一个位元组，其第二个位元组通常在 0x40 到 0xFC 的范围内（关于这些代码的完整表格，请参见 Nadine Kano 书中的附录 G）。

现在，我们就可以看看 KEYVIEW1 中的问题在哪里：如果您安装了希腊键盘布局并键入『abcde』，不考虑执行的 Windows 版本，Windows 将产生 WM_CHAR 讯息和字元代码 0xE1、0xE2、0xF8、0xE4 和 0xE5。但只有执行带有希腊系统字体的希腊版 Windows 时，这些字元代码才能与 τ 、 β 、 ψ 、 δ 和 ϵ 相对应。

如果您安装了俄语键盘布局并敲入『abcde』，不考虑所使用的 Windows 版本，Windows 将产生 WM_CHAR 讯息和字元代码 0xF4、0xE8、0xF1、0xE2 和 0xF3。但只有在使用俄语版 Windows 或者使用斯拉夫字母表的其他语言版，并且使用斯拉夫系统字体时，这些字元代码才会与字元 ϕ 、и、с、в 和 у 相对应。

如果您安装了德语键盘布局并按下=键（或者位於同一位置的键），然後按下 a、e、i、o 或者 u 键，不考虑使用的 Windows 版本，Windows 将产生 WM_CHAR 讯息和字元代码 0xE1、0xE9、0xED、0xF3 和 0xFA。只有执行西欧版或者美国版的 Windows 时，也就是说有西欧系统字体，这些字元代码才会和字元 $\&\nbsp;$ 、á、é、í、ó 和 ú 相对应。

如果安装了美国英语键盘布局，则您可在键盘上键入任何字元，Windows 将

产生 WM_CHAR 讯息以及与字元正确匹配的字元代码。

Unicode 怎么样？

我在第二章谈到过 Windows NT 支援的 Unicode 有助於为国际市场程式写作。让我们编译一下定义了 UNICODE 识别字的 KEYVIEW1, 并在不同版本的 Windows NT 下执行 (在本书附带的光碟中, Unicode 版的 KEYVIEW1 位於 DEBUG 目录中)。

如果程式编译时定义了 UNICODE 识别字, 则「KeyView1」视窗类别就用 RegisterClassW 函式注册, 而不是 RegisterClassA 函式。这意味著任何带有字元或文字资料的讯息传递给 WndProc 时都将使用 16 位元字元而不是 8 位元字元。特别是 WM_CHAR 讯息, 将传递 16 位元字元代码而不是 8 位元字元代码。

请在美国英语版的 Windows NT 下执行 Unicode 版的 KEYVIEW1。这里假定您已经安装了至少三种我们试验过的键盘布局——即德语、希腊语和俄语。

使用美国英语版的 Windows NT, 并安装了英语或者德语的键盘布局, Unicode 版的 KEYVIEW1 在工作时将与非 Unicode 版相同。它将接收相同的字元代码 (所有 0xFF 或者更低的值), 并显示同样正确的字元。这是因为最初的 256 个 Unicode 字元与 Windows 中使用的 ANSI 字元集相同。

现在切换到希腊键盘布局, 并键入『abcde』。WM_CHAR 讯息将含有 Unicode 字元代码 0x03B1、0x03B2、0x03C8、0x03B4 和 0x03B5。注意, 我们先看到的字元代码值比 0xFF 高。这些 Unicode 字元代码与希腊字母 τ 、 β 、 ψ 、 d 和 ε 相对应。不过, 所有这五个字元都显示为方块! 这是因为 SYSTEM_FIXED_FONT 只含有 256 个字元。

现在切换到俄语键盘布局, 并键入『abcde』。KEYVIEW1 显示 WM_CHAR 讯息和 Unicode 字元代码 0x0444、0x0438、0x0441、0x0432 和 0x0443, 这些字元对应於斯拉夫字母 ϕ 、 μ 、 c 、 B 和 y 。不过, 所有这五个字母也显示为实心方块。

简言之, 非 Unicode 版的 KEYVIEW1 显示错误字元的地方, Unicode 版的 KEYVIEW1 就显示实心方块, 以表示目前的字体没有那种特殊字元。虽然我不愿说 Unicode 版的 KEYVIEW1 是非 Unicode 版的改进, 但事实确实如此。非 Unicode 版显示错误字元, 而 Unicode 版不会这样。

Unicode 和非 Unicode 版 KEYVIEW1 的不同之处主要在两个方面。

首先, WM_CHAR 讯息伴随一个 16 位元字元代码, 而不是 8 位元字元代码。在非 Unicode 版本的 KEYVIEW1 中, 8 位元字元代码的含义取决於目前活动的键盘布局。如果来自德语键盘, 则 0xE1 代码表示 \acute{a} , 如果来自希腊语键盘则代表 α , 如果来自俄语键盘则代表 σ 。在 Unicode 版本程式中, 16 位元字元代码的含义

很明确：**a** 字元是 **0x00E1**，**α** 字元是 **0x03B1**，而 **σ** 字元是 **0x0431**。

第二，Unicode 的 TextOutW 函式显示的字元依据 16 位元字元代码，而不是非 Unicode 的 TextOutA 函式的 8 位元字元代码。因为这些 16 位元字元代码含义明确，GDI 可以确定目前在装置内容中选择的字体是否可显示每个字元。

在美国英语版 Windows NT 下执行 Unicode 版的 KEYVIEW1 多少让人感到有些迷惑，因为它所显示的就好像 GDI 只显示了 0x0000 到 0x00FF 之间的字元代码，而没有显示高於 0x00FF 的代码。也就是说，只是在字元代码和系统字体中 256 个字元之间简单的一对一映射。

然而，如果安装了希腊或者俄语版的 Windows NT，您将发现情况就大不一样了。例如，如果安装了希腊版的 Windows NT，则美国英语、德语、希腊语和俄语键盘将会产生与美国英语版 Windows NT 同样的 Unicode 字元代码。不过，希腊版的 Windows NT 将不显示德语重音字元或者俄语字元，因为这些字元并不在希腊系统字体中。同样，俄语版的 Windows NT 也不显示德语重音字元或者希腊字元，因为这些字元也不在俄语系统字体中。

其中，Unicode 版的 KEYVIEW1 的区别在日语版 Windows NT 下更具戏剧性。您从 IME 输入日文字元，这些字元可以正确显示。唯一的问题是格式：因为日文字元通常看起来非常复杂，它们的显示宽度是其他字元的两倍。

TrueType 和大字体

我们使用的点阵字体（在日文版 Windows 中带有附加字体）最多包括 256 个字元。这是我们所希望的，因为当假定字元代码是 8 位元时，点阵字体档案的格式就跟早期 Windows 时代的样子一样了。这就是为什么当我们使用 SYSTEM_FONT 或者 SYSTEM_FIXED_FONT 时，某些语言中一些字元总不能正确显示（日本系统字体有点不同，因为它是双位元组字元集；大多数字元实际上保存在 TrueType 集合档案中，档案副档名是 .TTC）。

TrueType 字体包含的字元可以多於 256 个。并不是所有 TrueType 字体中的字元都多於 256 个，但 Windows 98 和 Windows NT 中的字体包含多於 256 个字元。或者，安装了多语系支援後，TrueType 字体中也包含多於 256 个字元。在「[控制台](#)」的「[新增/删除程式](#)」中，单击「[Windows 安装程式](#)」页面标签，并确保选中了「[多语系支援](#)」。这个多语系支援包括五个字元集：波罗的海语系、中欧语系、斯拉夫语系、希腊语系和土耳其语系。波罗的海语系字元集用於爱沙尼亚语、拉脱维亚语和立陶宛语。中欧字元集用於阿尔巴尼亚语、捷克语、克罗地亚语、匈牙利语、波兰语、罗马尼亚语、斯洛伐克语和斯洛文尼亚语。斯拉夫字元集用於保加利亚语、白俄罗斯语、俄语、塞尔维亚语和乌

克兰语。

Windows 98 中的 TrueType 字体支援这五种字元集，再加上西欧 (ANSI) 字元集，西欧字元集实际上用於其他所有语言，但远东语言 (汉语、日语和朝鲜语) 除外。支援多种字元集的 TrueType 字体有时也称为「大字体」。在这种情况下下的「大」并不是指字元的大小，而是指数量。

即使在非 Unicode 程式中也可利用大字体，这意味著可以用大字体显示几种不同字母表中的字元。然而，为了要将得到的字体选进装置内容，还需要 GetStockObject 以外的函式。

函式 CreateFont 和 CreateFontIndirect 建立了一种逻辑字体，这与 CreatePen 建立逻辑画笔以及 CreateBrush 建立逻辑画刷的方式类似。CreateFont 用 14 个参数描述要建立的字体。CreateFontIndirect 只有一个参数，但该参数是指向 LOGFONT 结构的指标。LOGFONT 结构有 14 个栏位，分别对应於 CreateFont 函式的参数。我将在第十七章详细讨论这些函式。现在，让我们看一下 CreateFont 函式，但我们只注意其中两个参数，其他参数都设定为 0。

如果需要等宽字体 (就像 KEYVIEW1 程式中使用的)，将 CreateFont 的第 13 个参数设定为 FIXED_PITCH。如果需要非内定字元集的字体 (这也是我们所需要的)，将 CreateFont 的第 9 个参数设定为某个「字元集 ID」。此字元集 ID 将是 WINGDI.H 中定义的下列值之一。我已给出注释，指出和这些字元集相关的内码表：

#define ANSI_CHARSET	0	// 1252 Latin 1 (ANSI)
#define DEFAULT_CHARSET	1	
#define SYMBOL_CHARSET	2	
#define MAC_CHARSET	77	
#define SHIFTJIS_CHARSET	128	// 932 (DBCS, 日本)
#define HANGEUL_CHARSET	129	// 949 (DBCS, 韩文)
#define HANGUL_CHARSET	129	// " "
#define JOHAB_CHARSET	130	// 1361 (DBCS, 韩文)
#define GB2312_CHARSET	134	// 936 (DBCS, 简体中文)
#define CHINESEBIG5_CHARSET	136	// 950 (DBCS, 繁体中文)
#define GREEK_CHARSET	161	// 1253 希腊文
#define TURKISH_CHARSET	162	// 1254 Latin 5 (土耳其文)
#define VIETNAMESE_CHARSET	163	// 1258 越南文
#define HEBREW_CHARSET	177	// 1255 希伯来文
#define ARABIC_CHARSET	178	// 1256 阿拉伯文
#define BALTIC_CHARSET	186	// 1257 波罗的海字集

#define RUSSIAN_CHARSET	204	// 1251 俄文 (斯拉夫语系)
#define THAI_CHARSET	222	// 874 泰文
#define EASTEUROPE_CHARSET	238	// 1250 Latin 2 (中欧语系)
#define OEM_CHARSET	255	// 地区自订

为什么 Windows 对同一个字元集有两个不同的 ID: 字元集 ID 和内码表 ID? 这只是 Windows 中的一种怪癖。注意, 字元集 ID 只需要 1 位元组的储存空间, 这是 LOGFONT 结构中字元集栏位的大小 (试回忆 Windows 1.0 时期, 记忆体和储存空间有限, 每个位元组都必须斤斤计较)。注意, 有许多不同的 MS-DOS 内码表用於其他国家, 但只有一种字元集 ID——OEM_CHARSET——用於 MS-DOS 字元集。

您还会注意到, 这些字元集的值与 STOKFONT 程式最上头的「CharSet」值一致。在美国英语版 Windows 中, 我们看到常备字体的字元集 ID 是 0 (ANSI_CHARSET) 和 255 (OEM_CHARSET)。希腊版 Windows 中的是 161 (GREEK_CHARSET), 在俄语版中的是 204 (RUSSIAN_CHARSET), 在日语版中是 128 (SHIFTJIS_CHARSET)。

在上面的代码中, DBCS 代表双位元组字元集, 用於远东版的 Windows。其他版的 Windows 不支援 DBCS 字体, 因此不能使用那些字元集 ID。

CreateFont 传回 HFONT 值——逻辑字体的代号。您可以使用 SelectObject 将此字体选进装置内容。实际上, 您必须呼叫 DeleteObject 来删除您建立的所有逻辑字体。

大字体解决方案的其他部分是 WM_INPUTLANGCHANGE 讯息。一旦您使用桌面下端的突现式功能表来改变键盘布局, Windows 都会向您的视窗讯息处理程式发送 WM_INPUTLANGCHANGE 讯息。wParam 讯息参数是新键盘布局的字元集 ID。

程式 6-4 所示的 KEYVIEW2 程式实作了键盘布局改变时改变字体的逻辑。

程式 6-4 KEYVIEW2

```
KEYVIEW2.C
/*-----
--
KEYVIEW2.C -- Displays Keyboard and Character Messages
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
```

```

static TCHAR szAppName[] = TEXT ("KeyView2") ;
HWND          hwnd ;
MSG           msg ;
WNDCLASS      wndclass ;

wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc    = WndProc ;
wndclass.cbClsExtra     = 0 ;
wndclass.cbWndExtra     = 0 ;
wndclass.hInstance     = hInstance ;
wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName   = NULL ;
wndclass.lpszClassName = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Keyboard Message Viewer #2"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static DWORD dwCharSet = DEFAULT_CHARSET ;
    static int   cxClientMax, cyClientMax, cxClient, cyClient, cxChar, cyChar ;
    static int   cLinesMax, cLines ;
    static PMSG  pmsg ;
    static RECT  rectScroll ;

```

```

static TCHAR szTop[] = TEXT ("Message Key Char ")
                        TEXT ("Repeat Scan Ext ALT Prev
Tran");
static TCHAR szUnd[] = TEXT ("_____")
                        TEXT ("_____
_____");

static TCHAR * szFormat[2] = {
    TEXT ("%13s %3d %-15s%c%6u %4d %3s %3s %4s %4s"),
    TEXT ("%13s 0x%04X%1s%c %6u %4d %3s %3s %4s %4s") };

static TCHAR * szYes = TEXT ("Yes");
static TCHAR * szNo = TEXT ("No");
static TCHAR * szDown = TEXT ("Down");
static TCHAR * szUp = TEXT ("Up");

static TCHAR * szMessage [] = {
    TEXT ("WM_KEYDOWN"), TEXT ("WM_KEYUP"),
    TEXT ("WM_CHAR"), TEXT ("WM_DEADCHAR"),
    TEXT ("WM_SYSKEYDOWN"), TEXT ("WM_SYSKEYUP"),
    TEXT ("WM_SYSCHAR"), TEXT ("WM_SYSDEADCHAR") };

HDC hdc;
int i, iType;
PAINTSTRUCT ps;
TCHAR szBuffer[128], szKeyName [32];
TEXTMETRIC tm;

switch (message)
{
case WM_INPUTLANGCHANGE:
    dwCharSet = wParam;
    // fall through
case WM_CREATE:
case WM_DISPLAYCHANGE:
    // Get maximum size of client area
    cxClientMax = GetSystemMetrics (SM_CXMAXIMIZED);
    cyClientMax = GetSystemMetrics (SM_CYMAXIMIZED);

    // Get character size for fixed-pitch font
    hdc = GetDC (hwnd);
    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
                                dwCharSet, 0, 0, 0, FIXED_PITCH,
NULL));
    GetTextMetrics (hdc, &tm);
    cxChar = tm.tmAveCharWidth;
    cyChar = tm.tmHeight;

    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT)));

```



```
ReleaseDC (hwnd, hdc) ;

    // Allocate memory for display lines
if (pmsg)
    free (pmsg) ;
cLinesMax = cyClientMax / cyChar ;
pmsg = malloc (cLinesMax * sizeof (MSG)) ;
cLines = 0 ;
    // fall through
case WM_SIZE:
if (message == WM_SIZE)
{
    cxClient      = LOWORD (lParam) ;
    cyClient      = HIWORD (lParam) ;
}

    // Calculate scrolling rectangle

rectScroll.left      = 0 ;
rectScroll.right = cxClient ;
rectScroll.top       = cyChar ;
rectScroll.bottom= cyChar * (cyClient / cyChar) ;

InvalidateRect (hwnd, NULL, TRUE) ;

if (message == WM_INPUTLANGCHANGE)
    return TRUE ;
return 0 ;

case WM_KEYDOWN:
case WM_KEYUP:
case WM_CHAR:
case WM_DEADCHAR:
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
case WM_SYSDEADCHAR:
    // Rearrange storage array
for (i = cLinesMax - 1 ; i > 0 ; i--)
{
    pmsg[i] = pmsg[i - 1] ;
}

    // Store new message
pmsg[0].hwnd = hwnd ;
pmsg[0].message = message ;
pmsg[0].wParam = wParam ;
pmsg[0].lParam = lParam ;

cLines = min (cLines + 1, cLinesMax) ;
```

```

        // Scroll up the display
        ScrollWindow (hwnd, 0, -cyChar, &rectScroll, &rectScroll) ;
        break ;      // ie, call DefWindowProc so Sys messages work

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
        dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
    SetBkMode (hdc, TRANSPARENT) ;
    TextOut (hdc, 0, 0, szTop, strlen (szTop)) ;
    TextOut (hdc, 0, 0, szUnd, strlen (szUnd)) ;

    for (i = 0 ; i < min (cLines, cyClient / cyChar - 1) ; i++)
    {
        iType =      pmsg[i].message == WM_CHAR ||
                    pmsg[i].message == WM_SYSCHAR ||
                    pmsg[i].message == WM_DEADCHAR ||
                    pmsg[i].message == WM_SYSDEADCHAR ;

        GetKeyNameText (pmsg[i].lParam, szKeyName,
            sizeof (szKeyName) / sizeof (TCHAR)) ;

        TextOut (hdc, 0, (cyClient / cyChar - 1 - i) * cyChar, szBuffer,
            wsprintf ( szBuffer, szFormat [iType],
                szMessage [pmsg[i].message - WM_KEYFIRST],
                pmsg[i].wParam,
                (PTSTR) (iType ? TEXT (" ") : szKeyName),
                (TCHAR) (iType ? pmsg[i].wParam : ' '),
                LOWORD (pmsg[i].lParam),
                HIWORD (pmsg[i].lParam) & 0xFF,
                0x01000000 & pmsg[i].lParam ? szYes : szNo,
                0x20000000 & pmsg[i].lParam ? szYes : szNo,
                0x40000000 & pmsg[i].lParam ? szDown : szUp,
                0x80000000 & pmsg[i].lParam ? szUp : szDown));
    }
    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

注意，键盘输入语言改变後，KEYVIEW2 就清除画面并重新分配储存空间。

这样做有两个原因：第一，因为 KEYVIEW2 并不是某种字体专用的，当输入语言改变时字体文字的大小也会改变。程式需要根据新字元大小重新计算某些变数。第二，在接收每个字元讯息时，KEYVIEW2 并不有效地保留字元集 ID。因此，如果键盘输入语言改变了，而且 KEYVIEW2 需要重画显示区域时，所有的字元将用新字体显示。

第十七章将详细讨论字体和字元集。如果您想深入研究国际化问题，可以在 /Platform SDK/Windows Base Services/International Features 找到需要的文件，还有许多基础资讯则位於 /Platform SDK/Windows Base Services/General Library/String Manipulation。

插入符号（不是游标）

当您往程式中输入文字时，通常有一个底线、竖条或者方框来指示输入的下一个字元将出现在萤幕上的位置。这个标志通常称为「游标」，但是在 Windows 下写程式，您必须改变这个习惯。在 Windows 中，它称为「插入符号」。「游标」是指表示滑鼠位置的那个点阵图图像。

插入符号函式

主要有五个插入符号函式：

- CreateCaret 建立与视窗有关的插入符号
- SetCaretPos 在视窗中设定插入符号的位置
- ShowCaret 显示插入符号
- HideCaret 隐藏插入符号
- DestroyCaret 撤消插入符号

另外还有取得插入符号目前位置 (GetCaretPos) 和取得以及设定插入符号闪烁时间 (GetCaretBlinkTime 和 SetCaretBlinkTime) 的函式。

在 Windows 中，插入符号定义为水平线、与字元大小相同的方框，或者与字元同高的竖线。如果使用调和字体，例如 Windows 内定的系统字体，则推荐使用竖线插入符号。因为调和字体中的字元没有固定大小，水平线或方框不能设定为字元的大小。

如果程式中需要插入符号，那么您不应该简单地在视窗讯息处理程式的 WM_CREATE 讯息处理期间建立它，然後在 WM_DESTROY 讯息处理期间撤消。其原因显而易见：一个讯息伫列只能支援一个插入符号。因此，如果您的程式有多个视窗，那么各个视窗必须有效地共用相同的插入符号。

其实，它并不像听起来那么多限制。您再想想就会发现，只有在视窗有输

入焦点时，视窗内显示插入符号才有意义。事实上，闪烁的插入符号只是一种视觉提示：您可以在程式中输入文字。因为任何时候都只有一个视窗拥有输入焦点，所以多个视窗同时都有闪烁的插入符号是没有意义的。

通过处理 WM_SETFOCUS 和 WM_KILLFOCUS 讯息，程式就可以确定它是否有输入焦点。正如名称所暗示的，视窗讯息处理程式在有输入焦点的时候接收到 WM_SETFOCUS 讯息，失去输入焦点的时候接收到 WM_KILLFOCUS 讯息。这些讯息成对出现：视窗讯息处理程式在接收到 WM_KILLFOCUS 讯息之前将一直接收到 WM_SETFOCUS 讯息，并且在视窗打开期间，此视窗总是接收到相同数量的 WM_SETFOCUS 和 WM_KILLFOCUS 讯息。

使用插入符号的主要规则很简单：视窗讯息处理程式在 WM_SETFOCUS 讯息处理期间呼叫 CreateCaret，在 WM_KILLFOCUS 讯息处理期间呼叫 DestroyCaret。

这里还有几条其他规则：插入符号刚建立时是隐蔽的。如果想使插入符号可见，那么您在呼叫 CreateCaret 之後，视窗讯息处理程式还必须呼叫 ShowCaret。另外，当视窗讯息处理程式处理一条非 WM_PAINT 讯息而且希望在视窗内绘制某些东西时，它必须呼叫 HideCaret 隐藏插入符号。在绘制完毕後，再呼叫 ShowCaret 显示插入符号。HideCaret 的影响具有累积效果，如果多次呼叫 HideCaret 而不呼叫 ShowCaret，那么只有呼叫 ShowCaret 相同次数时，才能看到插入符号。

TYPYR 程式

程式 6-5 所示的 TYPYR 程式使用了本章讨论的所有内容，您可以认为 TYPYR 是一个相当简单的文字编辑器。在视窗中，您可以输入字元，用游标移动键（也可以称为插入符号移动键）来移动游标（I 型标），按下 Escape 键清除视窗的内容等。缩放视窗、改变键盘输入语言时都会清除视窗的内容。本程式没有滚动，没有文字寻找和定位功能，不能储存档案，没有拼写检查，但它确实是写作一个文字编辑器的开始。

程式 6-5 TYPYR

```
TYPYR.C
/*-----
TYPYR.C --      Typing Program
                  (c) Charles Petzold, 1998
-----*/

#include <windows.h>

#define BUFFER(x,y) *(pBuffer + y * cxBuffer + x)
```

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("Typer") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS            wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }
    hwnd = CreateWindow (  szAppName, TEXT ("Typing Program"),
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static DWORD        dwCharSet = DEFAULT_CHARSET ;
    static int          cxChar, cyChar, cxClient, cyClient, cxBuffer, cyBuffer,
```

```
        xCaret, yCaret ;

static TCHAR *   pBuffer = NULL ;
HDC              hdc ;
int              x, y, i ;
PAINTSTRUCT      ps ;
TEXTMETRIC       tm ;

switch (message)
{
case WM_INPUTLANGCHANGE:
    dwCharSet = wParam ;
    // fall through
case WM_CREATE:
    hdc = GetDC (hwnd) ;
    SelectObject (   hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
                                     dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;

    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cyChar = tm.tmHeight ;

    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    ReleaseDC (hwnd, hdc) ;
    // fall through
case WM_SIZE:
    // obtain window size in pixels

    if (message == WM_SIZE)
    {
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
    }

    // calculate window size in characters

    cxBuffer = max (1, cxClient / cxChar) ;
    cyBuffer = max (1, cyClient / cyChar) ;

    // allocate memory for buffer and clear it

    if (pBuffer != NULL)
        free (pBuffer) ;

    pBuffer = (TCHAR *) malloc (cxBuffer * cyBuffer * sizeof (TCHAR)) ;

    for (y = 0 ; y < cyBuffer ; y++)
        for (x = 0 ; x < cxBuffer ; x++)
            BUFFER(x,y) = ' ' ;
```

```
        // set caret to upper left corner

xCaret = 0 ;
yCaret = 0 ;

if (hwnd == GetFocus ())
    SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;

InvalidateRect (hwnd, NULL, TRUE) ;
return 0 ;

case WM_SETFOCUS:
    // create and show the caret
    CreateCaret (hwnd, NULL, cxChar, cyChar) ;
    SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
    ShowCaret (hwnd) ;
    return 0 ;

case WM_KILLFOCUS:
    // hide and destroy the caret
    HideCaret (hwnd) ;
    DestroyCaret () ;
    return 0 ;

case WM_KEYDOWN:
    switch (wParam)
    {
    case VK_HOME:
        xCaret = 0 ;
        break ;

    case VK_END:
        xCaret = cxBuffer - 1 ;
        break ;

    case VK_PRIOR:
        yCaret = 0 ;
        break ;

    case VK_NEXT:
        yCaret = cyBuffer - 1 ;
        break ;

    case VK_LEFT:
        xCaret = max (xCaret - 1, 0) ;
        break ;

    case VK_RIGHT:
```

```

        xCaret = min (xCaret + 1, cxBuffer - 1) ;
        break ;

    case VK_UP:
        yCaret = max (yCaret - 1, 0) ;
        break ;

    case VK_DOWN:
        yCaret = min (yCaret + 1, cyBuffer - 1) ;
        break ;

    case VK_DELETE:
        for (x = xCaret ; x < cxBuffer - 1 ; x++)
            BUFFER (x, yCaret) = BUFFER (x + 1, yCaret) ;

        BUFFER (cxBuffer - 1, yCaret) = ' ' ;

        HideCaret (hwnd) ;
        hdc = GetDC (hwnd) ;

        SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
                                         dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
        TextOut (hdc, xCaret * cxChar, yCaret * cyChar,
                 & BUFFER (xCaret, yCaret),
                 cxBuffer - xCaret) ;

        DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;

        ReleaseDC (hwnd, hdc) ;
        ShowCaret (hwnd) ;
        break ;
    }
    SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
    return 0 ;

case WM_CHAR:
    for (i = 0 ; i < (int) LOWORD (lParam) ; i++)
    {
        switch (wParam)
        {
            case '\\b': // backspace
                if (xCaret > 0)
                {
                    xCaret-- ;
                    SendMessage (hwnd, WM_KEYDOWN, VK_DELETE,
1) ;
                }
                break ;
        }
    }

```



```
case '\\t':                                // tab
    do
    {
        SendMessage (hwnd, WM_CHAR, ' ', 1) ;
    }
    while (xCaret % 8 != 0) ;
    break ;

case '\\n':                                // line feed
    if (++yCaret == cyBuffer)
        yCaret = 0 ;
    break ;

case '\\r':                                // carriage return
    xCaret = 0 ;

    if (++yCaret == cyBuffer)
        yCaret = 0 ;
    break ;

case '\\x1B':                              // escape
    for (y = 0 ; y < cyBuffer ; y++)
        for (x = 0 ; x < cxBuffer ; x++)
            BUFFER (x, y) = ' ' ;

    xCaret = 0 ;
    yCaret = 0 ;

    InvalidateRect (hwnd, NULL, FALSE) ;
    break ;

default:                                    // character codes
    BUFFER (xCaret, yCaret) = (TCHAR) wParam ;

    HideCaret (hwnd) ;
    hdc = GetDC (hwnd) ;

    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
                                   dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
    TextOut (hdc, xCaret * cxChar, yCaret * cyChar,
             & BUFFER (xCaret, yCaret), 1) ;
    DeleteObject (
        SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    ReleaseDC (hwnd, hdc) ;
    ShowCaret (hwnd) ;

    if (++xCaret == cxBuffer)
    {
```

```

        xCaret = 0 ;
        if (++yCaret == cyBuffer)
            yCaret = 0 ;
    }
    break ;
}

SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
                                   dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
    for (y = 0 ; y < cyBuffer ; y++)
        TextOut (hdc, 0, y * cyChar, & BUFFER(0,y), cxBuffer) ;
    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

为了简单起见, TYPED 程式使用一种等宽字体, 因为编写处理调和字体的文字编辑器要困难得多。程式在好几个地方取得装置内容: 在 WM_CREATE 讯息处理期间, 在 WM_KEYDOWN 讯息处理期间, 在 WM_CHAR 讯息处理期间以及在 WM_PAINT 讯息处理期间, 每次都通过 GetStockObject 和 SelectObject 呼叫来选择等宽字体。

在 WM_SIZE 讯息处理期间, TYPED 计算视窗的字元宽度和高度并把值保存在 cxBuffer 和 cyBuffer 变数中, 然後使用 malloc 分配缓冲区以保存在视窗内输入的所有字元。注意, 缓冲区的位元组大小取决於 cxBuffer、cyBuffer 和 sizeof (TCHAR), 它可以是 1 或 2, 这依赖於程式是以 8 位元的字元处理还是以 Unicode 方式编译的。

xCaret 和 yCaret 变数保存插入符号位置。在 WM_SETFOCUS 讯息处理期间, TYPED 呼叫 CreateCaret 来建立与字元有相同宽度和高度的插入符号, 呼叫 SetCaretPos 来设定插入符号的位置, 呼叫 ShowCaret 使插入符号可见。在 WM_KILLFOCUS 讯息处理期间, TYPED 呼叫 HideCaret 和 DestroyCaret。

对 WM_KEYDOWN 的处理大多要涉及游标移动键。Home 和 End 把插入符号送至一行的开始和末尾处, Page Up 和 Page Down 把插入符号送至视窗的顶端和底部, 箭头的用法不变。对 Delete 键, TYPED 将缓冲区中从插入符号之後的那个位置开始到行尾的所有内容向前移动, 并在行尾显示空格。

WM_CHAR 处理 Backspace、Tab、Linefeed (Ctrl-Enter)、Enter、Escape 和字元键。注意, 在处理 WM_CHAR 讯息时 (假设使用者输入的每个字元都非常重要), 我使用了 lParam 中的重复计数; 而在处理 WM_KEYDOWN 讯息时却不这么作 (避免有害的重复卷动)。对 Backspace 和 Tab 的处理由於使用了 SendMessage 函式而得到简化, Backspace 与 Delete 做法相仿, 而 Tab 则如同输入了若干个空格。

前面我已经提到过, 在非 WM_PAINT 讯息处理期间, 如果要在视窗中绘制内容, 则应该隐蔽游标。TYPED 为 Delete 键处理 WM_KEYDOWN 讯息和为字元键处理 WM_CHAR 讯息时即是如此。在这两种情况下, TYPED 改变缓冲区中的内容, 然後在视窗中绘制一个或者多个新字元。

虽然 TYPED 使用了与 KEYVIEW2 相同的做法以在字元集之间切换 (就像使用者切换键盘布局一样), 但對於远东版的 Windows, 它还是不能正常工作。TYPED 不允许使用两倍宽度的字元。此问题将在第十七章讨论, 那时我们将详细讨论字体与文字输出。

第七章 滑鼠

滑鼠是有一个或多个键的定位设备。虽然也可以使用诸如触摸画面和光笔之类的输入设备，但是只有滑鼠以及常用在膝上型电脑上的轨迹球等才是渗透了 PC 市场的唯一输入设备。

情况并非总是如此。当然，Windows 的早期开发人员认为他们不应该要求使用者为了执行其产品而必须买只滑鼠。因此，他们将滑鼠作为一种选择性的附加设备，而为 Windows 中的所有操作以及 applet 提供一种键盘介面（例如，查看 Windows 小算盘程式的线上说明资讯，可以看到每个按钮都提供了一个同等功效的键盘操作方式）。第三方软体发展人员使用键盘介面来提供与滑鼠操作相同的功能，这本书以前的版本也是这么做的。

理论上来说，现在的 Windows 需要滑鼠。至少，一些讯息方块是这样讲的。当然，您也可以拔下滑鼠，而且 Windows 仍然可以执行良好（只有讯息方块会提示您没有连接滑鼠）。试图不用滑鼠来使用 Windows 就像用脚趾来弹钢琴一样（至少在最初的一段时间里是这样），但您依然可以这样做。正因为如此，我还是喜欢为滑鼠功能提供键盘操作。打字员尤其喜欢让他们的手保持在键盘上，并且我认为每个人都有在杂乱的桌上找不到滑鼠，或者滑鼠移动不灵敏的经验。使用键盘通常不需要花费更多的精力和努力，并且为喜欢使用键盘的人提供更多的功能。

我们通常认为，键盘便於输入和操作文字资料，而滑鼠则便於画图和操作图形物件。实际上，本章大多数的范例程式都画了一些图形，并且用到了我们在第五章所学到的知识。

滑鼠基础

Windows 98 能支援单键、双键或者三键滑鼠，也可以使用摇杆或者光笔来模拟单键滑鼠。早期，由於许多使用者都有单键滑鼠，所以 Windows 应用程式总是避免使用双键或三键滑鼠。不过，由於双键滑鼠已经成为事实上的标准，因此不使用第二个键的传统已经不再合理了。当然，第二个滑鼠按键是用於启动一个「快显功能表」，亦即出现在普通功能表列之外的视窗中功能表，或者用於特殊的拖曳操作（拖曳将在後面加以解释）。然而，程式不能依赖双键滑鼠。

理论上，您可以用我们的老朋友 `GetSystemMetrics` 函式来确认滑鼠是否存在：

```
fMouse = GetSystemMetrics (SM_MOUSEPRESENT) ;
```

如果已经安装了滑鼠，fMouse 将传回 TRUE（非 0）；如果没有安装，则传回 0。然而，在 Windows 98 中，不论滑鼠是否安装，此函式都将传回 TRUE。在 Microsoft Windows NT 中，它可以正常工作。

要确定所安装滑鼠其上按键的个数，可使用

```
cButtons = GetSystemMetrics (SM_CMOUSEBUTTONS) ;
```

如果没有安装滑鼠，那么函式将传回 0。然而，在 Windows 98 下，如果没有安装滑鼠，此函式将传回 2。

习惯用左手的使用者可以使用 Windows 的「控制台」来切换滑鼠按键。虽然应用程式可以通过在 GetSystemMetrics 中使用 SM_SWAPBUTTON 参数来确定是否进行了这种切换，但通常没有这个必要。由食指触发的键被认为是左键，即使事实上是位於滑鼠的右边。不过，在一个教育训练程式中，您可能想在萤幕上画一个滑鼠，在这种情况下，您可能想知道滑鼠按键是否被切换过了。

您可以在「控制台」中设定滑鼠的其他参数，例如双击速度。从 Windows 应用程式，通过使用 SystemParametersInfo 函式可以设定或获得此项资讯。

一些简单的定义

当 Windows 使用者移动滑鼠时，Windows 在显示器上移动一个称为「滑鼠游标」的小点阵图。滑鼠游标有一个指向显示器上精确位置的单图素「热点」。当我提到滑鼠游标在萤幕上的位置时，指的是热点的位置。

Windows 支援几种预先定义的滑鼠游标，程式可以使用这些游标。最常见的是称为 IDC_ARROW 的斜箭头（在 WINUSER.H 中定义）。热点在箭头的顶端。IDC_CROSS 游标（在本章後面的 BLOKOUT 程式中有用到）的热点在十字交叉线的中心。IDC_WAIT 游标是一个沙漏，通常用於指示程式正在执行。程式写作者也可以设计自己的游标。我们将在第十章学习设计方法。在定义视窗类别结构时指定特定视窗的内定游标，例如：

```
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
```

下面是一些描述滑鼠按键动作的术语：

- **Clicking** 按下并放开一个滑鼠按键。
- **Double-clicking** 快速按下并放开滑鼠按键两次。
- **Dragging** 按住滑鼠按键并移动滑鼠。

对三键滑鼠来说，三个键分别称为左键、中键、右键。在 Windows 表头档案中定义的与滑鼠有关的识别字使用缩写 LBUTTON、MBUTTON 和 RBUTTON。双键滑鼠只有左键与右键，单键滑鼠只有一个左键。

滑鼠(Mouse)的复数

现在，为了展现我的勇气，我将面对输入装置最难辩的争论话题：什么是「mouse」的复数。虽然每个人都知道多只啮齿动物称为 mice，似乎没有人对该如何称呼多个输入装置有最後的答案。不管「mice」或「mouse」听起来都不对劲。我惯常参考的《American Heritage Dictionary of the English Language》第三版则只字未提。

《Wired style: Principles of English Usage in the Digital Age》(HardWired, 1996) 指出「mouse」比较好，以避免与啮齿动物搞混。在 1964 发明滑鼠的 Doug Engelbart 对此争议也帮不上忙。我曾经问过他 mouse 的复数是什么，他说我不知道。

最後，高权威的 Microsoft Manual of Style for Technical Publications 告诉我们「避免使用复数 mice。假如你必须提到多只 mouse，使用 mouse devices」。这听起来像是在逃避问题，但当一切听起来都不对劲时，它确实是个明智的忠告了。事实上，大部分需要 mouse 复数的句子都能重新修改来避开。例如，试著说“People use the almost as much as keyboard”，而不是“Pople use mice almost as much as keyboards”。

显示区域滑鼠讯息

在前一章中您已经看到，Windows 只把键盘讯息发送给拥有输入焦点的视窗。滑鼠讯息与此不同：只要滑鼠跨越视窗或者在某视窗中按下滑鼠按键，那么视窗讯息处理程式就会收到滑鼠讯息，而不管该视窗是否活动或者是否拥有输入焦点。Windows 为滑鼠定义了 21 种讯息，不过，其中有 11 个讯息和显示区域无关（下面称之为「非显示区域」讯息），Windows 程式经常忽略这些讯息。

当滑鼠移过视窗的显示区域时，视窗讯息处理程式收到 WM_MOUSEMOVE 讯息。当在视窗的显示区域中按下或者释放一个滑鼠按键时，视窗讯息处理程式会接收到下面这些讯息：

表 7-1

键	按下	释放	按下(双键)
左	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDOWNBLCLK
中	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDOWNBLCLK
右	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDOWNBLCLK

只有对三键滑鼠，视窗讯息处理程式才会收到 MBUTTON 讯息；只有对双键或者三键滑鼠，才会接收到 RBUTTON 讯息。只有当定义的视窗类别能接收 DBLCLK

(双击) 讯息, 视窗讯息处理程式才能接收到这些讯息 (请参见本章中「双击滑鼠按键」一节)。

对于所有这些讯息来说, 其 lParam 值均含有滑鼠的位置: 低字组为 x 座标, 高字组为 y 座标, 这两个座标是相对于视窗显示区域左上角的位置。您可以用 LOWORD 和 HIWORD 巨集来提取这些值:

```
x = LOWORD (lParam) ;
y = HIWORD (lParam) ;
```

wParam 的值指示滑鼠按键以及 Shift 和 Ctrl 键的状态。您可以使用表头档案 WINUSER.H 中定义的位元遮罩来测试 wParam。MK 字首代表「滑鼠按键」。

MK_LBUTTON	按下左键
MK_MBUTTON	按下中键
MK_RBUTTON	按下右键
MK_SHIFT	按下 Shift 键
MK_CONTROL	按下 Ctrl 键

例如, 如果收到了 WM_LBUTTONDOWN 讯息, 而且值

```
wparam & MK_SHIFT
```

是 TRUE (非 0), 您就知道当左键按下时也按下了 Shift 键。

当您把滑鼠移过视窗的显示区域时, Windows 并不为滑鼠的每个可能的图素位置都产生一个 WM_MOUSEMOVE 讯息。您的程式接收到 WM_MOUSEMOVE 讯息的次数, 依赖于滑鼠硬體, 以及您的视窗讯息处理程式在处理滑鼠移动讯息时的速度。换句话说, Windows 不能用未处理的 WM_MOUSEMOVE 讯息来填入讯息伫列。当您执行下面将描述的 CONNECT 程式时, 您将会更了解 WM_MOUSEMOVE 讯息处理的速率。

如果您在非活动视窗的显示区域中按下滑鼠左键, Windows 将把活动视窗改为在其中按下滑鼠按键的视窗, 然后把 WM_LBUTTONDOWN 讯息送到该视窗讯息处理程式。当视窗讯息处理程式得到 WM_LBUTTONDOWN 讯息时, 您的程式就可以安全地假定该视窗是活动化的了。不过, 您的视窗讯息处理程式可能在未接收到 WM_LBUTTONDOWN 讯息的情况下先接收到了 WM_LBUTTONUP 的讯息。如果在一个视窗中按下滑鼠按键, 然后移动到使用者视窗释放它, 就会出现这种情况。类似的情况, 当滑鼠按键在另一个视窗中被释放时, 视窗讯息处理程式只能接收到 WM_LBUTTONDOWN 讯息, 而没有相应的 WM_LBUTTONUP 讯息。

这些规则有两个例外:

视窗讯息处理程式可以「拦截滑鼠」并且连续地接收滑鼠讯息, 即使此时滑鼠在该视窗显示区域之外。您将在本章的后面学习如何拦截滑鼠。

如果正在显示一个系统模态讯息方块或者系统模态对话方块, 那么其他程

式就不能接收滑鼠讯息。当系统模态讯息方块或者对话方块活动时，禁止切换到其他视窗或者程式。一个显示系统模态讯息方块的例子，是当您关闭 Windows 时。

简单的滑鼠处理：一个例子

程式 7-1 中所示的 CONNECT 程式能作一些简单的滑鼠处理，使您对 Windows 如何向您的程式发送滑鼠讯息有一些体会。

程式 7-1 CONNECT

```
CONNECT.C
/*-----
    CONNECT.C -- Connect-the-Dots Mouse Demo Program
                (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
#define MAXPOINTS 1000
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("Connect") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Connect-the-Points Mouse Demo"),
```



```

WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT,
NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}

return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static POINT pt[MAXPOINTS] ;
    static int      iCount ;
    HDC             hdc ;
    int             i, j ;
    PAINTSTRUCT ps ;
    switch (message)
    {
    case WM_LBUTTONDOWN:
        iCount = 0 ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case WM_MOUSEMOVE:
        if (wParam & MK_LBUTTON && iCount < 1000)
        {
            pt[iCount ].x = LOWORD (lParam) ;
            pt[iCount++].y = HIWORD (lParam) ;

            hdc = GetDC (hwnd) ;
            SetPixel (hdc, LOWORD (lParam), HIWORD (lParam), 0) ;
            ReleaseDC (hwnd, hdc) ;
        }

        return 0 ;

    case WM_LBUTTONUP:
        InvalidateRect (hwnd, NULL, FALSE) ;
        return 0 ;
    }
}

```

```
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    for (i = 0 ; i < iCount - 1 ; i++)
        for (j = i + 1 ; j < iCount ; j++)
        {
            MoveToEx (hdc, pt[i].x, pt[i].y, NULL) ;
            LineTo   (hdc, pt[j].x, pt[j].y) ;
        }

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CONNECT 处理三个滑鼠讯息:

- **WM_LBUTTONDOWN** CONNECT 清除显示区域。
- **WM_MOUSEMOVE** 如果按下左键, 那么 CONNECT 就在显示区域中的滑鼠位置处绘制一个黑点, 并保存该座标。
- **WM_LBUTTONUP** CONNECT 把显示区域中绘制的点与其他每个点连接起来。有时会产生一个漂亮的图形, 有时则会是黑鸦鸦的一团糟 (见图 7-1)。

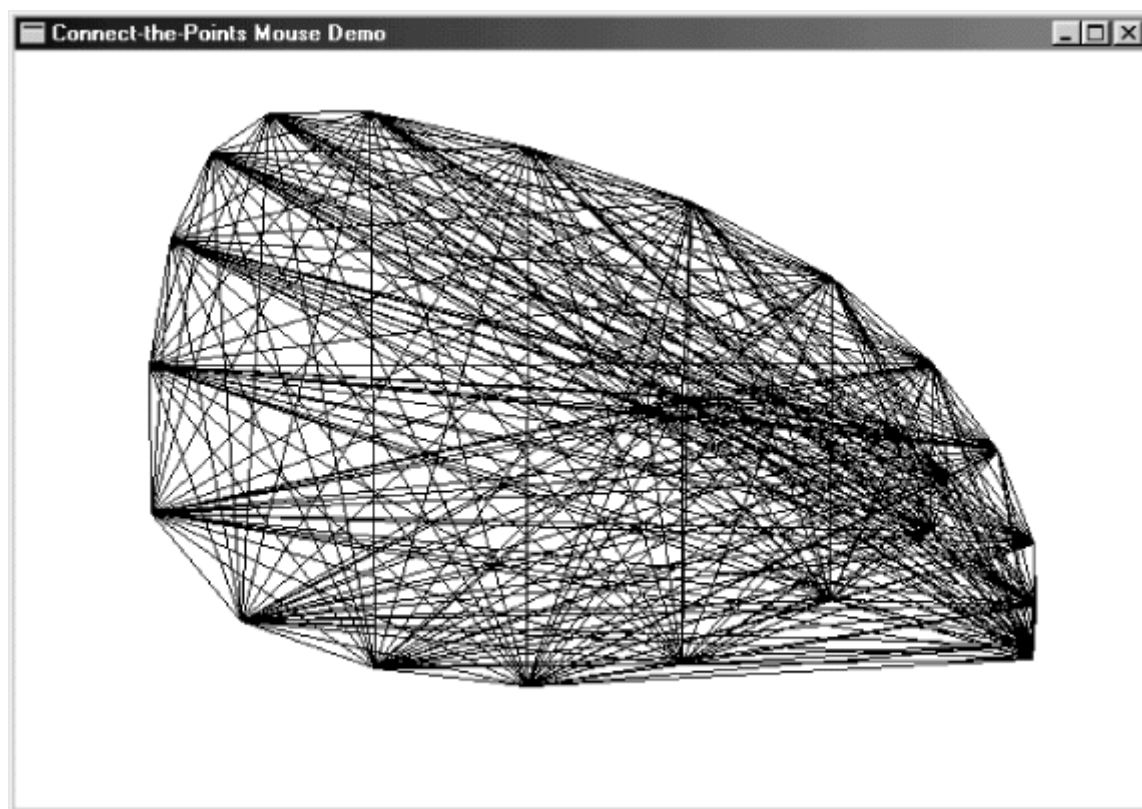


图 7-1 CONNECT 的萤幕显示

CONNECT 的使用方法：把滑鼠游标移动到显示区域中，按下左键，移动一下位置，释放左键。对几个构成曲线的点，CONNECT 能处理得很好，方法是按住左键，快速移动滑鼠，这样就可以绘制出该曲线图案。

CONNECT 使用了三个简单的图形装置介面 (GDI) 函式，我在第五章讨论过这些函式。当滑鼠左键按下时，SetPixel 为每个 WM_MOUSEMOVE 讯息绘制一个黑图素（对于高解析度的显示器，图素几乎看不见）。画直线需要 MoveToEx 和 LineTo 函式。

如果您在释放滑鼠按键之前把滑鼠游标移到显示区域之外，那么 CONNECT 就不会连接这些点，因为它没有收到 WM_LBUTTONDOWN 讯息。如果您把滑鼠移回显示区域内并按下左键，那么 CONNECT 将清除显示区域。如果想在显示区域外释放左键后还继续进行画图，那么可以在显示区域外按下滑鼠再移回显示区域中。

CONNECT 最多可以保存 1000 个点。设点数为 P ，则 CONNECT 画的线数就等于 $P \times (P - 1) / 2$ 。如果有 1000 个点，则要绘制 50 万条直线，大约需要几分钟才能画完（时间的长短取决于您的硬体设备）。由于 Windows 98 是一种优先权式多工环境，因此您可以在这一段时间切换到别的程式中。但是，当程式正在忙的时候，您将无法对 CONNECT 程式做任何事（诸如移动或者缩放等）。在第二十章中，我们将讨论解决这一问题的方法。

因为 CONNECT 可能会花一些时间来绘制直线，因此在处理 WM_PAINT 讯息时它将切换到沙漏游标，然后再恢复原状。这要求使用两个现有游标来呼叫

SetCursor.CONNECT 还呼叫两次 ShowCursor, 一次用 TRUE 参数, 另一次用 FALSE 参数。我将在本章的後面, 「使用键盘模拟滑鼠」一节中更详细地讨论这些呼叫。

有时, 我们使用「跟踪」这个词代表程式处理滑鼠移动的方法。但是, 跟踪并不意味著, 程式在视窗讯息处理程式中的某个回圈里, 不断跟随滑鼠在显示器上的运动。实际上, 视窗讯息处理程式处理每条滑鼠讯息, 然後迅速退出。

处理 Shift 键

当 CONNECT 接收到一个 WM_MOUSEMOVE 讯息时, 它把 wParam 和 MK_LBUTTON 进行位元与 (AND) 运算, 来确定是否按下了左键。wParam 也可以用於确定 Shift 键的状态。例如, 如果处理必须依赖於 Shift 和 Ctrl 键的状态, 那么您可以使用如下所示的方法:

```
if (wParam & MK_SHIFT)
{
    if (wParam & MK_CONTROL)
    {
        //按下了 Shift 和 Ctrl 键
    }
    else
    {
        //按下了 Shift 键
    }
}
else
{
    if (wParam & MK_CONTROL)
    {
        //按下了 Ctrl 键
    }
    else
    {
        //Shift 和 Ctrl 键均未按下
    }
}
```

如果您想在程式中同时使用左右键, 同时如果您还希望只有单键滑鼠的使用者也能使用您的程式, 那么您可以这样来写作程式: Shift 与左键的组合使用等效於右键。在这种情况下, 对滑鼠按键的处理可以采用如下所示的方法:

```
case WM_LBUTTONDOWN:
    if (!(wParam & MK_SHIFT))
    {
        //处理左键
        return 0 ;
    }
```

```

    }
                                // Fall through
case WM_RBUTTONDOWN:
    //处理右键
    return 0 ;

```

Windows 函式 `GetKeyState`（在第六章中介绍过）可以使用虚拟键码 `VK_LBUTTON`、`VK_RBUTTON`、`VK_MBUTTON`、`VK_SHIFT` 和 `VK_CONTROL` 来传回鼠标按键与 Shift 键的状态。如果 `GetKeyState` 传回负值，则说明已按下了鼠标按键或者 Shift 键。因为 `GetKeyState` 传回目前正在处理的鼠标按键或者 Shift 键的状态，所以全部状态资讯与相应的讯息都是同步的。但是，正如不能把 `GetKeyState` 用於尚未按下的键一样，您也不能为尚未按下的鼠标按键呼叫 `GetKeyState`。请不要这样做：

```
while (GetKeyState (VK_LBUTTON) >= 0) ; // WRONG !!!
```

只有在您呼叫 `GetKeyState` 期间处理讯息时，而左键已经按下，才会报告键已经按下的讯息。

双击鼠标按键

双击鼠标按键是指在短时间内单击两次。要确定为双击，则这两次单击必须发生在其相距的实际位置十分接近的状况下（内定范围是一个平均系统字体字元的宽，半个字元的高），并且发生在指定的时间间隔（称为「双击速度」）内。您可以在「控制台」中改变时间间隔。

如果希望您的视窗讯息处理程式能够收到双按键的鼠标讯息，那么在呼叫 `RegisterClass` 初始化视窗类别结构时，必须在视窗风格中包含 `CS_DBLCLKS` 识别字：

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS ;
```

如果在视窗风格中未包含 `CS_DBLCLKS`，而使用者在短时间内双击了鼠标按键，那么视窗讯息处理程式会接收到下面这些讯息：

- `WM_LBUTTONDOWN`
- `WM_LBUTTONUP`
- `WM_LBUTTONDOWN`
- `WM_LBUTTONUP`

视窗讯息处理程式可能在这些键的讯息之前还收到了其他讯息。如果您想实作自己的双击处理，那么您可以使用 Windows 函式 `GetMessageTime` 取得 `WM_LBUTTONDOWN` 讯息之间的相对时间。第八章将更详细地讨论这个函式。

如果您的视窗类别风格中包含了 `CS_DBLCLKS`，那么双击时视窗讯息处理程式将收到如下讯息：

- WM_LBUTTONDOWN
- WM_LBUTTONUP
- WM_LBUTTONDBLCLK
- WM_LBUTTONUP

WM_LBUTTONDBLCLK 讯息简单地替换了第二个 WM_LBUTTONDOWN 讯息。

如果双击中的第一次键操作完成单击的功能，那么双击这一讯息是很容易处理的。第二次按键 (WM_LBUTTONDBLCLK 讯息) 则完成第一次按键以外的事情。例如，看看 Windows Explorer 中是如何用滑鼠来操作档案列表的。按一次键将选中档案，Windows Explorer 用反白显示列指出被选择档案的位置。双击则实作两个功能：第一次是单击那个选中档案；第二次则指向 Windows Explorer 以打开该档案。执行方式相当简单，如果双击中的第一次按键不执行单击功能，那么滑鼠处理方式会变得非常复杂。

非显示区域滑鼠讯息

在视窗的显示区域内移动或按下滑鼠按键时，将产生 10 种讯息。如果滑鼠在视窗的显示区域之外但还在视窗内，Windows 就给视窗讯息处理程式发送一条「非显示区域」滑鼠讯息。视窗非显示区域包括标题列、功能表和视窗卷动列。

通常，您不需要处理非显示区域滑鼠讯息，而是将这些讯息传给 DefWindowProc，从而使 Windows 执行系统功能。就这方面来说，非显示区域滑鼠讯息类似於系统键盘讯息 WM_SYSKEYDOWN、WM_SYSKEYUP 和 WM_SYSCHAR。

非显示区域滑鼠讯息几乎完全与显示区域滑鼠讯息相对应。讯息中含有字母「NC」以表示是非显示区域讯息。如果滑鼠在视窗的非显示区域中移动，那么视窗讯息处理程式会接收到 WM_NCMOUSEMOVE 讯息。滑鼠按键产生如表 7-2 所示的讯息。

表 7-2

键	按下	释放	按下 (双击)
左	WM_NCLBUTTONDOWN	WM_NCLBUTTONUP	WM_NCLBUTTONDBLCLK
中	WM_NCMBBUTTONDOWN	WM_NCMBUTTONUP	WM_NCMBUTTONDBLCLK
右	WM_NCRBUTTONDOWN	WM_NCRBUTTONUP	WM_NCRBUTTONDBLCLK

对非显示区域滑鼠讯息，wParam 和 lParam 参数与显示区域滑鼠讯息的 wParam 和 lParam 参数不同。wParam 参数指明移动或者按滑鼠按键的非显示区域。它设定为 WINUSER.H 中定义的以 HT 开头的识别字之一 (HT 表示「命中测试」)。

lParam 参数的低位元 word 为 x 座标，高位元 word 为 y 座标，但是，它们是萤幕座标，而不是像显示区域滑鼠讯息那样指的是显示区域座标。对萤幕座

标，显示器左上角的 x 和 y 的值为 0。当往右移时 x 的值增加，往下移时 y 的值增加（见图 7-2）。

您可以用两个 Windows 函式将萤幕坐标转换为显示区域坐标或者反之：

```
ScreenToClient (hwnd, &pt) ;
ClientToScreen (hwnd, &pt) ;
```

这里 pt 是 POINT 结构。这两个函式转换了保存在结构中的值，而且没有保留以前的值。注意，如果萤幕坐标点在视窗显示区域的上面或者左边，显示区域坐标 x 或 y 值就是负值。



图 7-2 萤幕坐标与客户显示区域坐标

命中测试讯息

如果您数一下，就可以知道我们已经介绍了 21 个滑鼠讯息中的 20 个，最後一个讯息是 WM_NCHITTEST，它代表「非显示区域命中测试」。此讯息优先於所有其他的显示区域和非显示区域滑鼠讯息。lParam 参数含有滑鼠位置的 x 和 y 萤幕坐标，wParam 参数没有用。

Windows 應用程式通常把这个讯息传送给 DefWindowProc，然後 Windows 用

WM_NCHITTEST 讯息产生与滑鼠位置相关的所有其他滑鼠讯息。对于非显示区域滑鼠讯息，在处理 WM_NCHITTEST 时，从 DefWindowProc 传回的值将成为滑鼠讯息中的 wParam 参数，这个值可以是任意非显示区域滑鼠讯息的 wParam 值再加上以下内容：

HTCLIENT	显示区域
HTNOWHERE	不在视窗中
HTTRANSPARENT	视窗由另一个视窗覆盖
HTERROR	使 DefWindowProc 产生警示用的哔声

如果 DefWindowProc 在其处理 WM_NCHITTEST 讯息后传回 HTCLIENT，那么 Windows 将把萤幕坐标转换为显示区域坐标并产生显示区域滑鼠讯息。

如果您还记得我们如何通过拦截 WM_SYSKEYDOWN 讯息来停用所有的系统键盘功能，那么您可能会想我们可否通过拦截滑鼠讯息完成类似的事情。完全可以！只要您在视窗讯息处理程式中包含以下几条叙述：

```
case WM_NCHITTEST:
    return (LRESULT) HTNOWHERE ;
```

就可以有效地禁用您视窗中的所有显示区域和非显示区域滑鼠讯息。这样一来，当滑鼠在您的视窗（包括系统功能表图示、缩放按钮以及关闭按钮）中时，滑鼠按键将会失效。

从讯息产生讯息

Windows 用 WM_NCHITTEST 讯息产生所有其他滑鼠讯息，这种由讯息引出其他讯息的想法在 Windows 中是很普遍的。让我们来举个例子。您知道，如果您在一个 Windows 程式的系统功能表图示上双击一下，那么程式将会终止。双击产生一系列的 WM_NCHITTEST 讯息。由于滑鼠定位在系统功能表图示上，因此 DefWindowProc 将传回 HTSYSTEMMENU 的值，并且 Windows 把 wParam 等於 HTSYSTEMMENU 的 WM_NCLBUTTONDBLCLK 讯息放在讯息佇列中。

视窗讯息处理程式通常把滑鼠讯息传递给 DefWindowProc，当 DefWindowProc 接收到 wParam 参数等於 HTSYSTEMMENU 的 WM_NCLBUTTONDBLCLK 讯息时，它就把 wParam 参数等於 SC_CLOSE 的 WM_SYSCOMMAND 讯息放入讯息佇列中（这个 WM_SYSCOMMAND 讯息是在使用者从系统功能表中选择「Close」时产生的）。同样地，视窗讯息处理程式也把这个讯息传给 DefWindowProc。DefWindowProc 通过给视窗讯息处理程式发送 WM_CLOSE 讯息来处理该讯息。

如果一个程式在终止之前要求来自使用者的确认，那么视窗讯息处理程式就需要拦截 WM_CLOSE，否则，DefWindowProc 呼叫 DestroyWindow 函式来处理 WM_CLOSE。除了其他处理，DestroyWindow 还给视窗讯息处理程式发送一个

WM_DESTROY 讯息。视窗讯息处理程式通常用下列程式码来处理 WM_DESTROY 讯息：

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

PostQuitMessage 使得 Windows 把 WM_QUIT 讯息放入讯息佇列中，此讯息永远不会到达视窗讯息处理程式，因为它使 GetMessage 传回 0，并终止讯息回圈，从而也终止了程式。

程式中的命中测试

我在前面讨论了 Windows Explorer 如何回应滑鼠的单击和双击。显然，程式（或者更精确的说，如同 Windows Explorer 般使用 list view control）必须确定使用者滑鼠所指向的是哪一个档案。

这叫做「命中测试」。正如 DefWindowProc 在处理 WM_NCHITTEST 讯息时做一些命中测试一样，视窗讯息处理程式经常必须在显示区域中进行一些命中测试。一般来说，命中测试中会使用 x 和 y 座标值，它们由传到视窗讯息处理程式的滑鼠讯息的 lParam 参数给出。

一个假想的例子

有这样一个例子。假设您的程式需要显示几列按字母排列的档案。通常，您可以使用 list view control，他会帮您由於要做全部的命中测试工作。但我们假设您由於某种原因而不能使用，这时就需要自己来做了。让我们假定档案名保存在称为 szFileNames 的已排序字串指标阵列中。

让我们也假定档案列表开始於显示区域的顶端，显示区域为 cxClient 图素宽，cyClient 图素高，每列为 cxColWidth 图素宽，每个字元高度为 cyChar 图素高。那么每栏可填入的档案数就是：

```
iNumInCol = cyClient / cyChar ;
```

接收到一个滑鼠单击讯息後，您就能从 lParam 获得 cxMouse 和 cyMouse 座标。然後可以用下面的公式来计算使用者所指的是哪一系列的档案名：

```
iColumn = cxMouse / cxColWidth ;
```

相对於列顶端的档案名位置为：

```
iFromTop = cyMouse / cyChar ;
```

现在您就可以计算 szFileNames 阵列的下标：

```
iIndex = iColumn * iNumInCol + iFromTop ;
```

如果 iIndex 超过了阵列中的档案数，则表示使用者是在显示器的空白区域内按滑鼠按键。

在许多情况下，命中测试要比本例更加复杂。在显示一幅包含许多小图形

的图像时，您必须决定要显示的每个小图形的座标。在命中计算中，您必须从座标找到物件。但这将在使用不确定字体大小的字处理程式中变得非常凌乱，因为您必须找到字元在字串中的位置。

范例程式

程式 7-2 所示的 CHECKER1 程式展示了一些简单的命中测试，此程式把显示区域分为 5×5 的 25 个矩形。如果您在某个矩形中按下鼠标按键，那么在该矩形中将出现一个「X」。如果您再按一次，那么「X」将被删除。

程式 7-2 CHECKER1

```
CHECKER1.C
/*-----
CHECKER1.C --          Mouse Hit-Test Demo Program No. 1
                        (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#define DIVISIONS 5
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine,
int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Checker1") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style
                        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
                        = WndProc ;
    wndclass.cbClsExtra
                        = 0 ;
    wndclass.cbWndExtra
                        = 0 ;
    wndclass.hInstance
                        = hInstance ;
    wndclass.hIcon
                        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
                        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
                        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
                        = NULL ;
    wndclass.lpszClassName
                        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Checker1 Mouse Hit-Test Demo"),
                        WS_OVERLAPPEDWINDOW,
```

```

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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND      hwnd,      UINT      message,      WPARAM
wParam, LPARAM lParam)
{
    static BOOL      fState[DIVISIONS][DIVISIONS] ;
    static int      cxBlock, cyBlock ;
    HDC      hdc ;
    int      x, y ;
    PAINTSTRUCT      ps ;
    RECT      rect ;

    switch (message)
    {
    case WM_SIZE :
        cxBlock = LOWORD (lParam) / DIVISIONS ;
        cyBlock = HIWORD (lParam) / DIVISIONS ;
        return 0 ;

    case WM_LBUTTONDOWN :
        x = LOWORD (lParam) / cxBlock ;
        y = HIWORD (lParam) / cyBlock ;

        if (x < DIVISIONS && y < DIVISIONS)
        {
            fState [x][y] ^= 1 ;
            rect.left      = x * cxBlock ;
            rect.top       = y * cyBlock ;
            rect.right     = (x + 1) * cxBlock ;
            rect.bottom    = (y + 1) * cyBlock ;

            InvalidateRect (hwnd, &rect, FALSE) ;
        }
        else
            MessageBeep (0) ;
    }
}

```

```
        return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    for (x = 0 ; x < DIVISIONS ; x++)
    for (y = 0 ; y < DIVISIONS ; y++)
    {
        Rectangle (hdc, x * cxBlock, y * cyBlock,
                    (x + 1) * cxBlock, (y + 1) * cyBlock) ;

        if (fState [x][y])
        {
            MoveToEx (hdc, x * cxBlock, y * cyBlock, NULL) ;
            LineTo (hdc, (x+1) * cxBlock, (y+1) * cyBlock) ;
            MoveToEx (hdc, x * cxBlock, (y+1) * cyBlock, NULL) ;
            LineTo (hdc, (x+1) * cxBlock, y * cyBlock) ;
        }
    }
    EndPaint (hwnd, &ps);
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图 7-3 是 CHECKER1 的显示。程式画的 25 个矩形的宽度和高度均相同。这些宽度和高度保存在 cxBlock 和 cyBlock 中，当显示区域大小发生改变时，将重新对这些值进行计算。WM_LBUTTONDOWN 处理过程使用滑鼠坐标来确定在哪个矩形中按下了键，它在 fState 阵列中标志目前矩形的状态，并使该矩形区域失效，从而产生 WM_PAINT 讯息。

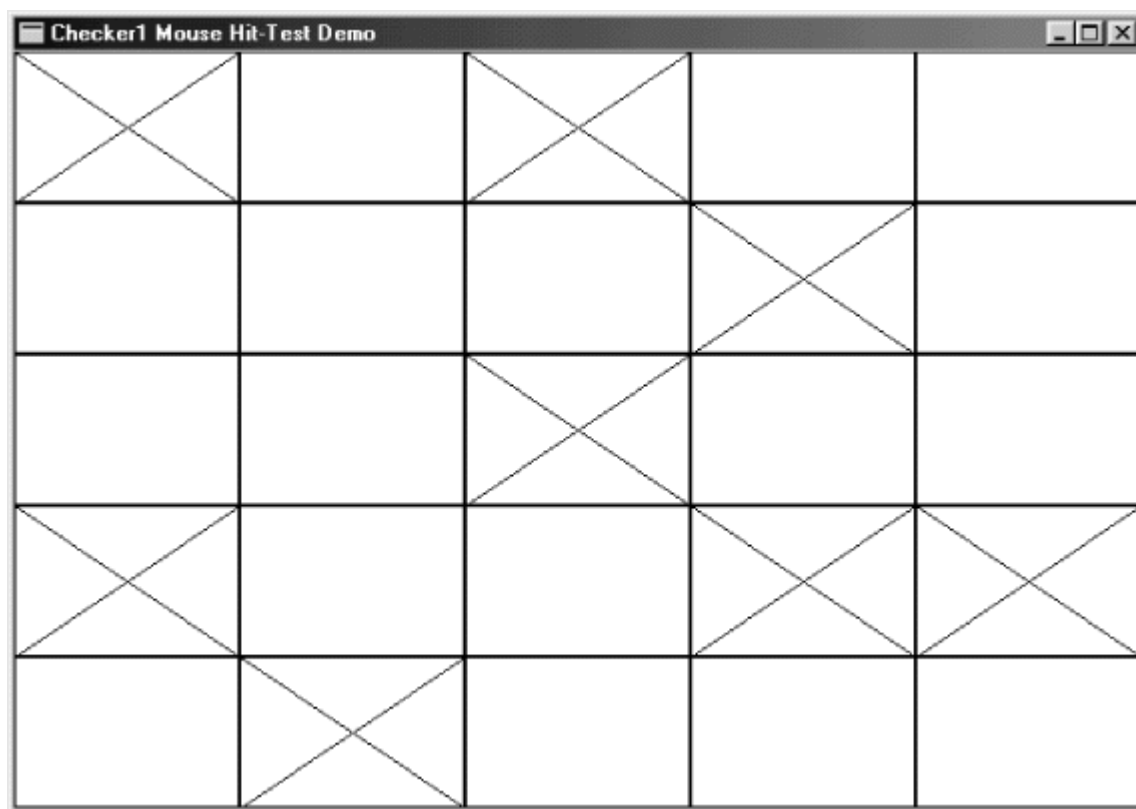


图 7-3 CHECKER1 的萤幕显示

如果显示区域的宽度和高度不能被 5 整除，那么在显示区域的左边和下边将有一小条区域不能被矩形所覆盖。对于错误情况，CHECKER1 通过呼叫 MessageBeep 回应此区域中的滑鼠按键操作。

当 CHECKER1 收到 WM_PAINT 讯息时，它通过 GDI 的 Rectangle 函式来重新绘制显示区域。如果设定了 fState 值，那么 CHECKER1 将使用 MoveToEx 和 LineTo 函式来绘制两条直线。在处理 WM_PAINT 期间，CHECKER1 在重新绘制之前并不检查每个矩形区域的有效性，尽管它可以这样做。检查有效性的一种方法是在回圈中为每个矩形块建立 RECT 结构（使用与 WM_LBUTTONDOWN 处理程式中相同的公式），并使用 IntersectRect 函式检查它是否与无效矩形 (ps.rcPaint) 相交。

使用键盘模拟滑鼠

CHECKER1 只能在装有滑鼠情况下才可执行。下面我们在程式中加入键盘介面，就如同第六章中对 SYSMETS 程式所做的那样。不过，即使在一个使用滑鼠游标作为指向用途的程式中加入键盘介面，我们还是必须处理滑鼠游标的移动和显示问题。

即使没有安装滑鼠，Windows 仍然可以显示一个滑鼠游标。Windows 为这个游标保存了一个「显示计数」。如果安装了滑鼠，显示计数会被初始化为 0；否则，显示计数会被初始化为 -1。只有在显示计数非负时才显示滑鼠游标。要增

加显示计数，您可以呼叫：

```
ShowCursor (TRUE) ;
```

要减少显示计数，可以呼叫：

```
ShowCursor (FALSE) ;
```

您在使用 ShowCursor 之前，不需要确定是否安装了滑鼠。如果您想显示滑鼠游标，而不管滑鼠存在与否，那么只需呼叫 ShowCursor 来增加显示计数。增加一次显示计数之後，如果没有安装滑鼠则减少它以隐藏游标，如果安装了滑鼠，则保留其显示。

即使没有安装滑鼠，Windows 也保留了滑鼠目前的位置。如果没有安装滑鼠，而您又显示滑鼠游标，游标就可能出现在显示器的任意位置，直到您确实移动了它。要获得游标的位置，可以呼叫：

```
GetCursorPos (&pt) ;
```

其中 pt 是 POINT 结构。函式使用滑鼠的 x 和 y 座标来填入 POINT 栏位。要设定游标位置，可以使用：

```
SetCursorPos (x, y) ;
```

在这两种情况下，x 和 y 都是萤幕座标，而不是显示区域座标（这是很明显的，因为这些函式没有要求 hwnd 参数）。前面已经提到过，呼叫 ScreenToClient 和 ClientToScreen 就能做到萤幕座标与客户座标的相互转换。

如果您在处理滑鼠讯息并转换显示区域座标时呼叫 GetCursorPos，这些座标可能与滑鼠讯息的 lParam 参数中的座标稍微有些不同。从 GetCursorPos 传回的座标表示滑鼠目前的位置。lParam 中的座标则是产生讯息时滑鼠的位置。

您或许想写一个键盘处理程式：使用键盘方向键来移动滑鼠游标，使用 Spacebar 和 Enter 键来模拟滑鼠按键。您肯定不希望每次按键只是将滑鼠游标移动一个图素，如果这样做，当要把滑鼠游标从显示器的一边移动到另一边时，会使用者在很长一段时间内都要按住同一个方向键。

如果您需要实作滑鼠游标的键盘介面，并保持游标的精确定位能力，那么您可以采用下面的方式来处理按键讯息：当按下方向键时，一开始滑鼠游标移动较慢，但随后会加快。您也许还记得 WM_KEYDOWN 讯息中的 lParam 参数标志著按键讯息是否是重复活动的结果，这就是此参数的一个重要应用。

在 CHECKER 中加入键盘介面

程式 7-3 所示的 CHECKER2 程式，除了包括键盘介面外，和 CHECKER1 是一样的，您可以使用左、右、上和下方方向键在 25 个矩形之间移动游标。Home 键把游标移动到矩形的左上角，End 键把游标移动到矩形的右下角。Spacebar 和 Enter 键都能切换 X 标记。

程式 7-3 CHECKER2

```

CHECKER2.C
/*-----
--
CHECKER2.C -- Mouse Hit-Test Demo Program No. 2
              (c) Charles Petzold, 1998
-----
*/

#include <windows.h>

#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Checker2") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL,
IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName
        = NULL ;
    wndclass.lpszClassName
        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Checker2 Mouse Hit-Test Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

```

```
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (    GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static BOOL fState[DIVISIONS][DIVISIONS] ;
    static int    cxBlock, cyBlock ;
    HDC          hdc ;
    int          x, y ;
    PAINTSTRUCT  ps ;
    POINT        point ;
    RECT         rect ;

    switch (message)
    {
    case WM_SIZE :
        cxBlock = LOWORD (lParam) / DIVISIONS ;
        cyBlock = HIWORD (lParam) / DIVISIONS ;
        return 0 ;

    case WM_SETFOCUS :
        ShowCursor (TRUE) ;
        return 0 ;

    case WM_KILLFOCUS :
        ShowCursor (FALSE) ;
        return 0 ;

    case WM_KEYDOWN :
        GetCursorPos (&point) ;
        ScreenToClient (hwnd, &point) ;
        x = max (0, min (DIVISIONS - 1, point.x / cxBlock)) ;
        y = max (0, min (DIVISIONS - 1, point.y / cyBlock)) ;

        switch (wParam)
        {
            case VK_UP :
                y-- ;
                break ;
        }
    }
```



```

        case VK_DOWN :
            y++ ;
            break ;

        case VK_LEFT :
            x-- ;
            break ;

        case VK_RIGHT :
            x++ ;
            break ;

        case VK_HOME :
            x = y = 0 ;
            break ;

        case VK_END :
            x = y = DIVISIONS - 1 ;
            break ;

        case VK_RETURN :
        case VK_SPACE :
            SendMessage (hwnd, WM_LBUTTONDOWN,
MK_LBUTTON,
                        MAKELONG (x * cxBlock, y * cyBlock)) ;
            break ;
    }
    x = (x + DIVISIONS) % DIVISIONS ;
    y = (y + DIVISIONS) % DIVISIONS ;

    point.x = x * cxBlock + cxBlock / 2 ;
    point.y = y * cyBlock + cyBlock / 2 ;

    ClientToScreen (hwnd, &point) ;
    SetCursorPos (point.x, point.y) ;
    return 0 ;
case WM_LBUTTONDOWN :
    x = LOWORD (lParam) / cxBlock ;
    y = HIWORD (lParam) / cyBlock ;

    if (x < DIVISIONS && y < DIVISIONS)
    {
        fState[x][y] ^= 1 ;

        rect.left    = x * cxBlock ;
        rect.top     = y * cyBlock ;
        rect.right   = (x + 1) * cxBlock ;
    }

```

```

rect.bottom = (y + 1) * cyBlock ;

InvalidateRect (hwnd, &rect, FALSE) ;

}

else

    MessageBeep (0) ;

    return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    for (x = 0 ; x < DIVISIONS ; x++)
    for (y = 0 ; y < DIVISIONS ; y++)
    {
        Rectangle (hdc, x * cxBlock, y * cyBlock,
            (x + 1) * cxBlock, (y + 1) * cyBlock) ;

        if (fState [x][y])
        {
            MoveToEx (hdc, x * cxBlock, y * cyBlock, NULL) ;
            LineTo (hdc, (x+1)*cxBlock, (y+1)*cyBlock) ;
            MoveToEx (hdc, x * cxBlock, (y+1)*cyBlock,
NULL) ;

            LineTo (hdc, (x+1)*cxBlock, y * cyBlock) ;
        }
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;

}

return DefWindowProc (hwnd, message, wParam, lParam) ;

}

```

CHECKER2 中的 WM_KEYDOWN 的处理方式决定游标的位置(用 GetCursorPos), 把萤幕座标转换为显示区域座标 (用 ScreenToClient), 并用矩形方块的宽度和高度来除这个座标。这会产生指示矩形位置的 x 和 y 值 (5×5 阵列)。当按下下一个键时, 滑鼠游标可能在或不在显示区域中, 所以 x 和 y 必须经过 min 和 max 巨集处理以保证它们的范围是 0 到 4 之间。

对方向键, CHECKER2 近似地增加或减少 x 和 y。如果是 Enter 键或 Spacebar 键, 那么 CHECKER2 使用 SendMessage 把 WM_LBUTTONDOWN 讯息发送给它自身。这种技术类似于在第六章 SYSMETS 程式中把键盘介面加到视窗卷动列时所使用的方法。WM_KEYDOWN 的处理方式是通过计算指向矩形中心的显示区域座标, 再用 ClientToScreen 转换成萤幕座标, 然後用 SetCursorPos 设定游标位置来实

作的。

将子视窗用於命中测试

有些程式（例如，Windows 的「画图」程式），把显示区域划分为几个小的逻辑区域。「画图」程式在其左边有一个由图示组成的工具功能表区，在底部有颜色功能表区。在这两个区做命中测试的时候，「画图」必须在使用者选中功能表项之前记住功能表的位置。

不过，也可能不需要这么做。实际上，画风经由使用子视窗简化了功能表的绘制和命中测试。子视窗把整个矩形区域划分为几个更小的矩形区，每个子视窗有自己的视窗代号、视窗讯息处理程式和显示区域，每个视窗讯息处理程式接收只适用於它的子视窗的滑鼠讯息。滑鼠讯息中的 lParam 参数含有相当於该子视窗显示区域左上角的座标，而不是其父视窗（那是「画图」的主应用程式视窗）显示区域左上角的座标。

以这种方式使用子视窗有助於程式的结构化和模组化。如果子视窗使用不同的视窗类别，那么每个子视窗都有它自己的视窗讯息处理程式。不同的视窗也可以定义不同的背景颜色和不同的内定游标。在第九章中，我将看到「子视窗控制项」——卷动列、按钮和编辑方块等预先定义的子视窗。现在，我们说明在 CHECKER 程式中是如何使用子视窗的。

CHECKER 中的子视窗

程式 7-4 所示的 CHECKER3 程式，这一版本建立了 25 个处理滑鼠单击的子视窗。它没有键盘介面，但是可以按本章後面的 CHECKER4 程式范例的方法添加。

程式 7-4 CHECKER3

```
CHECKER3.C
/*-----
-
CHECKER3.C -- Mouse Hit-Test Demo Program No. 3
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>

#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ChildWndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szChildClass[] = TEXT ("Checker3_Child") ;
```

```

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Checker3") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    wndclass.lpfnWndProc      = ChildWndProc ;
    wndclass.cbWndExtra       = sizeof (long) ;
    wndclass.hIcon            = NULL ;
    wndclass.lpszClassName    = szChildClass ;

    RegisterClass (&wndclass) ;
    hwnd = CreateWindow (  szAppName, TEXT ("Checker3 Mouse Hit-Test Demo"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

```

```

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static HWND          hwndChild[DIVISIONS][DIVISIONS] ;
    int                  cxBlock, cyBlock, x, y ;

    switch (message)
    {
        case WM_CREATE :
            for (x = 0 ; x < DIVISIONS ; x++)
                for (y = 0 ; y < DIVISIONS ; y++)
                    hwndChild[x][y] = CreateWindow (szChildClass, NULL,
                    WS_CHILDWINDOW | WS_VISIBLE,
                    0, 0, 0, 0,
                    hwnd, (HMENU) (y << 8 | x),
                    (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
                                                                    NULL) ;

            return 0 ;

        case WM_SIZE :
            cxBlock = LOWORD (lParam) / DIVISIONS ;
            cyBlock = HIWORD (lParam) / DIVISIONS ;
            for (x = 0 ; x < DIVISIONS ; x++)
                for (y = 0 ; y < DIVISIONS ; y++)
                    MoveWindow
                                                                    (
                    hwndChild[x][y],
                                                                    x * cxBlock, y * cyBlock,
                                                                    cxBlock, cyBlock, TRUE) ;

            return 0 ;

        case WM_LBUTTONDOWN :
            MessageBeep (0) ;
            return 0 ;

        case WM_DESTROY :
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ChildWndProc (HWND hwnd,   UINT message,
                                WPARAM wParam, LPARAM lParam)
{
    HDC          hdc ;
    PAINTSTRUCT  ps ;
    RECT         rect ;

```

```

switch (message)
{
case WM_CREATE :
    SetWindowLong (hwnd, 0, 0) ;           // on/off flag
    return 0 ;

case WM_LBUTTONDOWN :
    SetWindowLong (hwnd, 0, 1 ^ GetWindowLong (hwnd, 0)) ;
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;
    Rectangle (hdc, 0, 0, rect.right, rect.bottom) ;

    if (GetWindowLong (hwnd, 0))
    {
        MoveToEx (hdc, 0, 0, NULL) ;
        LineTo (hdc, rect.right, rect.bottom) ;
        MoveToEx (hdc, 0, rect.bottom, NULL) ;
        LineTo (hdc, rect.right, 0) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

CHECKER3 有两个视窗讯息处理程式 WndProc 和 ChildWndProc。WndProc 还是主（或父）视窗的视窗讯息处理程式。ChildWndProc 是针对 25 个子视窗的视窗讯息处理程式。这两个视窗讯息处理程式都必须定义为 CALLBACK 函数。

因为视窗讯息处理程式与特定的视窗类别结构相关联，该视窗类别结构由 Windows 呼叫 RegisterClass 函数来注册，CHECKER3 需要两个视窗类别。第一个视窗类别用于主视窗，名为「Checker3」。第二个视窗类别名为「Checker3_Child」。当然，您不必选择像这样有意义的名字。

CHECKER3 在 WinMain 函数中注册了这两个视窗类别。注册完常规的视窗类别之后，CHECKER3 只是简单地重新使用 wndclass 结构中的大多数的栏位来注册 Checker3_Child 类别。无论如何，有四个栏位根据子视窗类别而设定为不同的值：

- pfnWndProc 栏位设定为 ChildWndProc，子视窗类别的视窗讯息处理程式。

- cbWndExtra 栏位设定为 4 位元组，或者更确切地用 sizeof (long)。该栏位告诉 Windows 在其为依据此视窗类别的视窗保留的内部结构中，预留了 4 位元组额外的空间。您能使用此空间来保存每个视窗的可能有所不同的资讯。
- 因为像 CHECKER3 中的子视窗不需要图示，所以 hIcon 栏位设定为 NULL。
- pszClassName 栏位设定为「Checker3_Child」，是类别的名称。

通常，在 WinMain 中，CreateWindow 呼叫建立依据 Checker3 类别的主视窗。然而，当 WndProc 收到 WM_CREATE 讯息後，它呼叫 CreateWindow 25 次以建立 25 个 Checker3_Child 类别的子视窗。表 7-3 是在 WinMain 中 CreateWindow 呼叫的参数，与在建立 25 个子视窗的 WndProc 中 CreateWindow 呼叫的参数间的比较。

表 7-3

参数	主视窗	子视窗
视窗类别	「Checker3」	「Checker3_Child」
视窗标题	「Checker3...」	NULL
视窗样式	WS_OVERLAPPEDWINDOW	WS_CHILDWINDOW WS_VISIBLE
水平位置	CW_USEDEFAULT	0
垂直位置	CW_USEDEFAULT	0
宽度	CW_USEDEFAULT	0
高度	CW_USEDEFAULT	0
父视窗代号	NULL	hwnd
功能表代号/子 ID	NULL	(HMENU) (y << 8 x)
执行实体代号	hInstance	(HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE)
额外参数	NULL	NULL

一般情况下，子视窗要求有关位置和大小参数，但是在 CHECKER3 中的子视窗由 WndProc 确定位置和大小。对于主视窗，因为它本身就是父视窗，所以它的父视窗代号是 NULL。当使用 CreateWindow 呼叫来建立一个子视窗时，就需要父视窗代号了。

主视窗没有功能表，因此参数是 NULL。对于子视窗，相同位置的参数称为子 ID（或子视窗 ID）。这是唯一代表子视窗的数字。像我们在第十一章将看到的一样，在处理对话方块子视窗控制项时，子 ID 显得更为重要。对于 CHECKER3 来说，我只是简单地将子 ID 设定为一个数值，该数值是每个子视窗在 5×5 的主视窗中的 x 和 y 位置的组合。

CreateWindow 函式需要一个执行实体代号。在 WinMain 中, 执行实体代号可以很容易地取得, 因为它是 WinMain 的一个参数。在建立子视窗时, CHECKER3 必须用 GetWindowLong 来从 Windows 为视窗保留的结构中取得 hInstance 值(相对 GetWindowLong, 我也能将 hInstance 的值保存到整体变数, 并直接使用它)。

每一个子视窗都在 hwndChild 阵列中保存了不同的视窗代号。当 WndProc 接收到一个 WM_SIZE 讯息後, 它将为这 25 个子视窗呼叫 MoveWindow。MoveWindow 的参数表示子视窗左上角相对父视窗显示区域的座标、子视窗的宽度和高度以及子视窗是否需要重画。

现在让我们看一下 ChildWndProc。此视窗讯息处理程式为所有这 25 个子视窗处理讯息。ChildWndProc 的 hwnd 参数是子视窗接收讯息的代号。当 ChildWndProc 处理 WM_CREATE 讯息时(因为有 25 个子视窗, 所以要发生 25 次), 它用 SetWindowWord 在视窗结构保留的额外区域中储存一个 0 值(通过在定义视窗类别时使用的 cbWndExtra 来保留的空间)。ChildWndProc 用此值来恢复目前矩形的状态(有 X 或没有 X)。在子视窗中单击时, WM_LBUTTONDOWN 处理常式简单地修改这个整数值(从 0 到 1, 或从 1 到 0), 并使整个子视窗无效。此区域是被单击的矩形。WM_PAINT 的处理很简单, 因为它所绘制的矩形与显示区域一样大。

因为 CHECKER3 的 C 原始码档案和 .EXE 档案比 CHECKER1 的大(更不用说程式的说明了), 我不会试著告诉你说 CHECKER3 比 CHECKER1 更简单。但请注意, 我们没有做任何滑鼠命中测试! 我们所要的, 就是知道 CHECKER3 中是否有个子视窗得到了命中视窗的 WM_LBUTTONDOWN 讯息。

子视窗和键盘

为 CHECKER3 添加键盘介面就像 CHECKER 系列构想中的最後一步。但在这样做的时候, 可能有更适当的做法。在 CHECKER2 中, 滑鼠游标的位置决定按下 Spacebar 键时哪个区域将获得标记符号。当我们处理子视窗时, 我们能从对话方块功能中获得提示。在对话方块中, 带有闪烁的插入符号或点划的矩形的子视窗表示它有输入焦点(当然也可以用键盘进行定位)。

我们不需要把 Windows 内部已有的对话方块处理方式重新写过, 我只是要告诉您大致上应该如何在应用程式中模拟对话方块。研究过程中, 您会发现这样一件事: 父视窗和子视窗可能要共用同键盘讯息处理。按下 Spacebar 键和 Enter 键时, 子视窗将锁定复选标记。按下方向键时, 父视窗将在子视窗之间移动输入焦点。实际上, 当您在子视窗上单击时, 情况会有些复杂, 这时是父视窗而不是子视窗获得输入焦点。

CHECKER4.C 如程式 7-5 所示。

程式 7-5 CHECKER4

```
CHECKER4.C
/*-----
-
CHECKER4.C -- Mouse Hit-Test Demo Program No. 4
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ChildWndProc (HWND, UINT, WPARAM, LPARAM) ;
int idFocus = 0 ;
TCHAR szChildClass[] = TEXT ("Checker4_Child") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Checker4") ;
    HWND hwnd ;
    MSG msg ;
    WNDCLASS wndclass ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL,
IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("Program requires Windows NT!"),
szAppName,
MB_ICONERROR) ;
        return 0 ;
    }
}
```

```

    wndclass.lpfnWndProc          = ChildWndProc ;
    wndclass.cbWndExtra           = sizeof (long) ;
    wndclass.hIcon                = NULL ;
    wndclass.lpszClassName       = szChildClass ;

    RegisterClass (&wndclass) ;
    hwnd = CreateWindow (  szAppName, TEXT ("Checker4 Mouse Hit-Test Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static HWND      hwndChild[DIVISIONS][DIVISIONS] ;
    int              cxBlock, cyBlock, x, y ;

    switch (message)
    {
    case WM_CREATE :
        for (x = 0 ; x < DIVISIONS ; x++)
            for (y = 0 ; y < DIVISIONS ; y++)
                hwndChild[x][y] = CreateWindow (szChildClass, NULL,
                    WS_CHILDWINDOW | WS_VISIBLE,
                    0, 0, 0, 0,
                    hwnd, (HMENU) (y << 8 | x),
                    HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
                    NULL) ;

        return 0 ;
    case WM_SIZE :
        cxBlock = LOWORD (lParam) / DIVISIONS ;
        cyBlock = HIWORD (lParam) / DIVISIONS ;

        for (x = 0 ; x < DIVISIONS ; x++)
            for (y = 0 ; y < DIVISIONS ; y++)
                MoveWindow (      hwndChild[x][y],

```

```

        x * cxBlock, y * cyBlock,
        cxBlock, cyBlock, TRUE) ;
    return 0 ;

case WM_LBUTTONDOWN :
    MessageBeep (0) ;
    return 0 ;

    // On set-focus message, set focus to child window
case WM_SETFOCUS:
    SetFocus (GetDlgItem (hwnd, idFocus)) ;
    return 0 ;

    // On key-down message, possibly change the focus window

case WM_KEYDOWN:
    x = idFocus & 0xFF ;
    y = idFocus >> 8 ;

    switch (wParam)
    {
    case VK_UP:          y-- ;
break ;
        case VK_DOWN:          y++ ;
break ;
        case VK_LEFT:          x-- ;
break ;
        case VK_RIGHT:          x++ ;
break ;
        case VK_HOME:          x = y = 0 ;
break ;
        case VK_END:          x = y = DIVISIONS - 1 ;
break ;
        default:
return 0 ;
    }

    x = (x + DIVISIONS) % DIVISIONS ;
    y = (y + DIVISIONS) % DIVISIONS ;

    idFocus = y << 8 | x ;

    SetFocus (GetDlgItem (hwnd, idFocus)) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

```
LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT message,
                                WPARAM wParam, LPARAM lParam)
{
    HDC          hdc ;
    PAINTSTRUCT   ps ;
    RECT          rect ;

    switch (message)
    {
        case WM_CREATE :
            SetWindowLong (hwnd, 0, 0) ;      // on/off flag
            return 0 ;

        case WM_KEYDOWN:
            // Send most key presses to the parent window

            if (wParam != VK_RETURN && wParam != VK_SPACE)
            {
                SendMessage (GetParent (hwnd), message, wParam, lParam) ;
                return 0 ;
            }

            // For Return and Space, fall through to toggle the
square

        case WM_LBUTTONDOWN :
            SetWindowLong (hwnd, 0, 1 ^ GetWindowLong (hwnd, 0)) ;
            SetFocus (hwnd) ;
            InvalidateRect (hwnd, NULL, FALSE) ;
            return 0 ;

            // For focus messages, invalidate the window for repaint

        case WM_SETFOCUS:
            idFocus = GetWindowLong (hwnd, GWL_ID) ;

            // Fall through

        case WM_KILLFOCUS:
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;

        case WM_PAINT :
            hdc = BeginPaint (hwnd, &ps) ;

            GetClientRect (hwnd, &rect) ;
            Rectangle (hdc, 0, 0, rect.right, rect.bottom) ;

            // Draw the "x" mark
```

```

        if (GetWindowLong (hwnd, 0))
        {
            MoveToEx (hdc, 0, 0, NULL) ;
            LineTo   (hdc, rect.right, rect.bottom) ;
            MoveToEx (hdc, 0, rect.bottom, NULL) ;
            LineTo   (hdc, rect.right, 0) ;
        }

        // Draw the "focus" rectangle

        if (hwnd == GetFocus ())
        {
            rect.left  += rect.right / 10 ;
            rect.right -= rect.left ;
            rect.top    += rect.bottom / 10 ;
            rect.bottom -= rect.top ;

            SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
            SelectObject (hdc, CreatePen (PS_DASH, 0, 0)) ;
            Rectangle (hdc, rect.left, rect.top, rect.right,
rect.bottom) ;

            DeleteObject (SelectObject (hdc, GetStockObject
(BLACK_PEN))) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

您应该能回忆起每一个子视窗有唯一的子视窗 ID，该 ID 在呼叫 CreateWindow 建立视窗时定义。在 CHECKER3 中，此 ID 是矩形的 x 和 y 位置的组合。一个程式可以通过下面的呼叫来获得一个特定子视窗的子视窗 ID：

```
idChild = GetWindowLong (hwndChild, GWL_ID) ;
```

下面的函式也有同样的功能：

```
idChild = GetDlgCtrlID (hwndChild) ;
```

正如函式名称所表示的，它主要用於对话方块和控制视窗。如果您知道父视窗的代号和子视窗 ID，此函式也可以获得子视窗的代号：

```
hwndChild = GetDlgItem (hwndParent, idChild) ;
```

在 CHECKER4 中，整体变数 idFocus 用於保存目前输入焦点视窗的子视窗 ID。我在前面说过，当您在子视窗上面单击滑鼠时，它们不会自动获得输入焦点。因此，CHECKER4 中的父视窗将通过呼叫下面的函式来处理 WM_SETFOCUS 讯息：

```
SetFocus (GetDlgItem (hwnd, idFocus)) ;
```

这样设定一个子视窗为输入焦点。

ChildWndProc 处理 WM_SETFOCUS 和 WM_KILLFOCUS 讯息。对于 WM_SETFOCUS，它将保存在整体变数 idFocus 中接收输入焦点的子视窗 ID。对于这两种讯息，视窗是无效的，并产生一个 WM_PAINT 讯息。如果 WM_PAINT 讯息画出了有输入焦点的子视窗，则它将用 PS_DASH 画笔的风格画一个矩形以表示此视窗有输入焦点。

ChildWndProc 也处理 WM_KEYDOWN 讯息。对于除了 Spacebar 和 Enter 键以外的其他讯息，WM_KEYDOWN 都将给父视窗发送讯息。另外，视窗讯息处理程式也处理类似 WM_LBUTTONDOWN 讯息的讯息。

处理方向移动键是父视窗的事情。在风格相似的 CHECKER2 中，此程式可获得有输入焦点的子视窗的 x 和 y 座标，并根据按下的特定方向键来改变它们。然后通过呼叫 SetFocus 将输入焦点设定给新的子视窗。

拦截滑鼠

一个视窗讯息处理程式通常只在滑鼠游标位于视窗的显示区域，或非显示区域上时才接收滑鼠讯息。一个程式也可能需要在滑鼠位于视窗外时接收滑鼠讯息。如果是这样，程式可以自行「拦截」滑鼠。别害怕，这么做没什么大不了的。

设计矩形

为了说明拦截滑鼠的必要性，请让我们看一下 BLOKOUT1 程式（如程式 7-6 所示）。此程式看起来达到了一定的功能，但它却有十分严重的缺陷。

程式 7-6 BLOKOUT1

```
BLOKOUT1.C
/*-----
--
BLOKOUT1.C -- Mouse Button Demo Program
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR    szAppName[] = TEXT ("BlokOut1") ;
    HWND            hwnd ;
```

```

MSG                                msg ;
WNDCLASS                           wndclass ;

wndclass.style                      = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc                = WndProc ;
wndclass.cbClsExtra                 = 0 ;
wndclass.cbWndExtra                 = 0 ;
wndclass.hInstance                 = hInstance ;
wndclass.hIcon                     = LoadIcon (NULL,
IDI_APPLICATION) ;
wndclass.hCursor                   = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground             = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
wndclass.lpszMenuName              = NULL ;
wndclass.lpszClassName              = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
szAppName,
MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Mouse Button Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void DrawBoxOutline (HWND hwnd, POINT ptBeg, POINT ptEnd)
{
    HDC hdc ;
    hdc = GetDC (hwnd) ;
    SetROP2 (hdc, R2_NOT) ;
    SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
    Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;
}

```

```
    ReleaseDC (hwnd, hdc) ;
}
LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static BOOL        fBlocking, fValidBox ;
    static POINT       ptBeg, ptEnd, ptBoxBeg, ptBoxEnd ;
    HDC                hdc ;
    PAINTSTRUCT ps ;
    switch (message)
    {
    case WM_LBUTTONDOWN :
        ptBeg.x = ptEnd.x = LOWORD (lParam) ;
        ptBeg.y = ptEnd.y = HIWORD (lParam) ;

        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

        SetCursor (LoadCursor (NULL, IDC_CROSS)) ;

        fBlocking = TRUE ;
        return 0 ;

    case WM_MOUSEMOVE :
        if (fBlocking)
        {
            SetCursor (LoadCursor (NULL, IDC_CROSS)) ;

            DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

            ptEnd.x = LOWORD (lParam) ;
            ptEnd.y = HIWORD (lParam) ;

            DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
        }
        return 0 ;

    case WM_LBUTTONUP :
        if (fBlocking)
        {
            DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

            ptBoxBeg                = ptBeg ;
            ptBoxEnd.x              = LOWORD (lParam) ;
            ptBoxEnd.y              = HIWORD (lParam) ;

            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
            fBlocking                = FALSE ;
        }
    }
```



```

        fValidBox                = TRUE ;

        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

case WM_CHAR :
    if (fBlocking & wParam == '\x1B')        // i.e., Escape
    {
        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        fBlocking = FALSE ;
    }
    return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    if (fValidBox)
    {
        SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
        Rectangle (    hdc, ptBoxBeg.x, ptBoxBeg.y,
                        ptBoxEnd.x, ptBoxEnd.y) ;
    }

    if (fBlocking)
    {
        SetROP2 (hdc, R2_NOT) ;
        SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
        Rectangle (hdc,  ptBeg.x,  ptBeg.y,  ptEnd.x,
ptEnd.y) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

此程式展示了一些，它可以实作在 Windows 的「画图」程式中的东西。由按下滑鼠左键开始确定矩形的一角，然後拖动滑鼠。程式将画一个矩形的轮廓，其相对位置是滑鼠目前的位置。当您释放滑鼠後，程式将填入这个矩形。图 7-4

显示了一个已经画完的矩形和另一个正在画的矩形。

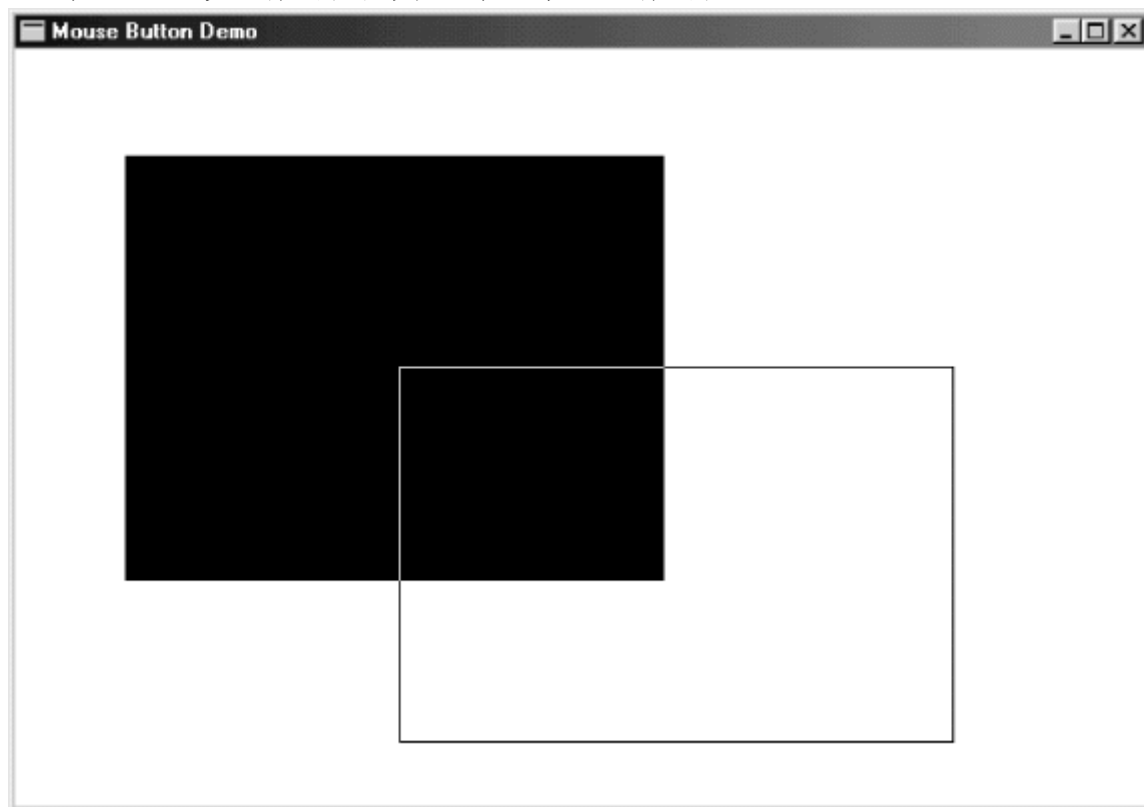


图 7-4 BLOKOUT1 的萤幕显示

那么，问题在哪里呢？

请试一试下面的操作：在 BLOKOUT1 的显示区域按下滑鼠的左键，然后将游标移出视窗。程式将停止接收 WM_MOUSEMOVE 讯息。现在释放按钮，BLOKOUT1 将不再获得 WM_BUTTONUP 讯息，因为游标在显示区域以外。然后将游标移回 BLOKOUT1 的显示区域，视窗讯息处理程式仍然认为按钮处于按下状态。

这样做并不好，因为程式不知道发生了什么事情。

拦截的解决方案

BLOKOUT1 显示了一些常见的程式功能，但它的程式码显然有缺陷。这种问题就是要使用滑鼠拦截来对付。如果使用者正在拖曳滑鼠，那么当滑鼠短时间内被拖出视窗时应该没有什么大问题，程式应该仍然控制著滑鼠。

拦截滑鼠要比放置一个老鼠夹子容易一些，您只要呼叫：

```
SetCapture (hwnd) ;
```

在这个函式呼叫之後，Windows 将所有滑鼠讯息发给视窗代号为 hwnd 的视窗讯息处理程式。之後收到滑鼠讯息都是以显示区域讯息的型态出现，即使滑鼠正在视窗的非显示区域。lParam 参数将指示滑鼠在显示区域座标中的位置。不过，当滑鼠位于显示区域的左边或者上方时，这些 x 和 y 座标可以是负的。当您想释放滑鼠时，呼叫：

```
ReleaseCapture () ;
```

从而使处理恢复正常。

在 32 位元的 Windows 中，滑鼠拦截要比在以前的 Windows 版本中有多一些限制。特别是，如果滑鼠被拦截，而滑鼠按键目前并未被按下，并且滑鼠游标移到了另一个视窗上，那么将不是由拦截滑鼠的那个视窗，而是由游标下面的视窗来接收滑鼠讯息。对于防止一个程式在拦截滑鼠之後不释放它而引起整个系统的混乱，这是必要的。

换句话说，只有当滑鼠按键在您的显示区域中被按下时才拦截滑鼠；当滑鼠按键被释放时，才释放滑鼠拦截。

BLOKOUT2 程式

展示滑鼠拦截的 BLOKOUT2 程式如程式 7-7 所示。

程式 7-7 BLOKOUT2

```
BLOKOUT2.C
/*-----
--
--      BLOKOUT2.C --      Mouse Button & Capture Demo Program
--                               (c) Charles Petzold, 1998
--
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("BlokOut2") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
```

```
{
    MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Mouse Button & Capture Demo"),
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void DrawBoxOutline (HWND hwnd, POINT ptBeg, POINT ptEnd)
{
    HDC hdc ;
    hdc = GetDC (hwnd) ;
    SetROP2 (hdc, R2_NOT) ;
    SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
    Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;

    ReleaseDC (hwnd, hdc) ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static BOOL  fBlocking, fValidBox ;
    static POINT ptBeg, ptEnd, ptBoxBeg, ptBoxEnd ;
    HDC          hdc ;
    PAINTSTRUCT  ps ;

    switch (message)
    {
    case WM_LBUTTONDOWN :
        ptBeg.x = ptEnd.x = LOWORD (lParam) ;
        ptBeg.y = ptEnd.y = HIWORD (lParam) ;
```

```
        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

        SetCapture (hwnd) ;
        SetCursor (LoadCursor (NULL, IDC_CROSS)) ;

        fBlocking = TRUE ;
        return 0 ;

case WM_MOUSEMOVE :
    if (fBlocking)
    {
        SetCursor (LoadCursor (NULL, IDC_CROSS)) ;

        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

        ptEnd.x = LOWORD (lParam) ;
        ptEnd.y = HIWORD (lParam) ;

        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
    }
    return 0 ;

case WM_LBUTTONDOWN :
    if (fBlocking)
    {
        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;

        ptBoxBeg                = ptBeg ;
        ptBoxEnd.x               = LOWORD (lParam) ;
        ptBoxEnd.y               = HIWORD (lParam) ;

        ReleaseCapture () ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        fBlocking                = FALSE ;
        fValidBox                 = TRUE ;

        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

case WM_CHAR :
    if (fBlocking & wParam == '\x1B') // i.e., Escape
    {
        DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
        ReleaseCapture () ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
    }
}
```

```

        fBlocking = FALSE ;
    }
    return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    if (fValidBox)
    {
        SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
        Rectangle (hdc, ptBoxBeg.x, ptBoxBeg.y,
            ptBoxEnd.x, ptBoxEnd.y) ;
    }

    if (fBlocking)
    {
        SetROP2 (hdc, R2_NOT) ;
        SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
        Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x,
ptEnd.y) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

BLOKOUT2 程式和 BLOKOUT1 程式一样，只是多了三行新程式码：在 WM_LBUTTONDOWN 讯息处理期间呼叫 SetCapture，而在 WM_LBUTTONDOWN 和 WM_CHAR 讯息处理期间呼叫 ReleaseCapture。检查画出视窗：使视窗小於萤幕大小，开始在显示区域画出一块矩形，然後将滑鼠游标移出显示区域的右边或下边，最後释放滑鼠按键。程式将获得整个矩形的座标。但是需要扩大视窗才能看清楚它。

拦截滑鼠并非只适用於那些古怪的应用程式。如果您需要滑鼠按键在显示区域按下时都能够追踪 WM_MOUSEMOVE 讯息，并直到滑鼠按键被释放为止，那么您就应该拦截滑鼠。这样将简化您的程式，同时又符合使用者的期望。

滑鼠滑轮

与传统的滑鼠相比，Microsoft IntelliMouse 的特点是在两个键之间多了

一个小滑轮。您可以按下这个滑轮，这时它的功能相当於滑鼠按键的中键；或者您也可以用食指来转动它，这会产生一条特殊的讯息，叫做 WM_MOUSEWHEEL。使用滑鼠滑轮的程式通过滚动或放大文件来回应此讯息。它最初听起来像一个不必要的隐藏机关，但我必须承认，我很快就习惯於使用滑鼠滑轮来滚动 Microsoft Word 和 Microsoft Internet Explorer 了。

我不想讨论滑鼠滑轮的所有使用方法。实际上，我只是想告诉您如何在现有的程式（例如程式 SYSMETS4）中添加滑鼠滑轮处理程式，以便在显示区域中卷动资料。最终的 SYSMETS 程式如程式 7-8 所示。

程式 7-8 SYSMETS4

```
SYSMETS.C
/*-----
--
SYSMETS.C -- Final System Metrics Display Program
              (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include "sysmets.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("SysMets") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;
    wndclass.style     = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                        szAppName,
```

```

MB_ICONERROR) ;

        return 0 ;

    }

    hwnd = CreateWindow (  szAppName, TEXT ("Get System Metrics"),
                          WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      int      cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
    static      int      iDeltaPerLine, iAccumDelta ;                //
for mouse wheel logic
    HDC          hdc ;
    int          i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
    PAINTSTRUCT      ps ;
    SCROLLINFO        si ;
    TCHAR            szBuffer[10] ;
    TEXTMETRIC        tm ;
    ULONG            ulScrollLines ;                // for mouse
wheel logic
    switch (message)
    {
    case  WM_CREATE:
        hdc = GetDC (hwnd) ;

        GetTextMetrics (hdc, &tm) ;
        cxChar      = tm.tmAveCharWidth ;
        cxCaps      = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        ReleaseDC (hwnd, hdc) ;

        // Save the width of the three columns

```



```

        iMaxWidth = 40 * cxChar + 22 * cxCaps ;

                                // Fall through for mouse wheel
information

        case WM_SETTINGCHANGE:
                SystemParametersInfo (SPI_GETWHEELSCROLLLINES, 0,
&ulScrollLines, 0) ;

                // ulScrollLines usually equals 3 or 0 (for no scrolling)
                // WHEEL_DELTA equals 120, so iDeltaPerLine will be 40

                if (ulScrollLines)
                        iDeltaPerLine = WHEEL_DELTA / ulScrollLines ;
                else
                        iDeltaPerLine = 0 ;

        return 0 ;

        case WM_SIZE:
                cxClient = LOWORD (lParam) ;
                cyClient = HIWORD (lParam) ;

                                // Set vertical scroll bar range and page
size

                si.cbSize          = sizeof (si) ;
                si.fMask            = SIF_RANGE | SIF_PAGE ;
                si.nMin             = 0 ;
                si.nMax             = NUMLINES - 1 ;
                si.nPage            = cyClient / cyChar ;
                SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;

                                // Set horizontal scroll bar range and
page size

                si.cbSize          = sizeof (si) ;
                si.fMask            = SIF_RANGE | SIF_PAGE ;
                si.nMin             = 0 ;
                si.nMax             = 2 + iMaxWidth / cxChar ;
                si.nPage            = cxClient / cxChar ;
                SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
                return 0 ;

        case WM_VSCROLL:
                                // Get all the vertical scroll bar
information

```

```
        si.cbSize = sizeof (si) ;
        si.fMask = SIF_ALL ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;

                                // Save the position for comparison
later on

        iVertPos = si.nPos ;
        switch (LOWORD (wParam))
        {
        case SB_TOP:
                si.nPos = si.nMin ;
                break ;

        case SB_BOTTOM:
                si.nPos = si.nMax ;
                break ;

        case SB_LINEUP:
                si.nPos -= 1 ;
                break ;

        case SB_LINEDOWN:
                si.nPos += 1 ;
                break ;

        case SB_PAGEUP:
                si.nPos -= si.nPage ;
                break ;

        case SB_PAGEDOWN:
                si.nPos += si.nPage ;
                break ;

        case SB_THUMBTRACK:
                si.nPos = si.nTrackPos ;
                break ;

        default:
                break ;
        }

                                // Set the position and then retrieve it. Due
to adjustments

                                // by Windows it may not be the same as the value set.

        si.fMask = SIF_POS ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;

                                // If the position has changed, scroll the
```

```
window and update it

        if (si.nPos != iVertPos)
        {
            ScrollWindow (    hwnd, 0, cyChar * (iVertPos -
si.nPos),
                            NULL, NULL) ;
            UpdateWindow (hwnd) ;
        }
        return 0 ;

case WM_HSCROLL:
                                // Get all the vertical scroll bar information

        si.cbSize = sizeof (si) ;
        si.fMask = SIF_ALL ;

                                // Save the position for comparison later on

        GetScrollInfo (hwnd, SB_HORZ, &si) ;
        iHorzPos = si.nPos ;

        switch (LOWORD (wParam))
        {
        case SB_LINELEFT:
                si.nPos -= 1 ;
                break ;

        case SB_LINERIGHT:
                si.nPos += 1 ;
                break ;

        case SB_PAGELEFT:
                si.nPos -= si.nPage ;
                break ;

        case SB_PAGERIGHT:
                si.nPos += si.nPage ;
                break ;

        case SB_THUMBPOSITION:
                si.nPos = si.nTrackPos ;
                break ;

        default:
                break ;
        }

                                // Set the position and then retrieve it. Due to
```

```
adjustments

        // by Windows it may not be the same as the value set.

    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

        // If the position has changed, scroll the window

    if (si.nPos != iHorzPos)
    {
        ScrollWindow (    hwnd, cxChar * (iHorzPos -
si.nPos), 0,
                                NULL, NULL) ;
    }
    return 0 ;

case WM_KEYDOWN :
    switch (wParam)
    {
    case VK_HOME :
        SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
        break ;

    case VK_END :
        SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
        break ;

    case VK_PRIOR :
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
        break ;

    case VK_NEXT :
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ;
        break ;

    case VK_UP :
        SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
        break ;

    case VK_DOWN :
        SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
        break ;

    case VK_LEFT :
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEUP, 0) ;
        break ;
```

```
        case VK_RIGHT :
            SendMessage (hwnd, WM_HSCROLL, SB_PAGEDOWN, 0) ;
            break ;
    }
    return 0 ;

case WM_MOUSEWHEEL:
    if (iDeltaPerLine == 0)
        break ;

    iAccumDelta += (short) HIWORD (wParam) ;    // 120 or -120

    while (iAccumDelta >= iDeltaPerLine)
    {
        SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
        iAccumDelta -= iDeltaPerLine ;
    }

    while (iAccumDelta <= -iDeltaPerLine)
    {
        SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
        iAccumDelta += iDeltaPerLine ;
    }

    return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    // Get vertical scroll bar position

    si.cbSize      = sizeof (si) ;
    si.fMask       = SIF_POS ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    iVertPos       = si.nPos ;

    // Get horizontal scroll bar position

    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos = si.nPos ;

    // Find painting limits

    iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
    iPaintEnd = min (NUMLINES - 1,
        iVertPos + ps.rcPaint.bottom / cyChar) ;

    for (i = iPaintBeg ; i <= iPaintEnd ; i++)
```

```

        {
            x = cxChar * (1 - iHorzPos) ;
            y = cyChar * (i - iVertPos) ;

            TextOut (   hdc, x, y,
sysmetrics[i].szLabel,
            lstrlen (sysmetrics[i].szLabel)) ;

            TextOut (   hdc, x + 22 * cxCaps, y,
sysmetrics[i].szDesc,
            lstrlen (sysmetrics[i].szDesc)) ;

            SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;

            TextOut (   hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
wsprintf (szBuffer, TEXT ("%5d"),
GetSystemMetrics (sysmetrics[i].iIndex))) ;

            SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

转动滑轮会导致 Windows 在有输入焦点的视窗（不是滑鼠游标下面的视窗）产生 WM_MOUSEWHEEL 讯息。与平常一样，lParam 将获得滑鼠的位置，当然座标是相对於萤幕左上角的，而不是显示区域的。另外，wParam 的低字组包含一系列的旗标，用於表示滑鼠按键、Shift 与 Ctrl 键的状态。

新的资讯保存在 wParam 的高字组。其中有一个「delta」值，该值目前可以是 120 或-120，这取决於滑轮的向前转动（也就是说，向滑鼠的前面，即带有按钮与电缆的一端）还是向後转动。值 120 或-120 表示文件将分别向上或向下卷动三行。这里的构想是，以後版本的滑鼠滑轮能有比现在的滑鼠产生更精确的移动速度资讯，并且用 delta 值，例如 40 和-40，来产生 WM_MOUSEWHEEL 讯息。这些值能使文件只向上或向下卷动一行。

为使程式能在一般化环境执行，SYSMETS 将在 WM_CREATE 和 WM_SETTINGCHANGE 讯息处理时，以 SPI_GETWHEELSCROLLLINES 作为参数来呼叫 SystemParametersInfo。此值说明 WHEEL_DELTA 的 delta 值将滚动多少行，

WHEEL_DELTA 在 WINUSER.H 中定义。WHEEL_DELTA 等於 120, 并且, 在内定情况下 SystemParametersInfo 传回 3, 因此与滚动一行相联系的 delta 值就是 40。SYSMETs 将此值保存在 iDeltaPerLine。

在 WM_MOUSEWHEEL 讯息处理期间, SYSMETs 将 delta 值给静态变数 iAccumDelta。然後, 如果 iAccumDelta 大於或等於 iDeltaPerLine (或者是小於或等於 -iDeltaPerLine), SYSMETs 用 SB_LINEUP 或 SB_LINEDOWN 值产生 WM_VSCROLL 讯息。對於每一个 WM_VSCROLL 讯息, iAccumDelta 由 iDeltaPerLine 增加 (或减少)。此代码允许 delta 值大於、小於或等於滚动一行所需要的 delta 值。

下面还有

还有一个引人注目的滑鼠问题: 建立自订滑鼠游标。我将在第十章, 与其他 Windows 资源一起讨论此问题。

第八章 计时器

Microsoft Windows 计时器是一种输入设备，它周期性地在每经过一个指定的时间间隔后就通知應用程式一次。您的程式将时间间隔告诉 Windows，例如「每 10 秒钟通知我一声」，然後 Windows 给您的程式发送周期性发生的 WM_TIMER 讯息以表示时间到了。

初看之下，Windows 计时器似乎不如键盘和滑鼠设备重要，而且对许多應用程式来说确实如此。但是，计时器比您可能认为的要重要得多，它不只用於计时程式，比如出现在工具列中的 Windows 时钟和这一章中的两个时钟程式。下面是 Windows 计时器的其他应用，有些可能并不那么明显：

多工 虽然 Windows 98 是一个优先权式的多工环境，但有时候如果程式尽快将控制传回给 Windows 效率会更高。如果一个程式必须进行大量的处理，那么它可以将作业分成小块，每接收到一个 WM_TIMER 讯息处理一块（我将在第二十章中对此做更多的讨论）。

维护更新过的状态报告 程式可以利用计时器来显示持续变化资讯的「即时」更新，比如关于系统资源的变化或某个任务的进展情况。

实作「自动储存」功能 计时器提示 Windows 程式在指定的时间过去後把使用者的工作储存到磁片上。

终止程式展示版本的执行 一些程式的展示版本被设计成在其开始後，多长时间结束，比如说，30 分钟。如果时间已到，那么计时器就会通知應用程式。

步进移动 游戏中的图形物件或电脑辅助教学程式中的连续显示，需要按指定的速率来处理。利用计时器可以消除由於微处理器速度不同而造成的不一致。

多媒体 播放 CD 声音、声音或音乐的程式通常在背景播放声音资料。一个程式可以使用计时器来周期性地检查已播放了多少声音资料，并据此协调萤幕上的视觉资讯。

另一项应用可以保证程式在退出视窗讯息处理程式後，能够重新得到控制。在大多数情况下，程式不能够知道何时下一个讯息会到来。

计时器入门

您可以通过呼叫 SetTimer 函式为您的 Windows 程式分配一个计时器。SetTimer 有一个时间间隔范围为 1 毫秒到 4,294,967,295 毫秒（将近 50 天）的整数型态参数，这个值指示 Windows 每隔多久时间给您的程式发送 WM_TIMER 讯息。例如，如果间隔为 1000 毫秒，那么 Windows 将每秒给程式发送一个 WM_TIMER

讯息。

当您的程式用完计时器时，它呼叫 KillTimer 函式来停止计时器讯息。在处理 WM_TIMER 讯息时，您可以通过呼叫 KillTimer 函式来编写一个「限用一次」的计时器。KillTimer 呼叫清除讯息伫列中尚未被处理的 WM_TIMER 讯息，从而使程式在呼叫 KillTimer 之後就不会再接收到 WM_TIMER 讯息。

系统和计时器

Windows 计时器是 PC 硬体和 ROM BIOS 架构下之计时器一种相对简单的扩充。回到 Windows 以前的 MS-DOS 程式写作环境下，应用程式能够通过拦截者称为 timer tick 的 BIOS 中断来实作时钟或计时器。一些为 MS-DOS 编写的程式自己拦截这个硬体中断以实作时钟和计时器。这些中断每 54.915 毫秒产生一次，或者大约每秒 18.2 次。这是原始的 IBM PC 的微处理器时脉值 4.772720 MHz 被 218 所除而得出的结果。

Windows 应用程式不拦截 BIOS 中断，相反地，Windows 本身处理硬体中断，这样应用程式就不必进行处理。对于目前拥有计时器的每个程式，Windows 储存一个每次硬体 timer tick 减少的计数。当这个计数减到 0 时，Windows 在应用程式讯息伫列中放置一个 WM_TIMER 讯息，并将计数重置为其最初值。

因为 Windows 应用程式从正常的讯息伫列中取得 WM_TIMER 讯息，所以您的程式在进行其他处理时不必担心 WM_TIMER 讯息会意外中断了程式。在这方面，计时器类似於键盘和滑鼠。驱动程式处理非同步硬体中断事件，Windows 把这些事件·译为规律、结构化和顺序化的讯息。

在 Windows 98 中，计时器与其下的 PC 计时器一样具有 55 毫秒的解析度。在 Microsoft Windows NT 中，计时器的解析度为 10 毫秒。

Windows 应用程式不能以高於这些解析度的频率（在 Windows 98 下，每秒 18.2 次，在 Windows NT 下，每秒大约 100 次）接收 WM_TIMER 讯息。在 SetTimer 呼叫中指定的时间间隔总是截尾後 tick 数的整数倍。例如，1000 毫秒的间隔除以 54.925 毫秒，得到 18.207 个 tick，截尾後是 18 个 tick，它实际上是 989 毫秒。对每个小於 55 毫秒的间隔，每个 tick 都会产生一个 WM_TIMER 讯息。

计时器讯息不是非同步的

因为计时器使用硬体计时器中断，程式写作者有时会误解，认为他们的程式会非同步地被中断来处理 WM_TIMER 讯息。

然而，WM_TIMER 讯息并不是非同步的。WM_TIMER 讯息放在正常的讯息伫列之中，和其他讯息排列在一起，因此，如果在 SetTimer 呼叫中指定间隔为 1000

毫秒，那么不能保证程式每 1000 毫秒或者 989 毫秒就会收到一个 WM_TIMER 讯息。如果其他程式的执行事件超过一秒，在此期间内，您的程式将收不到任何 WM_TIMER 讯息。您可以使用本章的程式来展示这一点。事实上，Windows 对 WM_TIMER 讯息的处理非常类似於对 WM_PAINT 讯息的处理，这两个讯息都是低优先顺序的，程式只有在讯息伫列中没有其他讯息时才接收它们。

WM_TIMER 还在另一方面和 WM_PAINT 相似：Windows 不能持续向讯息伫列中放入多个 WM_TIMER 讯息，而是将多余的 WM_TIMER 讯息组合成一个讯息。因此，应用程式不会一次收到多个这样的讯息，尽管可能在短时间内得到两个 WM_TIMER 讯息。应用程式不能确定这种处理方式所导致的 WM_TIMER 讯息「遗漏」的数目。

这样，WM_TIMER 讯息仅仅在需要更新时才提示程式，程式本身不能经由统计 WM_TIMER 讯息的数目来计时（在本章後面，我们将编写两个每秒更新一次的时钟程式，并可以看到如何做到这一点）。

为了方便起见，下面在讨论时钟时，我将使用「每秒得到一次 WM_TIMER 讯息」这样的叙述，但是请记住，这些讯息并非精确的 tick 中断。

计时器的使用：三种方法

如果您需要在整个程式执行期间都使用计时器，那么您将得从 WinMain 函式中或者在处理 WM_CREATE 讯息时呼叫 SetTimer，并在退出 WinMain 或回应 WM_DESTROY 讯息时呼叫 KillTimer。根据呼叫 SetTimer 时使用的参数，可以下列三种方法之一使用计时器。

方法一

这是最方便的一种方法，它让 Windows 把 WM_TIMER 讯息发送到应用程式的正常视窗讯息处理程式中，SetTimer 呼叫如下所示：

```
SetTimer (hwnd, 1, uiMsecInterval, NULL) ;
```

第一个参数是其视窗讯息处理程式将接收 WM_TIMER 讯息的视窗代号。第二个参数是计时器 ID，它是一个非 0 数值，在整个例子中假定为 1。第三个参数是一个 32 位元无正负号整数，以毫秒为单位指定一个时间间隔，一个 60,000 的值将使 Windows 每分钟发送一次 WM_TIMER 讯息。

您可以通过呼叫

```
KillTimer (hwnd, 1) ;
```

在任何时刻停止 WM_TIMER 讯息（即使正在处理 WM_TIMER 讯息）。此函式的第二个参数是 SetTimer 呼叫中所用的同一个计时器 ID。在终止程式之前，您

应该回应 WM_DESTROY 讯息停止任何活动的计时器。

当您的视窗讯息处理程式收到一个 WM_TIMER 讯息时, wParam 参数等於计时器的 ID 值 (上述情形为 1), lParam 参数为 0。如果需要设定多个计时器, 那么对每个计时器都使用不同的计时器 ID。wParam 的值将随传递到视窗讯息处理程式的 WM_TIMER 讯息的不同而不同。为了使程式更具有可读性, 您可以使用 #define 叙述定义不同的计时器 ID:

```
#define TIMER_SEC 1
#define TIMER_MIN 2
```

然後您可以使用两个 SetTimer 呼叫来设定两个计时器:

```
SetTimer (hwnd, TIMER_SEC, 1000, NULL) ;
SetTimer (hwnd, TIMER_MIN, 60000, NULL) ;
```

WM_TIMER 的处理如下所示:

```
case WM_TIMER:
    switch (wParam)
    {
        case TIMER_SEC:
            //每秒一次的处理
            break ;
        case TIMER_MIN:
            //每分钟一次的处理
            break ;
    }
return 0 ;
```

如果您想将一个已经存在的计时器设定为不同的时间间隔, 您可以简单地用不同的时间值再次呼叫 SetTimer。在时钟程式里, 如果显示秒或不显示秒是可以选择的, 您就可以这样做, 只需简单地将时间间隔在 1000 毫秒和 60 000 毫秒间切换就可以了。

程式 8-1 显示了一个使用计时器的简单程式, 名为 BEEPER1, 计时器的时间间隔设定为 1 秒。当它收到 WM_TIMER 讯息时, 它将显示区域的颜色由蓝色变为红色或由红色变为蓝色, 并通过呼叫 MessageBeep 函式发出响声。(虽然 MessageBeep 通常用於 MessageBox, 但它确实是一个全功能的鸣叫函式。在有音效卡的 PC 机上, 一般可以使用不同的 MB_ICON 参数作为 MessageBeep 的一个参数以用於 MessageBox, 来播放使用者在「控制台」的「声音」程式中选择的 不同声音)。

BEEPER1 在视窗讯息处理程式处理 WM_CREATE 讯息时设定计时器。在处理 WM_TIMER 讯息处理期间, BEEPER1 呼叫 MessageBeep, 转 bFlipFlop 的值并使视窗无效以产生 WM_PAINT 讯息。在处理 WM_PAINT 讯息处理期间, BEEPER1 通过呼叫 GetClientRect 获得视窗大小的 RECT 结构, 并通过呼叫 FillRect 改变视

窗的颜色。

程式 8-1 BEEPER1

```

BEEPER1.C
/*-----
    BEEPER1.C -- Timer Demo Program No. 1
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>

#define ID_TIMER    1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("Beeper1") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style              = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc        = WndProc ;
    wndclass.cbClsExtra          = 0 ;
    wndclass.cbWndExtra          = 0 ;
    wndclass.hInstance          = hInstance ;
    wndclass.hIcon               = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor             = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName        = NULL ;
    wndclass.lpszClassName       = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Beeper1 Timer Demo"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
}

```

```

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static BOOL      fFlipFlop = FALSE ;
    HBRUSH           hBrush ;
    HDC              hdc ;
    PAINTSTRUCT ps ;
    RECT             rc ;

    switch (message)
    {
    case WM_CREATE:
        SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
        return 0 ;

    case WM_TIMER :
        MessageBeep (-1) ;
        fFlipFlop = !fFlipFlop ;
        InvalidateRect (hwnd, NULL, FALSE) ;
        return 0 ;

    case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rc) ;
        hBrush = CreateSolidBrush (fFlipFlop ? RGB(255,0,0) :
RGB(0,0,255)) ;
        FillRect (hdc, &rc, hBrush) ;

        EndPaint (hwnd, &ps) ;
        DeleteObject (hBrush) ;
        return 0 ;

    case WM_DESTROY :
        KillTimer (hwnd, ID_TIMER) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

因为 BEEPER1 每次收到 WM_TIMER 讯息时，都用颜色的变换显示出来，所以您可以通过呼叫 BEEPER1 来查看 WM_TIMER 讯息的性质，并完成 Windows 内部的一些其他操作。

例如，首先呼叫 **控制台** 的 **显示器** 程式，选择 **效果**，确定 **拖曳时显示视窗内容** 核取方块没有被选中。现在，试著移动或者缩放 BEEPER1 视窗，这将导致程式进入「模态讯息回圈」。Windows 通过在内部讯息而非您程式的讯息回圈中拦截所有讯息，来禁止对移动或者缩放操作的任何干扰。通过此回圈到达程式视窗的大多数讯息都被丢弃，这就是 BEEPER1 停止蜂鸣的原因。当完成了移动与缩放之後，您将会注意到 BEEPER1 不能取得它所丢弃的所有 WM_TIMER 讯息，尽管前两个讯息的间隔可能少於 1 秒。

在「拖曳时显示视窗内容」核取方块被选中时，Windows 中，的模态讯息回圈会试图给您的视窗讯息处理程式传递一些丢失的讯息。这样做有时工作得很好，有时却不行。

方法二

设定计时器的第一种方法是把 WM_TIMER 讯息发送到通常的视窗讯息处理程式，而第二种方法是让 Windows 直接将计时器讯息发送给您程式的另一个函式。

接收这些计时器讯息的函式被称为「callback」函式，这是一个在您的程式之中但是由 Windows 呼叫的函式。您先告诉 Windows 此函式的位址，然後 Windows 呼叫此函式。这看起来也很熟悉，因为程式的视窗讯息处理程式实际上也是一种 callback 函式。当注册视窗类别时，要将函式的位址告诉 Windows，当发送讯息给程式时，Windows 会呼叫此函式。

SetTimer 并非是唯一使用 callback 函式的 Windows 函式。CreateDialog 和 DialogBox 函式（将在第十一章中介绍）使用 callback 函式处理对话方块中的讯息；有几个 Windows 函式（EnumChildWindow、EnumFonts、EnumObjects、EnumProps 和 EnumWindow）把列举资讯传递给 callback 函式；还有几个不那么常用的函式（GrayString、LineDDA 和 SetWindowHookEx）也要求 callback 函式。

像视窗讯息处理程式一样，callback 函式也必须定义为 CALLBACK，因为它是由 Windows 从程式的程式码段呼叫的。callback 函式的参数和 callback 函式的传回值取决於 callback 函式的目的。跟计时器有关的 callback 函式中，输入参数与视窗讯息处理程式的输入参数一样。计时器 callback 函式不向 Windows 传回值。

我们把以下的 callback 函式称为 TimerProc（您能够选择与其他一些用语

不会发生冲突的任何名称)，它只处理 WM_TIMER 讯息：

```
VOID CALLBACK TimerProc (    HWND hwnd, UINT message, UINT iTimerID, DWORD dwTime)
{
    处理 WM_TIMER 讯息
}
```

TimerProc 的参数 hwnd 是在呼叫 SetTimer 时指定的视窗代号。Windows 只把 WM_TIMER 讯息送给 TimerProc，因此讯息参数总是等於 WM_TIMER。iTimerID 值是计时器 ID，dwTimer 值是与从 GetTickCount 函式的传回值相容的值。这是自 Windows 启动後所经过的毫秒数。

在 BEEPER1 中已经看到过，用第一种方法设定计时器时要求下面格式的 SetTimer 呼叫：

```
SetTimer (hwnd, iTimerID, iMsecInterval, NULL) ;
```

您使用 callback 函式处理 WM_TIMER 讯息时，SetTimer 的第四个参数由 callback 函式的位址取代，如下所示：

```
SetTimer (hwnd, iTimerID, iMsecInterval, TimerProc) ;
```

我们来看看一些范例程式码，这样您就会了解这些东西是如何组合在一起的。在功能上，除了 Windows 发送一个计时器讯息给 TimerProc 而非 WndProc 之外，程式 8-2 所示的 BEEPER2 程式与 BEEPER1 是相同的。注意，TimerProc 和 WndProc 一起被宣告在程式的开始处。

程式 8-2 BEEPER2

```
BEEPER2.C
/*-----
-
    BEEPER2.C --      Timer Demo Program No. 2
                        (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
#define ID_TIMER      1

LRESULT      CALLBACK      WndProc      (HWND, UINT, WPARAM, LPARAM) ;
VOID      CALLBACK      TimerProc (HWND, UINT, UINT,      DWORD ) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static char szAppName[]      = "Beeper2" ;
    HWND      hwnd ;
    MSG      msg ;
    WNDCLASS      wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
```

```

        wndclass.lpfnWndProc          = WndProc ;
        wndclass.cbClsExtra           = 0 ;
        wndclass.cbWndExtra           = 0 ;
        wndclass.hInstance            = hInstance ;
        wndclass.hIcon                = LoadIcon (NULL,
IDI_APPLICATION) ;
        wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground        = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
        wndclass.lpszMenuName          = NULL ;
        wndclass.lpszClassName        = szAppName ;

        if (!RegisterClass (&wndclass))
        {
                MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
                                szAppName,
MB_ICONERROR) ;

                return 0 ;
        }

        hwnd = CreateWindow ( szAppName, "Beeper2 Timer Demo",
                                WS_OVERLAPPEDWINDOW,
                                CW_USEDEFAULT, CW_USEDEFAULT,
                                CW_USEDEFAULT, CW_USEDEFAULT,
                                NULL, NULL, hInstance, NULL) ;

        ShowWindow (hwnd, iCmdShow) ;
        UpdateWindow (hwnd) ;

        while (GetMessage (&msg, NULL, 0, 0))
        {
                TranslateMessage (&msg) ;
                DispatchMessage (&msg) ;
        }
        return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
        case WM_CREATE:
                SetTimer (hwnd, ID_TIMER, 1000, TimerProc) ;
                return 0 ;

        case WM_DESTROY:
                KillTimer (hwnd, ID_TIMER) ;
                PostQuitMessage (0) ;

```



```

        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

VOID CALLBACK TimerProc (HWND hwnd, UINT message, UINT iTimerID, DWORD dwTime)
{
    static BOOL fFlipFlop = FALSE ;
    HBRUSH          hBrush ;
    HDC              hdc ;
    RECT             rc ;

    MessageBeep (-1) ;
    fFlipFlop = !fFlipFlop ;

    GetClientRect (hwnd, &rc) ;
    hdc = GetDC (hwnd) ;
    hBrush = CreateSolidBrush (fFlipFlop ? RGB(255,0,0) : RGB(0,0,255)) ;

    FillRect (hdc, &rc, hBrush) ;
    ReleaseDC (hwnd, hdc) ;
    DeleteObject (hBrush) ;
}

```

方法三

设定计时器的第三种方法类似於第二种方法，只是传递给 SetTimer 的 hwnd 参数被设定为 NULL，并且第二个参数（通常为计时器 ID）被忽略了，最後，此函式传回计时器 ID：

```
iTimerID = SetTimer (NULL, 0, wParamInterval, TimerProc) ;
```

如果没有可用的计时器，那么从 SetTimer 传回的 iTimerID 值将为 NULL。

KillTimer 的第一个参数（通常是视窗代号）也必须为 NULL，计时器 ID 必须是 SetTimer 的传回值：

```
KillTimer (NULL, iTimerID) ;
```

传递给 TimerProc 计时器函式的 hwnd 参数也必须是 NULL。这种设定计时器的方法很少被使用。如果在您的程式在不同时刻有一系列的 SetTimer 呼叫，而又不希望追踪您已经用过了那些计时器 ID，那么使用此方法是很方便的。

既然您已经知道了如何使用 Windows 计时器，就可以开始讨论一些有用的计时器程式了。

计时器用於时钟

时钟是计时器最明显的应用，因此让我们来看看两个时钟，一个数位时钟，一个类比时钟。

建立数位时钟

程式 8-3 所示的 DIGCLOCK 程式，使用类似 LED 的 7 个显示方块显示了目前的时间。

程式 8-3 DIGCLOCK

```
DIGCLOCK.C
/*-----
--
--      DIGCLOCK.C --      Digital Clock
--                        (c) Charles Petzold, 1998
--
*/

#include <windows.h>
#define ID_TIMER      1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("DigClock") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
                    = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
                    = WndProc ;
    wndclass.cbClsExtra
                    = 0 ;
    wndclass.cbWndExtra
                    = 0 ;
    wndclass.hInstance
                    = hInstance ;
    wndclass.hIcon
                    = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor
                    = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
                    = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName
                    = NULL ;
    wndclass.lpszClassName
                    = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("Program requires Windows NT!"),
                        szAppName,
```

```

MB_ICONERROR) ;

        return 0 ;

    }

    hwnd = CreateWindow (  szAppName, TEXT ("Digital Clock"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void DisplayDigit (HDC hdc, int iNumber)
{
    static BOOL fSevenSegment [10][7] = {
        1, 1, 1, 0, 1, 1, 1, // 0
        0, 0, 1, 0, 0, 1, 0, // 1
        1, 0, 1, 1, 1, 0, 1, // 2
        1, 0, 1, 1, 0, 1, 1, // 3
        0, 1, 1, 1, 0, 1, 0, // 4
        1, 1, 0, 1, 0, 1, 1, // 5
        1, 1, 0, 1, 1, 1, 1, // 6
        1, 0, 1, 0, 0, 1, 0, // 7
        1, 1, 1, 1, 1, 1, 1, // 8
        1, 1, 1, 1, 0, 1, 1 } ; // 9

    static POINT ptSegment [7][6] = {
        7, 6, 11, 2, 31, 2, 35, 6, 31, 10, 11, 10,
        6, 7, 10, 11, 10, 31, 6, 35, 2, 31, 2, 11,
        36, 7, 40, 11, 40, 31, 36, 35, 32, 31, 32, 11,
        7, 36, 11, 32, 31, 32, 35, 36, 31, 40, 11, 40,
        6, 37, 10, 41, 10, 61, 6, 65, 2, 61, 2, 41,
        36, 37, 40, 41, 40, 61, 36, 65, 32, 61, 32, 41,
        7, 66, 11, 62, 31, 62, 35, 66, 31, 70, 11, 70 } ;

    int iSeg ;
    for (iSeg = 0 ; iSeg < 7 ; iSeg++)
        if (fSevenSegment [iNumber][iSeg])
            Polygon (hdc, ptSegment [iSeg], 6) ;
}

```

```

void DisplayTwoDigits (HDC hdc, int iNumber, BOOL fSuppress)
{
    if (!fSuppress || (iNumber / 10 != 0))
        DisplayDigit (hdc, iNumber / 10) ;
    OffsetWindowOrgEx (hdc, -42, 0, NULL) ;
    DisplayDigit (hdc, iNumber % 10) ;
    OffsetWindowOrgEx (hdc, -42, 0, NULL) ;
}

void DisplayColon (HDC hdc)
{
    POINT ptColon [2][4] = {
        2,    21,   6,    17,   10,   21,   6,
        25,  2,    51,   6,    47,   10,   51,   6,   55 } ;

    Polygon (hdc, ptColon [0], 4) ;
    Polygon (hdc, ptColon [1], 4) ;

    OffsetWindowOrgEx (hdc, -12, 0, NULL) ;
}

void DisplayTime (HDC hdc, BOOL f24Hour, BOOL fSuppress)
{
    SYSTEMTIME st ;
    GetLocalTime (&st) ;
    if (f24Hour)
        DisplayTwoDigits (hdc, st.wHour, fSuppress) ;
    else
        DisplayTwoDigits (hdc, (st.wHour % 12) ? st.wHour : 12, fSuppress) ;
    DisplayColon (hdc) ;
    DisplayTwoDigits (hdc, st.wMinute, FALSE) ;
    DisplayColon (hdc) ;
    DisplayTwoDigits (hdc, st.wSecond, FALSE) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam, LPARAM lParam)
{
    static BOOL          f24Hour, fSuppress ;
    static HBRUSH        hBrushRed ;
    static int           cxClient, cyClient ;
    HDC                  hdc ;
    PAINTSTRUCT ps ;
    TCHAR                szBuffer [2] ;

    switch (message)
    {
    case WM_CREATE:
        hBrushRed = CreateSolidBrush (RGB (255, 0, 0)) ;
        SetTimer (hwnd, ID_TIMER, 1000, NULL) ; // fall through
    }
}

```

```
case WM_SETTINGCHANGE:
    GetLocaleInfo (LOCALE_USER_DEFAULT, LOCALE_ITIME, szBuffer, 2) ;
    f24Hour = (szBuffer[0] == '1') ;

    GetLocaleInfo (LOCALE_USER_DEFAULT, LOCALE_ITLZERO, szBuffer, 2) ;
    fSuppress = (szBuffer[0] == '0') ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_TIMER:
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetMapMode (hdc, MM_ISOTROPIC) ;
    SetWindowExtEx (hdc, 276, 72, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

    SetWindowOrgEx (hdc, 138, 36, NULL) ;
    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
    SelectObject (hdc, GetStockObject (NULL_PEN)) ;
    SelectObject (hdc, hBrushRed) ;

    DisplayTime (hdc, f24Hour, fSuppress) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hBrushRed) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIGCLOCK 视窗如图 8-1 所示。



图 8-1 DIGCLOCK 的萤幕显示

虽然，在图 8-1 中您看不到时钟的数字是红色的。DIGCLOCK 的视窗讯息处理程式在处理 WM_CREATE 讯息处理期间建立了一个红色的画刷并在处理 WM_DESTROY 讯息处理期间清除它。WM_CREATE 讯息也为 DIGCLOCK 设定了一个一秒的计时器，该计时器在处理 WM_DESTROY 讯息处理期间被终止（待会将讨论对 GetLocaleInfo 的呼叫）。

在收到 WM_TIMER 讯息後，DIGCLOCK 的视窗程序呼叫 InvalidateRect 简单地使整个视窗无效。这不是最佳方法，因为每秒整个视窗都要被擦除和重画，有时会引起显示器的闪烁。依据目前的时间使视窗需要更新的部分无效是最好的解决方法。然而，在逻辑上这样做的确很复杂。

在处理 WM_TIMER 讯息处理期间使视窗无效会迫使所有程式的真正活动转入 WM_PAINT。DIGCLOCK 在 WM_PAINT 讯息一开始将映射方式设定为 MM_ISOTROPIC。这样，DIGCLOCK 将使用水平方向和垂直方向相等的轴。这些轴（由 SetWindowExtEx 呼叫设定）是水平 276 个单位，垂直 72 个单位。当然，这些轴定得有点太随意了，但它们是按照时钟数位元的大小和间距安排的。

DIGCLOCK 将视窗原点设定为 (138, 36)，这是视窗范围的中心；将视埠原点设定为 (cxClient / 2, cyClient / 2)。这意味著时钟的显示位於 DIGCLOCK 显示区域的中心，但是该 DIGCLOCK 也可以使用在显示幕左上角的原点 (0, 0) 的轴。

然後 WM_PAINT 将目前画刷设定为之前建立的红画刷，将目前画笔设定为

NULL_PEN, 并呼叫 DIGCLOCK 中的函式 DisplayTime。

取得目前时间

DisplayTime 函式开始呼叫 Windows 函式 GetLocalTime, 它带有一个的 SYSTEMTIME 结构的参数, 在 WINBASE.H 中定义为:

```
typedef struct _SYSTEMTIME
{
    WORD    wYear ;
    WORD    wMonth ;
    WORD    wDayOfWeek ;
    WORD    wDay ;
    WORD    wHour ;
    WORD    wMinute ;
    WORD    wSecond ;
    WORD    wMilliseconds ;
}
SYSTEMTIME, * PSYSTEMTIME ;
```

很明显, SYSTEMTIME 结构包含日期和时间。月份由 1 开始递增 (也就是说, 一月是 1), 星期由 0 开始递增 (星期天是 0)。wDay 成员是本月目前的日子, 也是由 1 开始递增的。

SYSTEMTIME 主要用於 GetLocalTime 和 GetSystemTime 函式。GetSystemTime 函式传回目前的世界时间 (Coordinated Universal Time, UTC), 大概与英国格林威治时间相同。GetLocalTime 函式传回当地时间, 依据电脑所在的时区。这些值的精确度完全决定於使用者所调整的时间精确度以及是否指定了正确的时区。可以双击工作列的时间显示来检查电脑上的时区设定。第二十三章会有一个程式, 能够通过 Internet 精确地设定时间。

Windows 还有 SetLocalTime 和 SetSystemTime 函式, 以及在/Platform SDK/Windows Base Services/General Library/Time 中说明的其他与时间有关的函式。

显示数字和冒号

如果 DIGCLOCK 使用一种模拟 7 段显示的字体将会简单一些。否则, 它就得使用 Polygon 函式做所有的工作。

DIGCLOCK 中的 DisplayDigit 函式定义了两个阵列。fSevenSegment 阵列有 7 个 BOOL 值, 用於从 0 到 9 的每个十进位数字。这些值指出了哪一段需要显示 (为 1), 哪一段不需要显示 (为 0)。在这个阵列中, 7 段由上到下、由左到右排序。7 段中的每个段都是一个 6 边的多边形。ptSegment 阵列是一个 POINT

结构的阵列，指出了 7 个段中每个点的图形座标。每个数字由下列程式码画出：

```
for (iSeg = 0 ; iSeg < 7 ; iSeg++)  
    if ( fSevenSegment [iNumber][iSeg])  
        Polygon (hdc, ptSegment [iSeg], 6) ;
```

类似地（但更简单），DisplayColon 函式在小时与分钟、分钟与秒之间画一个冒号。数字是 42 个单位宽，冒号是 12 个单位宽，因此 6 个数字与 2 个冒号，总宽度是 276 个单位，SetWindowExtEx 呼叫中使用了这个大小。

回到 DisplayTime 函式，原点位於最左数字位置的左上角。DisplayTime 呼叫 DisplayTwoDigits，DisplayTwoDigits 呼叫 DisplayDigit 两次，并且在每次呼叫 OffsetWindowOrgEx 後，将视窗原点向右移动 42 个单位。类似地，DisplayColon 函式在画完冒号後，将视窗原点向右移动 12 个单位。用这种方法，不管物件出现在视窗内的哪个地方，函式对数字和冒号都使用同样的座标。

这个程式的其他技巧是以 12 小时或 24 小时的格式显示时间以及当最左边的小时数字为 0 时不显示它。

国际化

尽管像 DIGCLOCK 这样显示时间是非常简单的，但是要显示复杂的日期和时间还是要依赖 Windows 的国际化支援。格式化日期和时间的最简单的方法是呼叫 GetDateFormat 和 GetTimeFormat 函式。这些函式在 /Platform SDK/Windows Base Services/General Library/String Manipulation/String Manipulation Reference/String Manipulation Functions 中有记载，但是它们在 /Platform SDK/Windows Base Services/International Features/National Language Support 中进行了说明。这些函式接受 SYSTEMTIME 结构并且依据使用者在「控制台」的「区域设定」程式中所做的选择而将日期和时间格式化。

DIGCLOCK 不能使用 GetDateFormat 函式，因为它只知道显示数字和冒号，然而，DIGCLOCK 应该能够根据使用者的参数选择来显示 12 小时或 24 小时的格式，并禁止（或不禁止）开头的小时数字。您可以从 GetLocaleInfo 函式中取得这种资讯。虽然 GetLocaleInfo 在 /Platform SDK/Windows Base Services/General Library/String Manipulation/String Manipulation Reference/String Manipulation Functions 中有记载，但是这个函式使用的识别字在 /Platform SDK/Windows Base Services/International Features/National Language Support/National Language Support Constants 中有说明。

DIGCLOCK 在处理 WM_CREATE 讯息时，最初呼叫 GetLocaleInfo 两次，第一次使用 LOCALE_ITIME 识别字（确定使用的是 12 小时还是 24 小时格式），然後

使用 LOCALE_ITLZERO 识别字（在小时显示中禁止前面显示 0）。GetLocaleInfo 函式在字串中传回所有的资讯，但是在大多数情况下把字串转变为整数并不是非常容易。DIGCLOCK 把字串储存在两个静态变数中并把它们传递给 DisplayTime 函式。

如果使用者更改了任何系统设定，则会将 WM_SETTINGCHANGE 讯息传送给所有的應用程式。DIGCLOCK 通过再次呼叫 GetLocaleInfo 处理这个讯息。以这种方式，您可以在「控制台」的「区域设定」程式中进行不同的设定来实验一下。

在理论上，DIGCLOCK 也应该使用 LOCALE_STIME 识别字呼叫 GetLocaleInfo。这会传回使用者为时间的小时、分钟和秒等单个部分选择的字元。因为 DIGCLOCK 被设定为仅显示冒号，所以不管选择了什么，都会得到冒号。要指出时间是 A. M. 或 P. M.，應用程式可以使用带有 LOCALE_S1159 和 LOCALE_S2359 识别字的 GetLocaleInfo 函式。这些识别字使程式获得适合於使用者国家/地区和语言的字串。

我们也可以让 DIGCLOCK 处理 WM_TIMECHANGE 讯息，这样它将系统时间与日期发生变化的讯息通知應用程式。DIGCLOCK 因 WM_TIMER 讯息而每秒更新一次，实际上没有必要这样作，对 WM_TIMECHANGE 讯息的处理使得每分钟更新一次的时钟变得更为合理。

建立类比时钟

类比时钟不必关心国际化问题，但是由於图形所引起的复杂性却抵消了这种简化。为了正确地产生时钟，您需要知道一些三角函数。CLOCK 如程式 8-4 所示。

程式 8-4 CLOCK

```
CLOCK.C
/*-----
-
CLOCK.C --      Analog Clock Program
                  (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <math.h>

#define ID_TIMER          1
#define TWOPI              (2 * 3.14159)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```

                                PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Clock") ;
    HWND              hwnd;
    MSG               msg;
    WNDCLASS          wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = NULL ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground =          (HBRUSH)      GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (          NULL, TEXT ("Program requires Windows NT!"),
                                szAppName,
MB_ICONERROR) ;
        return 0 ;
    }
    hwnd = CreateWindow (  szAppName, TEXT ("Analog Clock"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void SetIsotropic (HDC hdc, int cxClient, int cyClient)
{
    SetMapMode (hdc, MM_ISOTROPIC) ;
    SetWindowExtEx (hdc, 1000, 1000, NULL) ;

```

```
    SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
}

void RotatePoint (POINT pt[], int iNum, int iAngle)
{
    int i ;
    POINT ptTemp ;

    for (i = 0 ; i < iNum ; i++)
    {
        ptTemp.x = (int) (pt[i].x * cos (TWOPI * iAngle / 360) +
                           pt[i].y * sin (TWOPI * iAngle / 360)) ;

        ptTemp.y = (int) (pt[i].y * cos (TWOPI * iAngle / 360) -
                           pt[i].x * sin (TWOPI * iAngle / 360)) ;

        pt[i] = ptTemp ;
    }
}

void DrawClock (HDC hdc)
{
    int iAngle ;
    POINT pt[3] ;
    for (iAngle = 0 ; iAngle < 360 ; iAngle += 6)
    {
        pt[0].x = 0 ;
        pt[0].y = 900 ;

        RotatePoint (pt, 1, iAngle) ;

        pt[2].x = pt[2].y = iAngle % 5 ? 33 : 100 ;

        pt[0].x - = pt[2].x / 2 ;
        pt[0].y - = pt[2].y / 2 ;

        pt[1].x = pt[0].x + pt[2].x ;
        pt[1].y = pt[0].y + pt[2].y ;

        SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;

        Ellipse (hdc, pt[0].x, pt[0].y, pt[1].x, pt[1].y) ;
    }
}

void DrawHands (HDC hdc, SYSTEMTIME * pst, BOOL fChange)
{

```

```

static POINT pt[3][5] = {0, -150, 100, 0, 0, 600, -100, 0, 0, -150,
                          0, -200,      50,  0, 0, 800, -50, 0, 0, -200,
                          0, 0,      0,  0, 0, 0, 0, 0, 0, 800 } ;

int          i, iAngle[3] ;
POINT        ptTemp[3][5] ;

iAngle[0]    = (pst->wHour * 30) % 360 + pst->wMinute / 2 ;
iAngle[1]    = pst->wMinute * 6 ;
iAngle[2]    = pst->wSecond * 6 ;

memcpy (ptTemp, pt, sizeof (pt)) ;
for (i = fChange ? 0 : 2 ; i < 3 ; i++)
{
    RotatePoint (ptTemp[i], 5, iAngle[i]) ;
    Polyline (hdc, ptTemp[i], 5) ;
}
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam, LPARAM
lParam)
{
    static int          cxClient, cyClient ;
    static SYSTEMTIME  stPrevious ;
    BOOL                fChange ;
    HDC                 hdc ;
    PAINTSTRUCT         ps ;
    SYSTEMTIME          st ;

    switch (message)
    {
    case  WM_CREATE :
        SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
        GetLocalTime (&st) ;
        stPrevious = st ;
        return 0 ;

    case  WM_SIZE :
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case  WM_TIMER :
        GetLocalTime (&st) ;

        fChange =  st.wHour      !  = stPrevious.wHour ||
                   st.wMinute !  =
stPrevious.wMinute ;

```

```
        hdc = GetDC (hwnd) ;

        SetIsotropic (hdc, cxClient, cyClient) ;

        SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
        DrawHands (hdc, &stPrevious, fChange) ;

        SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
        DrawHands (hdc, &st, TRUE) ;

        ReleaseDC (hwnd, hdc) ;

        stPrevious = st ;
        return 0 ;

case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;

        SetIsotropic (hdc, cxClient, cyClient) ;
        DrawClock (hdc) ;
        DrawHands (hdc, &stPrevious, TRUE) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY :
        KillTimer (hwnd, ID_TIMER) ;
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CLOCK 萤幕显示如图 8-2。

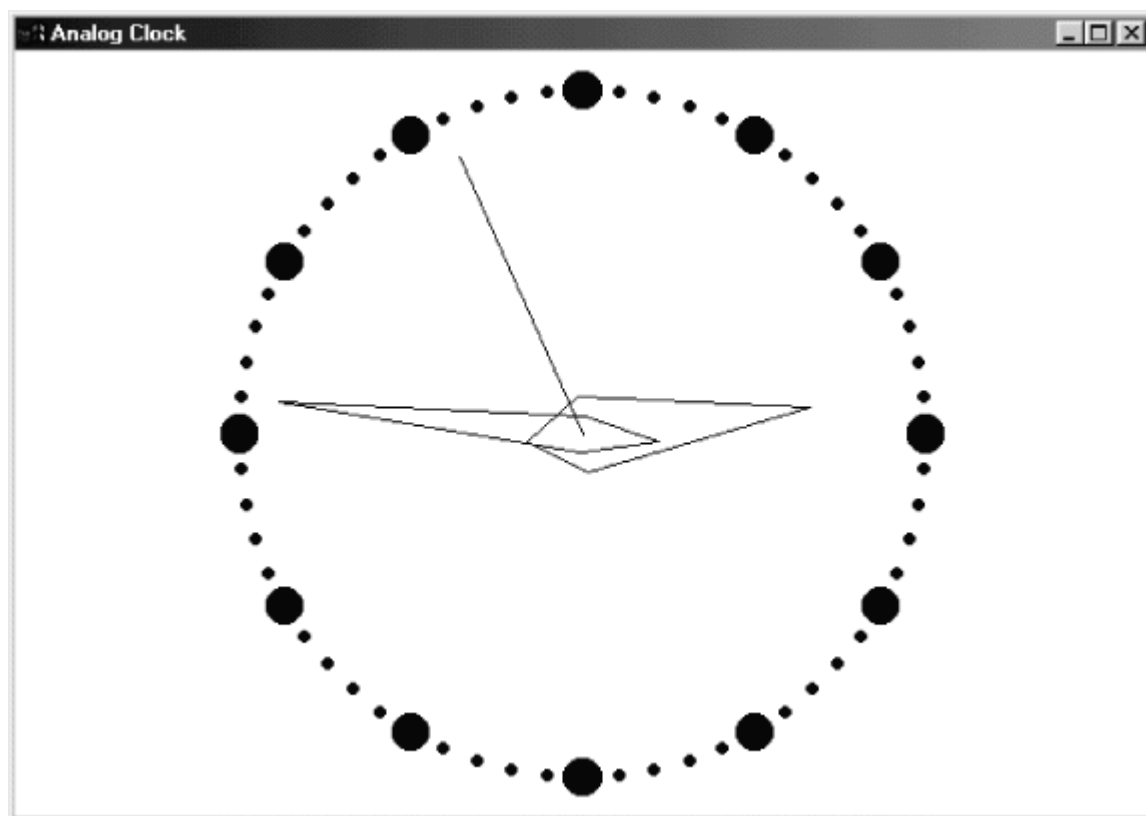


图 8-2 CLOCK 的萤幕显示

等方向性(isotropic)映射对于这样的应用来说是理想的, CLOCK.C 中的 SetIsotropic 函式负责设定此模式。在呼叫 SetMapMode 之後, SetIsotropic 将视窗范围设定为 1000, 并将视埠范围设定为显示区域的一半宽度和显示区域的负的一半高度。视埠原点被设定为显示区域的中心。我在第五章中讨论过, 这将建立一个笛卡儿坐标系, 其点(0, 0)位于显示区域的中心, 在所有方向上的范围都是 1000。

RotatePoint 函式是用到三角函数的地方, 此函式的三个参数分别是一个或者多个点的阵列、阵列中点的个数以及以度为单位的旋转角度。函式以原点为中心按顺时针方向(这对一个时钟正合适)旋转这些点。例如, 如果传给函式的点是(0, 100)——即 12:00 的位置——而角度为 90 度, 那么该点将被变换为(100, 0)——即 3:00。它使用下列公式来做到这一点:

```
x' = x * cos (a) + y * sin (a)
y' = y * cos (a) - x * sin (a)
```

RotatePoint 函式在绘制时钟表面的点和表针时都是有用的, 我们将马上看到这一点。

DrawClock 函式绘制 60 个时钟表面的点, 从顶部(12:00)开始, 其中每个点离原点 900 单位, 因此第一个点位于(0, 900), 此後的每个点按顺时针依次增加 6 度。这些点中的 12 个直径为 100 个单位; 其余的为 33 个单位。使用 Ellipse 函式来画点。

DrawHands 函式绘制时钟的时针、分针和秒针。定义表针轮廓（当它们垂直向上时的形状）的座标存放在一个 POINT 结构的阵列中。根据时间，这些座标使用 RotatePoint 函式进行旋转，并用 Windows 的 Polyline 函式进行显示。注意时针和分针只有当传递给 DrawHands 的 bChange 参数为 TRUE 时才被显示。当程式更新时钟的表针时，大多数情况下时针和分针不需要重画。

现在让我们将注意力转到视窗讯息处理程式。在 WM_CREATE 讯息处理期间，视窗讯息处理程式取得目前时间并将它存放在名为 dtPrevious 的变数中，这个变数将在以後被用於确定时针或者分针从上次更新以来是否改变过。

第一次绘制时钟是在第一个 WM_PAINT 讯息处理期间，这只不过是依次呼叫 SetIsotropic、DrawClock 和 DrawHands，後者的 bChange 参数被设定为 TRUE。

在 WM_TIMER 讯息处理期间，WndProc 首先取得新的时间并确定是否需要重新绘制时针和分针。如果需要，则使用一个白色画笔和上一次时间绘制所有的表针，从而有效地擦除它们。否则，只对秒针使用白色画笔进行擦除，然後，再使用一个黑色画笔绘制所有的表针。

以计时器进行状态报告

本章的最後一个程式是我在第五章提到过的。它是一个使用 GetPixel 函式的好例子。

WHATCLR （见程式 8-5）显示了滑鼠游标下目前图素的 RGB 颜色。

程式 8-5 WHATCLR

```
WHATCLR.C
/*-----
    WHATCLR.C -- Displays Color Under Cursor
                (c) Charles Petzold, 1998
    -----*/
/

#include <windows.h>
#define ID_TIMER    1
void FindWindowSize (int *, int *) ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("WhatClr") ;
    HWND              hwnd ;
    int                cxWindow, cyWindow ;
    MSG                msg ;
    WNDCLASS            wndclass ;
```

```

    wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc          = WndProc ;
    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance            = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    FindWindowSize (&cxWindow, &cyWindow) ;
    hwnd = CreateWindow (szAppName, TEXT ("What Color"),
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_BORDER,
        CW_USEDEFAULT, CW_USEDEFAULT,
        cxWindow, cyWindow,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void FindWindowSize (int * pcxWindow, int * pcyWindow)
{
    HDC                hdcScreen ;
    TEXTMETRIC tm ;

    hdcScreen = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
    GetTextMetrics (hdcScreen, &tm) ;
    DeleteDC (hdcScreen) ;

```



```
* pcxWindow = 2 *      GetSystemMetrics (SM_CXBORDER)  +
                        12 * tm.tmAveCharWidth ;
* pcyWindow = 2 *      GetSystemMetrics (SM_CYBORDER)  +
                        GetSystemMetrics (SM_CYCAPTION) +
                        2 * tm.tmHeight ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static COLORREF      cr, crLast ;
    static HDC           hdcScreen ;
    HDC                  hdc ;
    PAINTSTRUCT          ps ;
    POINT                pt ;
    RECT                 rc ;
    TCHAR                szBuffer [16] ;

    switch (message)
    {
    case WM_CREATE:
        hdcScreen = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
        SetTimer (hwnd, ID_TIMER, 100, NULL) ;
        return 0 ;

    case WM_TIMER:
        GetCursorPos (&pt) ;
        cr = GetPixel (hdcScreen, pt.x, pt.y) ;

        SetPixel (hdcScreen, pt.x, pt.y, 0) ;

        if (cr != crLast)
        {
            crLast = cr ;
            InvalidateRect (hwnd, NULL, FALSE) ;
        }
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rc) ;

        wsprintf (szBuffer, TEXT (" %02X %02X %02X "),
            GetRValue (cr), GetGValue (cr), GetBValue (cr)) ;

        DrawText (hdc, szBuffer, -1, &rc,
            DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
```

```
        EndPaint (hwnd, &ps) ;  
        return 0 ;  
  
    case WM_DESTROY:  
        DeleteDC (hdcScreen) ;  
        KillTimer (hwnd, ID_TIMER) ;  
        PostQuitMessage (0) ;  
        return 0 ;  
    }  
    return DefWindowProc (hwnd, message, wParam, lParam) ;  
}
```

WHATCLR 在 WinMain 中做了一点与以往不同的事。因为 WHATCLR 的视窗只需要显示十六进位 RGB 值那么大，所以它在 CreateWindow 函式中使用 WS_BORDER 视窗样式建立了一个不能改变大小的视窗。要计算视窗的大小，WHATCLR 通过先呼叫 CreateIC 再呼叫 GetSystemMetrics 以取得用於视讯显示的装置内容资讯。计算好的视窗宽度和高度值被传递给 CreateWindow。

WHATCLR 的视窗讯息处理程式在处理 WM_CREATE 讯息处理期间，呼叫 CreateDC 建立了用於整个视讯显示的装置内容。这个装置内容在程式的生命周期内都有效。在处理 WM_TIMER 讯息处理期间，程式取得目前滑鼠游标位置的图素。在处理 WM_PAINT 讯息处理期间显示 RGB 颜色。

您可能想知道，从 CreateDC 函式中取得的装置内容代号是否能让您在萤幕的任意位置显示一些东西，而不光只是取得图素颜色。答案是可以的，一般而言，让一个应用程式在另一个程式控制的画面区域上画图是不好的，但在某些特殊情况下，这可能会非常有用。

第九章 子视窗控制项

回忆第七章的 CHECKER 程式。这些程式显示了矩形网格。当您在一个矩形中按下鼠标按键时,该程式就画一个 x;如果您再按一次鼠标按键,那么 x 就消失。虽然这个程式的 CHECKER1 和 CHECKER2 版本只使用一个主视窗,但 CHECKER3 版本却为每个矩形使用一个子视窗。这些矩形由一个叫做 ChildProc 的独立视窗讯息处理程式维护。

如果有必要,无论矩形是否被选中,都可以给 ChildProc 增加一种向其父视窗讯息处理程式(WndProc)发送讯息的手段。通过呼叫 GetParent,子视窗讯息处理程式能确定其父视窗的视窗代号:

```
hwndParent = GetParent (hwnd) ;
```

其中, hwnd 是子视窗的视窗代号。它可以向其父视窗讯息处理程式发送讯息:

```
SendMessage (hwndParent, message, wParam, lParam) ;
```

那么 message 应该设定为什么呢?您可以随意地设定,数值大小可以与 WM_USER 相同或更大,这些数字代表和预先定义的 WM_ 讯息不冲突的讯息。也许对这个讯息,子视窗可以将 wParam 设定为它的子视窗 ID。如果在该子视窗单击,那么 lParam 可以被设为 1;如果未在该子视窗上单击,那么 lParam 将被设为 0。这是处理方式的一种选择。

事实上,这是在建立一个「子视窗控制项」。当子视窗的状态改变时,子视窗处理鼠标和键盘讯息并通知父视窗。使用这种方法,子视窗就变成了其父视窗的高阶输入装置。它将与自己在萤幕上的图形外观相应的处理,对使用者输入的回应以及在发生重要的输入事件时通知另一个视窗的方法给封装起来。

虽然您可以建立自己的子视窗控制项,但是也可以利用一些预先定义的视窗类别(和视窗讯息处理程式)来建立标准的子视窗控制项,您一定在别的 Windows 程式中看到过这些控制项。这些控制项采用的形式有:按钮、核取方块、编辑方块、清单方块、下拉式清单方块、字串标签和卷动列。例如,如果想在您的试算表程式的某个角落放置一个标有「Recalculate」的按钮,那么您可以通过呼叫 CreateWindow 来建立这个按钮。您不必担心鼠标操作、按钮显示操作或按下该按钮时的自动闪烁操作,这些是由 Windows 内部完成的。您所要做的只是拦截 WM_COMMAND 讯息——当按钮被按下时,它通过这一讯息通知您的视窗讯息处理程式。真的这样简单吗?是的,一点也没错。

子视窗控制项在对话方块中最常用。在第十一章中您将会看到,子视窗控制项的位置和尺寸,是在范例程式的资源描述叙述中的对话方块模板里定义的。

但是，您也可以使用预先定义的，在普通视窗显示区域里的子视窗控制项。您可以呼叫一次 `CreateWindow` 来建立一个子视窗，并通过呼叫 `MoveWindow` 来调整子视窗的位置和尺寸。父视窗讯息处理程式向子视窗控制项发送讯息，子视窗控制项向父视窗讯息处理程式传回讯息。

在建立普通视窗时，首先定义视窗类别，并使用 `RegisterClass` 将其注册到 Windows 中，然後用 `CreateWindow` 命令依据该视窗类别建立一个普通视窗，从第三章开始，我们就是这么做的。但是，当您使用预先定义的某个控制项时，不必为子视窗注册视窗类别，视窗类别已经存在於 Windows 之中，并且有一个预先定义的名字。您只需在 `CreateWindow` 中把它们用作视窗类别参数。`CreateWindow` 中的视窗样式参数准确地定义了子视窗控制项的外形和功能。Windows 内建了处理发送给依据这些视窗类别建立的子视窗讯息的视窗讯息处理程式。

直接在您的视窗上使用子视窗控制项完成某些任务，这些任务的层次低於在对话方块中使用子视窗控制项所要求的层次。这里，对话方块管理器在您的程式和控制项之间增加一个隔离层。值得一提的，您可能会发现在您的视窗上建立的子视窗控制项，没有利用 `Tab` 键或方向键将输入焦点从一个控制项移动到另一个控制项的内部功能。子视窗控制项能够获得输入焦点，但是获得後，它将不能把输入焦点传回给父视窗。这就是本章要解决的问题。

Windows 程式设计的文件在两个地方讨论了子视窗控制项：首先是，简单的常用控制项，我们可以在 `/Platform SDK/User Interface Services/Controls` 的文件所描述的无数对话方块中看到。这些子视窗包括按钮（其中包括核取方块的单选按钮）、静态控制项（例如文字标签）、编辑方块（您可以在此编辑一行或多行文字）、卷动列、清单方块和下拉式清单方块。除下拉式清单方块以外，在 Windows 1.0 中就包括了这些控制项。这部分的 Windows 文件还包括 Rich Text 文字编辑控制项，它与编辑方块相似，但还允许编辑不同字体与样式的格式化文字，以及桌面应用工具列。

相对於「常用控制项」，还有一些神秘的特殊控制项。这些控制项在 `/Platform SDK/User Interface Services/Shell and Common Controls/Common Controls` 描述。本章不讨论常用控制项，但它们将出现在本书的其他部分。在这部分的 Windows 文件中，很容易找到您想从别的 Windows 應用程式中应用到您自己的應用程式里头那些部分资讯。

按钮类别

下面我们将通过叫做 `BTNL00K`（「button look」）的程式来开始介绍按钮

视窗类别，如程式 9-1 所示。BTNLOOK 建立 10 个子视窗按钮控制项，每个控制项对应一个标准的按钮样式，因此共有 10 种标准按钮样式。

程式 9-1 BTNLOOK

```

BTNLOOK.C
/*-----
    BTNLOOK.C --      Button Look Program
                        (c) Charles Petzold, 1998
-----*/

/

#include <windows.h>
struct
{
    int            iStyle ;
    TCHAR *        szText ;
}
button[] =
{
    BS_PUSHBUTTON,      TEXT ("PUSHBUTTON"),
    BS_DEFPUSHBUTTON,   TEXT ("DEFPUSHBUTTON"),
    BS_CHECKBOX,        TEXT ("CHECKBOX"),
    BS_AUTOCHECKBOX,    TEXT ("AUTOCHECKBOX"),
    BS_RADIOBUTTON,     TEXT ("RADIOBUTTON"),
    BS_3STATE,          TEXT ("3STATE"),
    BS_AUTO3STATE,      TEXT ("AUTO3STATE"),
    BS_GROUPBOX,        TEXT ("GROUPBOX"),
    BS_AUTORADIOBUTTON, TEXT ("AUTORADIO"),
    BS_OWNERDRAW,       TEXT ("OWNERDRAW")
} ;

#define NUM (sizeof button / sizeof button[0])
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("BtnLook") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL,
    IDI_APPLICATION) ;

```

```

    wndclass.hCursor          = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground    = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName      = NULL ;
    wndclass.lpszClassName     = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Button Look"),
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND      hwndButton[NUM] ;
    static RECT      rect ;
    static TCHAR     szTop[]          = TEXT ("message  wParam  lParam"),
                    szUnd[]          = TEXT ("_____"),
                    szFormat[] = TEXT ("% -16s%04X-%04X  %04X-%04X"),
                    szBuffer[50] ;
    static int       cxChar, cyChar ;
    HDC              hdc ;
    PAINTSTRUCT      ps ;
    int              i ;

    switch (message)
    {
    case WM_CREATE :
        cxChar = LOWORD (GetDialogBaseUnits ()) ;

```

```

        cyChar = HIWORD (GetDialogBaseUnits ()) ;

        for (i = 0 ; i < NUM ; i++)
            hwndButton[i] = CreateWindow
( TEXT("button"),button[i].szText,
  WS_CHILD | WS_VISIBLE | button[i].iStyle,
  cxChar, cyChar * (1 + 2 * i),
  20 * cxChar, 7 * cyChar / 4,
  hwnd, (HMENU) i,
  ((LPCREATESTRUCT) lParam)->hInstance, NULL) ;
        return 0 ;

case WM_SIZE :
    rect.left          = 24 * cxChar ;
    rect.top           = 2 * cyChar ;
    rect.right         = LOWORD (lParam) ;
    rect.bottom        = HIWORD (lParam) ;
    return 0 ;
case WM_PAINT :
    InvalidateRect (hwnd, &rect, TRUE) ;

    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    SetBkMode (hdc, TRANSPARENT) ;

    TextOut (hdc, 24 * cxChar, cyChar, szTop, lstrlen (szTop)) ;
    TextOut (hdc, 24 * cxChar, cyChar, szUnd, lstrlen (szUnd)) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DRAWITEM :
case WM_COMMAND :
    ScrollWindow (hwnd, 0, -cyChar, &rect, &rect) ;

    hdc = GetDC (hwnd) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

    TextOut(    hdc, 24 * cxChar, cyChar * (rect.bottom / cyChar
- 1),

                szBuffer,
                wsprintf (szBuffer, szFormat,
                message == WM_DRAWITEM ?

TEXT ("WM_DRAWITEM") :

                TEXT ("WM_COMMAND"),
                HIWORD (wParam), LOWORD (wParam) ,
                HIWORD    (lParam),    LOWORD
(lParam))) ;

```

```

        ReleaseDC (hwnd, hdc) ;
        ValidateRect (hwnd, &rect) ;
        break ;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
    }

    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

单击按钮时，按钮就给父视窗信息处理程式发送一个 WM_COMMAND 讯息，也就是我们所熟悉的 WndProc。BTNLOOK 的 WndProc 将该讯息的 wParam 参数和 lParam 参数显示在显示区域的右边，如图 9-1 所示。

具有 BS_OWNERDRAW 样式的按钮在视窗上显示为一个背景阴影，因为这种样式的按钮是由程式来负责绘制的。该按钮表示它需要由包含 lParam 讯息参数的 WM_DRAWITEM 讯息来绘制，而 lParam 讯息参数是一个指向 DRAWITEMSTRUCT 型态结构的指标。在 BTNLOOK 中，这些讯息也同样被显示。我将在本章的后面更详细地讨论这种拥有者绘制 (owner draw) 按钮。

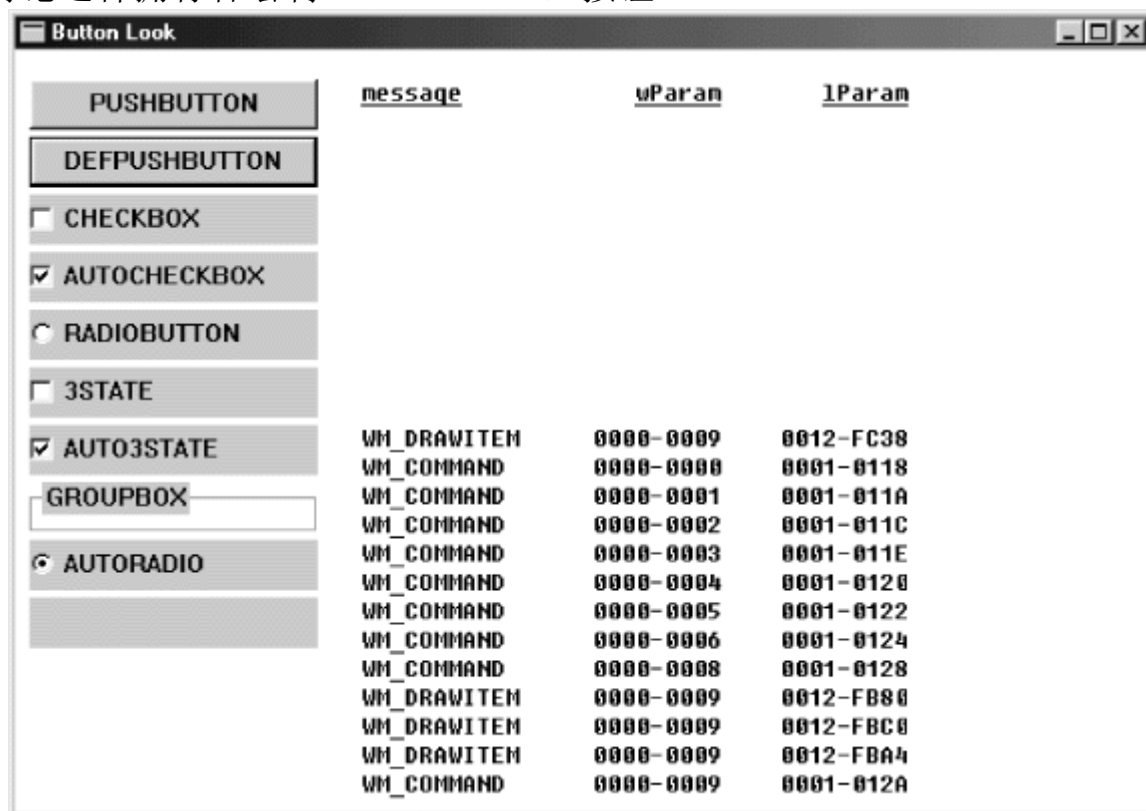


图 9-1 BTNLOOK 的萤幕显示

建立子视窗

BTNLOOK 定义了一个叫做 button 的结构，它包括了按钮视窗样式和描述性

字串，它们对应於 10 个按钮型态，所有按钮视窗样式都以字母「BS」开头，它表示「按钮样式」。10 个按钮子视窗是在 WndProc 中处理 WM_CREATE 讯息的过程中使用一个 for 回圈建立的。CreateWindow 呼叫使用下面这些参数：

Class name (类别名称)	TEXT ("button")
Window text (视窗文字)	button[i].szText
Window style (视窗样式)	WS_CHILD WS_VISIBLE button[i].iStyle
x position (x 位置)	cxChar
y position (y 位置)	cyChar * (1 + 2 * i)
Width (宽度)	20 * xChar
Height (高度)	7 * yChar / 4
Parent window (父视窗)	hwnd
Child window ID (子视窗 ID)	(HMENU) i
Instance handle (执行实体代号)	((LPCREATESTRUCT) lParam) -> hInstance
Extra parameters (附加参数)	NULL

类别名称参数是预先定义的名字。视窗样式使用 WS_CHILD、WS_VISIBLE 以及在 button 结构中定义的 10 个按钮样式之一 (BS_PUSHBUTTON、BS_DEFPUSHBUTTON 等等)。视窗文字参数 (对於普通视窗来说，它是显示在标题列中的文字) 将在每个按钮上显示出来。我简单地使用标识按钮样式文字的 x 位置和 y 位置参数，说明子视窗左上角相对於父视窗显示区域左上角的位置。宽度和高度参数规定了每个子视窗的宽度和高度。请注意，我用的是 GetDialogBaseUnits 函式来获得内定字体字元的宽度和高度。这是对话方块用来获得文字尺寸的函式。此函式传回一个 32 位元的值，其中低字组表示宽度，高字组表示高度。由於 GetDialogBaseUnits 传回的值与从 GetTextMetrics 获得的值大致上相同，但 GetDialogBaseUnits 有时使用起来会更方便些，而且能够与对话方块控制项更好地保持一致。

对每个子视窗，它的子视窗 ID 参数应该各不相同。在处理来自子视窗的 WM_COMMAND 讯息时，ID 帮助您视窗讯息处理程式识别出相应的子视窗。注意子视窗 ID 是作为 CreateWindow 的一个参数传递的，该参数通常用於指定程式的功能表，因此子视窗 ID 必须被强制转换为 HMENU。

CreateWindow 呼叫的执行实体代号看起来有点奇怪，但是它利用了如下的事实，亦即在处理 WM_CREATE 讯息的过程中，lParam 实际上是指向 CREATESTRUCT (「建立结构」) 结构的指标，该结构有一个 hInstance 成员。所以将 lParam 转换成指向 CREATESTRUCT 结构的一个指标，并取出 hInstance。

有些 Windows 程式使用名为 hInst 的整体变数，使视窗讯息处理程式能存

取 WinMain 中的执行实体代号。在 WinMain 中，您只需在建立主视窗之前设定：

```
hInst = hInstance ;
```

在第七章中的 CHECKER3 程式中，我们曾用 GetWindowLong 取得执行实体代号：

```
GetWindowLong (hwnd, GWL_HINSTANCE)
```

这几种方法都是正确的。

在呼叫 CreateWindow 之後，我们不必再为这些子视窗做任何事情，由 Windows 中的按钮视窗讯息处理程式负责维护它们，并处理所有的重画工作（BS_OWNERDRAW 样式的按钮例外，它要求程式绘制它，这些将在後面加以讨论）。在程式终止时，如果父视窗已经被清除，那么 Windows 将清除这些子视窗。

子视窗向父视窗发讯息

当您执行 BTNLOOK 时，将看到在显示区域的左边会显示出不同的按钮型态。我在前面已经提到过，用滑鼠单击按钮时，子视窗控制项就向其父视窗发送一个 WM_COMMAND 讯息。BTNLOOK 拦截 WM_COMMAND 讯息并显示 wParam 和 lParam 的值，它们的含义如下：

LOWORD (wParam)	子视窗 ID
HIWORD (wParam)	通知码
lParam	子视窗代号

如果您正在移植 16 位元 Windows 程式，那么要注意改变这些讯息参数以容纳 32 位元的代号。

子视窗 ID 是在建立子视窗时传递给 CreateWindow 的值。在 BTNLOOK 中，这些 ID 被显示在显示区域中，并使用 0 到 9 分别标识 10 个按钮。子视窗代号是 Windows 从 CreateWindow 传回的值。

通知码更详细表示了讯息的含义。按钮通知码的可能值在 Windows 表头档案中定义如下：

表 9-1

按钮通知码识别字	值
BN_CLICKED	0
BN_PAINT	1
BN_HILITE or BN_PUSHED	2
BN_UNHILITE or BN_UNPUSHED	3
BN_DISABLE	4
BN_DOUBLECLICKED or BN_DBLCLK	5

BN_SETFOCUS	6
BN_KILLFOCUS	7

实际上，您不会看到这些按钮值中的大多数。从 1 到 4 的通知码是用於一种叫做 BS_USERBUTTON 的已不再使用的按钮的（它已经由 BS_OWNERDRAW 和另一种不同的通知方式所替换）。通知码 6 到 7 只有当按钮样式包括标识 BS_NOTIFY 才发送。通知码 5 只对 BS_RADIOBUTTON、BS_AUTORADIOBUTTON 和 BS_OWNERDRAW 按钮发送，或者当按钮样式中包括 BS_NOTIFY 时，也为其他按钮发送。

您会注意到，在用滑鼠单击按钮时，该按钮文字的周围会有虚线。这表示该按钮拥有了输入焦点，所有键盘输入都将传送给子视窗按钮控制项，而不是传送给主视窗。但是，当该按钮控制项拥有输入焦点时，它将忽略所有的键盘输入，除了 Spacebar 键例外，此时 Spacebar 键与滑鼠具有相同的效果。

父视窗向子视窗发送讯息

虽然 BTNLOOK 中没有显示这一事实，但是父视窗讯息处理程式也能向子视窗控制项发送讯息。这些讯息包括以字首 WM 开头的许多讯息。另外，在 WINUSER.H 中还定义了 8 个按钮说明讯息；字首 BM 表示「按钮讯息」。这些按钮讯息如下表所示：

表 9-2

按钮讯息	值
BM_GETCHECK	0x00F0
BM_SETCHECK	0x00F1
BM_GETSTATE	0x00F2
BM_SETSTATE	0x00F3
BM_SETSTYLE	0x00F4
BM_CLICK	0x00F5
BM_GETIMAGE	0x00F6
BM_SETIMAGE	0x00F7

BM_GETCHECK 和 BM_SETCHECK 讯息由父视窗发送给子视窗控制项，以取得或者设定核取方块和单选按钮的选中标记。BM_GETSTATE 和 BM_SETSTATE 讯息表示按钮处於正常状态还是（滑鼠或 Spacebar 键按下时的）「按下」状态。我们将在讨论按钮的每种型态时，看到这些讯息是如何起作用的。BM_SETSTYLE 讯息允许您在按钮建立之後改变按钮样式。

每个子视窗控制项都具有一个在其兄弟中唯一的视窗代号和 ID 值。对於代号和 ID 这两者，知道其中的一个您就可以获得另一个。如果您知道子视窗控制

项的视窗代号, 那么您可以用下面的叙述来获得 ID:

```
id = GetWindowLong (hwndChild, GWL_ID) ;
```

第七章的 CHECKER3 程式曾用此函式 (与 SetWindowLong 一起) 来维护注册视窗类别时保留的特殊区域的资料。在建立子视窗时, Windows 保留了 GWL_ID 识别字存取的资料。您也可以使用:

```
id = GetDlgCtrlID (hwndChild) ;
```

虽然函式中的「Dlg」部分指的是对话方块, 但实际上这是一个通用的函式。

知道 ID 和父视窗代号, 您就能获得子视窗代号:

```
hwndChild = GetDlgItem (hwndParent, id) ;
```

按键

在 BTNLOOK 中显示的前两个按钮是「压入」按钮。按钮是一个矩形, 包括了 CreateWindow 呼叫中视窗文字参数所指定的文字。该矩形占用了在 CreateWindow 或者 MoveWindow 呼叫中给出的全部高度和宽度, 而文字在矩形的中心。

按键控制项主要用来触发一个立即回应的动作, 而不保留任何形式的开/关指示。两种型态的按钮控制项有两种视窗样式, 分别叫做 BS_PUSHBUTTON 和 BS_DEFPUSHBUTTON, BS_DEFPUSHBUTTON 中的「DEF」代表「内定」。当用来设计对话方块时, BS_PUSHBUTTON 控制项和 BS_DEFPUSHBUTTON 控制项的作用不同。但是当用作子视窗控制项时, 两种型态的按钮作用相同, 尽管 BS_DEFPUSHBUTTON 的边框要粗一些。

当按钮的高度为文字字元高度的 7/4 倍时, 按钮的外观看起来最好, 其中文字字元由 BTNLOOK 使用; 而按钮的宽度至少调节到文字的宽度再加上两个字元的宽度。

当滑鼠游标在按钮中时, 按下滑鼠按键将使按钮用三维阴影重画自己, 就好像真的被按下一样。放开滑鼠按键时, 就恢复按钮的原貌, 并向父视窗发送一个 WM_COMMAND 讯息和 BN_CLICKED 通知码。与其他按钮型态相似, 当按钮拥有输入焦点时, 在文字的周围就有虚线, 按下及释放 Spacebar 键与按下及释放滑鼠按键具有相同的效果。

您可以通过给视窗发送 BM_SETSTATE 讯息来模拟按钮闪动。以下的操作将导致按钮被按下:

```
SendMessage (hwndButton, BM_SETSTATE, 1, 0) ;
```

下面的呼叫使按钮恢复正常:

```
SendMessage (hwndButton, BM_SETSTATE, 0, 0) ;
```

hwndButton 视窗代号是从 CreateWindow 呼叫传回的值。

您也可以向按钮发送 BM_GETSTATE 讯息，子视窗控制项传回按钮目前的状态：如果按钮被按下，则传回 TRUE；如果按钮处于正常状态，则传回 FALSE。但是，绝大多数应用并不需要这一讯息。因为按钮不保留任何开/关资讯，所以 BM_SETCHECK 讯息和 BM_GETCHECK 讯息不会被用到。

核取方块

核取方块是一个文字方块，文字通常出现在核取方块的右边（如果您在建立按钮时指定了 BS_LEFTTEXT 样式，那么文字会出现在左边；您也许将用 BS_RIGHT 直接调整文字来组合此样式）。核取方块通常用于允许使用者对选项进行选择的应用程式中。核取方块的常用功能如同一个开关：单击框一次将显示勾选标记，再次单击清除勾选标记。

核取方块最常用的两种样式是 BS_CHECKBOX 和 BS_AUTOCHECKBOX。在使用 BS_CHECKBOX 时，您需要自己向该控制项发送 BM_SETCHECK 讯息来设定勾选标记。wParam 参数设 1 时设定勾选标记，设 0 时清除勾选标记。通过向该控制项发送 BM_GETCHECK 讯息，您可以得到该核取方块的目前状态。在处理来自控制项的 WM_COMMAND 讯息时，您可以用如下的指令来翻转 X 标记：

```
SendMessage ((HWND) lParam, BM_SETCHECK, (WPARAM)
!SendMessage ((HWND) lParam, BM_GETCHECK, 0, 0), 0) ;
```

注意第二个 SendMessage 呼叫前面的运算符「!」，其中 lParam 是在 WM_COMMAND 讯息中传给使用者视窗讯息处理程式的子视窗代号。如果您以后又想知道按钮的状态，那么可以向它发送另一条 BM_GETCHECK 讯息；您也可以将目前状态储存在您的视窗讯息处理程式中的一个静态变数里，或者向它发送 BM_SETCHECK 讯息来初始化带勾选标记的 BS_CHECKBOX 核取方块：

```
SendMessage (hwndButton, BM_SETCHECK, 1, 0) ;
```

对 BS_AUTOCHECKBOX 样式，按钮自己触发勾选标记的开和关，所以您的视窗讯息处理程式可以忽略 WM_COMMAND 讯息。当您需要按钮目前的状态时，可以向控制项发送 BM_GETCHECK 讯息：

```
iCheck = (int) SendMessage (hwndButton, BM_GETCHECK, 0, 0) ;
```

如果该按钮被选中，则 iCheck 的值为 TRUE 或者非零数；如果按钮未被选中，则 iCheck 的值为 FALSE 或 0。

其余两种核取方块样式是 BS_3STATE 和 BS_AUTO3STATE，正如它们名字所暗示的，这两种样式能显示第三种状态——核取方块内是灰色——它出现在向控制项发送 wParam 等於 2 的 WM_SETCHECK 讯息时。灰色是向使用者表示此框不能被选中的或者禁止使用。

核取方块沿矩形的左边框对齐，并集中在呼叫 CreateWindow 时规定的矩形

的顶边和底边之间，在该矩形内的任何地方按下滑鼠都会向其父视窗发送一个 WM_COMMAND 讯息。核取方块的最小高度是一个字元的高度，最小宽度是文字中的字元数加 2。

单选按钮

单选按钮的名称在一列按钮的後面，这些按钮就像汽车上的收音机一样。汽车收音机上的每一个按钮都对应一种收音状态，而且一次只能有一个按钮被按下。在对话方块中，单选按钮组常常用来表示相互排斥的选项。与核取方块不同，单选按钮的工作与开关不一样，也就是说，当第二次按单选按钮时，它的状态会保持不变。

单选按钮的形状是一个圆圈，而不是方框，除此之外，它非常像核取方块。圆圈内的加重圆点表示该单选按钮已经被选中。单选按钮有视窗样式 BS_RADIOBUTTON 或 BS_AUTORADIOBUTTON 两种，但是後者只用於对话方块。

当您收到来自单选按钮的 WM_COMMAND 讯息时，应该向它发送 wParam 等於 1 的 BM_SETCHECK 讯息来显示其选中状态：

```
SendMessage (hwndButton, BM_SETCHECK, 1, 0) ;
```

对同组中的其他所有单选按钮，您可以通过向它们发送 wParam 等於 0 的 BM_SETCHECK 讯息来显示其未选中状态：

```
SendMessage (hwndButton, BM_SETCHECK, 0, 0) ;
```

分组方块

分组方块即样式为 BS_GROUPBOX 的选择框，它是按钮类中的特例，既不处理滑鼠输入和键盘输入，也不向其父视窗发送 WM_COMMAND 讯息。分组方块是一个矩形框，分组方块标题在其顶部显示。分组方块常用来包含其他的按钮控制项。

改变按钮文字

您可以通过 SetWindowText 来改变按钮（或者其他任何视窗）内的文字：

```
SetWindowText (hwnd, pszString) ;
```

其中 hwnd 是欲改变视窗的代号，pszString 是一个指向以 null 为终结的字符串指标。对於一般的视窗来说，这个文字是标题列的文字；对於按钮控制项来说，它是随著该按钮显示的文字。

您也可以取得视窗目前的文字：

```
iLength = GetWindowText (hwnd, pszBuffer, iMaxLength) ;
```

iMaxLength 指定复制到 pszBuffer 指向的缓冲区中的最大字元数。该函式

传回复制的字元数。您可以首先通过下面的呼叫来获得特定文字的长度：

```
iLength = GetWindowTextLength (hwnd) ;
```

可见的和启用的按钮

为了接收滑鼠和键盘输入，子视窗必须是可见的（被显示）和被启用的。当视窗是可见的而未被启用时，那么视窗将以灰色而非黑色显示文字。

如果在建立子视窗时，您没有将 WS_VISIBLE 包含在视窗类别中，那么直到呼叫 ShowWindow 时子视窗才会被显示出来：

```
ShowWindow (hwndChild, SW_SHOWNORMAL) ;
```

如果您将 WS_VISIBLE 包含在视窗类别中，就没有必要呼叫 ShowWindow。但是，您可以通过呼叫 ShowWindow 将子视窗隐藏起来：

```
ShowWindow (hwndChild, SW_HIDE) ;
```

您可以通过下面的呼叫来确定子视窗是否可见：

```
IsWindowVisible (hwndChild) ;
```

您也可以使子视窗被启用或者不被启用。在内定情况下，视窗是被启用的。您可以通过下面的呼叫使视窗不被启用：

```
EnableWindow (hwndChild, FALSE) ;
```

对于按钮控制项，这具有使按钮字串变成灰色的作用。按钮将不再对滑鼠输入和键盘输入做出回应，这是表示按钮选项目前不可用的最好方法。

您可以通过下面的呼叫使子视窗再次被启用：

```
EnableWindow (hwndChild, TRUE) ;
```

您还可以使用下面的呼叫来确定子视窗是否被启用：

```
IsWindowEnabled (hwndChild) ;
```

按钮和输入焦点

我在本章前面已经提到过，当用滑鼠单击按钮、核取方块、单选框和拥有者绘制按钮时，它们接收到输入焦点。这些控制项使用文字周围的虚线来表示它拥有了输入焦点。当子视窗控制项得到输入焦点时，其父视窗就失去了输入焦点；所有的键盘输入都进入子视窗控制项，而不会进入父视窗中。但是，子视窗控制项只对 Spacebar 键作出回应，此时 Spacebar 键的作用就如同滑鼠按键一样。这种情形导致了一个明显的问题：您的程式失去了对键盘处理的控制项。让我们看看我们对此能做一些什么。

我在第六章中已经提到过，当 Windows 将输入焦点从一个视窗（例如一个父视窗）转换到另一个视窗（例如一个子视窗控制项）时，它首先给正在失去输入焦点的视窗发送一个 WM_KILLFOCUS 讯息，wParam 参数是接收输入焦点的视窗的代号。然后，Windows 向正在接收输入焦点的视窗发送一个 WM_SETFOCUS 讯

息，同时 wParam 是还在失去输入焦点的视窗的代号（在这两种情况中，wParam 值可能为 NULL，它表示没有视窗拥有或者正在接收输入焦点）。

通过处理 WM_KILLFOCUS 讯息，父视窗可以阻止子视窗控制项获得输入焦点。假定阵列 hwndChild 包含了所有子视窗的视窗代号（它们是在呼叫 CreateWindow 来建立视窗的时候储存到阵列中的）。NUM 是子视窗的数目：

```
case WM_KILLFOCUS :
    for ( i = 0 ; i < NUM ; i++)
        if (hwndChild [i] == (HWND) wParam)
        {
            SetFocus (hwnd) ;
            break ;
        }
    return 0 ;
```

在这段程式码中，当父视窗获知它正在失去输入焦点，而让它的某个子视窗得到输入焦点时，它将呼叫 SetFocus 来重新取得输入焦点。

下面是可达到相同目的、但更为简单（但不太直观）的方法：

```
case WM_KILLFOCUS :
    if (hwnd == GetParent ((HWND) wParam))
        SetFocus (hwnd) ;
    return 0 ;
```

但是，这两种方法都有缺点：它们阻止按钮对 Spacebar 键作出回应，因为该按钮总是得不到输入焦点。一个更好的方法是使按钮得到输入焦点，也能让使用者用 Tab 键从一个按钮转移到另一个按钮。这听起来似乎不太可能，在本章的後面，我们将要说明在 COLORS1 程式中如何用「视窗子类别化」技术来实作这种方法。

控制项与颜色

您可以在图 9-1 中看到，许多按钮的显示看起来并不正确。按键还好，但是其他按钮却带有一个本不应该在那里一个矩形灰色背景。这是因为这些按钮本来是为对话方块中的显示而设计的，而在 Windows 98 中，对话方块有一个灰色的表面。我们的视窗有一个白色的表面，这是因为我们在 WNDCLASS 结构中就是这样定义的。

```
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
```

我们已经这么做了，因为我们经常在显示区域中显示文字，而 GDI 使用在内定装置内容中定义的文字颜色和背景颜色，它们总是黑色和白色。为了使这些按钮更加美观一些，我们必须改变显示区域的颜色使之和按钮的背景颜色一致，所以要以某种方法将按钮的背景颜色改为白色。

解决此问题的第一步，是理解 Windows 对「系统颜色」的使用。

系统颜色

Windows 保留了 29 种系统颜色以供各种显示使用。您可以使用 GetSysColor 和 SetSysColors 来获得和设定这些颜色。在 Windows 表头档案中定义的识别字规定了系统颜色。使用 SetSysColors 设定的系统颜色只在目前 Windows 对话过程中有效。

借助 Windows「控制台」程式的「显示器」部分，您可以改变一些（但不是全部）系统颜色。若是 Microsoft Windows NT，选中的颜色会储存在系统登录中；若是 Microsoft Windows 98，则储存在 WIN.INI 档案中。系统登录和 WIN.INI 档案都为这 29 种系统颜色使用了关键字（与 GetSysColor 和 SetSysColors 的识别字不同），在系统颜色的後面跟著红、绿、蓝三种颜色的值，该值的变化范围是 0 到 255。下表说明了这 29 种系统颜色是如何在 GetSysColor、SetSysColors 以及 WIN.INI 关键字中用常数来标识的。这张表是按照 COLOR_ 常数值（从 0 开始到 28 结束）顺序排列的：

表 9-3

GetSysColor 和 SetSysColors	系统登录键或 WIN.INI 识别字	内定的 RGB 值
COLOR_SCROLLBAR	Scrollbar	C0-C0-C0
COLOR_BACKGROUND	Background	00-80-80
COLOR_ACTIVECAPTION	ActiveTitle	00-00-80
COLOR_INACTIVECAPTION	InactiveTitle	80-80-80
COLOR_MENU	Menu	C0-C0-C0
COLOR_WINDOW	Window	FF-FF-FF
COLOR_WINDOWFRAME	WindowFrame	00-00-00
COLOR_MENUTEXT	MenuText	C0-C0-C0
COLOR_WINDOWTEXT	WindowText	00-00-00
COLOR_CAPTIONTEXT	TitleText	FF-FF-FF
COLOR_ACTIVEBORDER	ActiveBorder	C0-C0-C0
COLOR_INACTIVEBORDER	InactiveBorder	C0-C0-C0
COLOR_APPWORKSPACE	AppWorkspace	80-80-80
COLOR_HIGHLIGHT	Highlight	00-00-80
COLOR_HIGHLIGHTTEXT	HighlightText	FF-FF-FF
COLOR_BTNFACE	ButtonFace	C0-C0-C0
COLOR_BTNSHADOW	ButtonShadow	80-80-80
COLOR_GRAYTEXT	GrayText	80-80-80
COLOR_BTNTEXT	ButtonText	00-00-00

COLOR_INACTIVECAPTIONTEXT	InactiveTitleText	C0-C0-C0
COLOR_BTNHIGHLIGHT	ButtonHighlight	FF-FF-FF
COLOR_3DDKSHADOW	ButtonDkShadow	00-00-00
COLOR_3DLIGHT	ButtonLight	C0-C0-C0
COLOR_INFOTEXT	InfoText	00-00-00
COLOR_INFOBK	InfoWindow	FF-FF-FF
[no identifier; use value 25]	ButtonAlternateFace	B8-B4-B8
COLOR_HOTLIGHT	HotTrackingColor	00-00-FF
COLOR_GRADIENTACTIVECAPTION	GradientActiveTitle	00-00-80
COLOR_GRADIENTINACTIVECAPTION	GradientInactiveTitle	80-80-80

这 29 种颜色的预设值是由显示驱动程式提供的，在不同的机器上可能略有不同。

坏消息：虽然这些颜色中有许多似乎都可以从颜色常数名称上了解其代表意义（例如，COLOR_BACKGROUND 是所有视窗后面的桌面区域颜色），在最近版本的 Windows 中系统颜色的使用变得非常混乱。以前，Windows 在视觉上要比今天简单得多。实际上，在 Windows 3.0 以前，只定义了前 13 种系统颜色。但随著使用看起来越来越难以控制的立体外观，相对应地也需要更多的系统颜色。

按钮颜色

对需要多种颜色的每一个按钮来说，这个问题更加地明显。COLOR_BTNFACE 被用於按键主要的表面颜色，以及其他按钮主要的背景颜色（这也是用於对话方块和讯息方块的系统颜色）。COLOR_BTNSHADOW 被建议用作按键右下边、以及核取方块内部和单选按钮圆点的阴影。对於按键，COLOR_BTNTEXT 被用作文字颜色；而对於其他的按钮，则使用 COLOR_WINDOWTEXT 作为文字颜色。还有其他几种系统颜色用於按钮设计的各个部分。

因此，如果您想在我们的显示区域表面显示按钮，那么一种避免颜色冲突的方法便是屈服於这些系统颜色。首先，在定义视窗类别时使用 COLOR_BTNFACE 作为您显示区域的背景颜色：

```
wndclass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1) ;
```

您可以在 BTNLOOK 程式中尝试这种方法。当 WNDCLASS 结构中的 hbrBackground 值是这个值时，Windows 会明白这实际上指的是一种系统颜色而非一个实际的代号。Windows 要求当您在 WNDCLASS 结构的 hbrBackground 栏中指定这些识别字时加上 1，这样做的目的是防止其值为 NULL，而没有任何其他目的。如果您的在程式执行过程中，系统颜色恰好发生了变化，那么显示区域

将变得无效，而 Windows 将使用新的 COLOR_BTNFACE 值。但是现在我们又引发了另一个问题。当您使用 TextOut 显示文字时，Windows 使用的是在装置内容中为背景颜色（它擦除文字後的背景）和文字颜色定义的值，其预设值为白色（背景）和黑色（文字），而不管系统颜色和视窗类别结构中的 hbrBackground 栏位为何值。所以，您需要使用 SetTextColor 和 SetBkColor 将文字和文字背景的颜色改变为系统颜色。您可以在获得装置内容代号之後这么做：

```
SetBkColor (hdc, GetSysColor (COLOR_BTNFACE)) ;  
SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;
```

这样，显示区域背景、文字背景和文字的颜色都与按钮的颜色一致了。但是，如果当您的程式执行时，使用者改变了系统颜色，您可能要改变文字背景颜色和文字颜色。这时您可以使用下面的程式码：

```
case WM_SYSCOLORCHANGE:  
    InvalidateRect (hwnd, NULL, TRUE) ;  
    break ;
```

WM_CTLCOLORBTN 讯息

在这边已经看到了如何将显示区域的颜色和文字颜色调节成按钮的背景颜色。我们是否可以将程式中按钮的颜色调节为我们喜欢的颜色呢？理论上没有问题，但在实际中请别这样做。用 SetSysColors 来改变按钮的外观可能不是您想做的，这会影响目前在 Windows 下执行的所有程式，这也是使用者不太喜欢的。

更好的方法（同样也只是理论上）是处理 WM_CTLCOLORBTN 讯息，这是当子视窗即将为其显示区域著色时，由按钮控制项发送给其父视窗讯息处理程式的一个讯息。父视窗可以利用这个机会来改变子视窗讯息处理程式将用来著色的颜色（在 Windows 的 16 位元版本中，一个称为 WM_CTLCOLOR 的讯息被用於所有的控制项，现在针对每种型态的标准控制项，分别代之以不同的讯息）。

当父视窗讯息处理程式收到 WM_CTLCOLORBTN 讯息时，wParam 讯息参数是按钮的装置内容代号，lParam 是按钮的视窗代号。当父视窗讯息处理程式得到这个讯息时，按钮控制项已经获得了它的装置内容。当您的视窗讯息处理程式处理一个 WM_CTLCOLORBTN 讯息时，您必须完成以下三个动作：

- 使用 SetTextColor 选择设定一种文字颜色。
- 使用 SetBkColor 选择设定一种文字背景颜色。
- 将一个画刷代号传回给子视窗。

理论上，子视窗使用该画刷来著色背景。当不再需要这个画刷时，您应该负责清除它。

下面是使用 WM_CTLCOLORBTN 的问题所在：只有按键和拥有者绘制按钮才给其父视窗发送 WM_CTLCOLORBTN，而只有拥有者绘制按钮才会回应父视窗讯息处理程式对讯息的处理，而使用画刷来著色背景。这基本上是没有意义的，因为无论怎样都是由父视窗来负责绘制拥有者绘制按钮。

在本章後面，我们将说明，在某些情况下，一些类似於 WM_CTLCOLORBTN 但适用於其他型态控制项的讯息将更为有用。

拥有者绘制按钮

如果您想对按钮的所有可见部分实行全面控制，而不想被键盘和滑鼠讯息处理所干扰，那么您可以建立 BS_OWNERDRAW 样式的按钮，如程式 9-2 所展示的那样。

程式 9-2 OWNDRAW

```
OWNDRAW.C
/*-----
   OWNDRAW.C --   Owner-Draw Button Demo Program
                   (c) Charles Petzold, 1996
   -----*/

#include <windows.h>

#define ID_SMALLER                1
#define ID_LARGER                 2
#define BTN_WIDTH                 ( 8 * cxChar)
#define BTN_HEIGHT                ( 4 * cyChar)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("OwnDraw") ;
    MSG               msg ;
    HWND              hwnd ;
    WNDCLASS           wndclass ;

    hInst = hInstance ;
    wndclass.style           = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
```

```
wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName       = szAppName ;
wndclass.lpszClassName      = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Owner-Draw Button Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void Triangle (HDC hdc, POINT pt[])
{
    SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
    Polygon (hdc, pt, 3) ;
    SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND          hwndSmaller, hwndLarger ;
    static int           cxClient, cyClient, cxChar, cyChar ;
    int                  cx, cy ;
    LPDRAWITEMSTRUCT pdis ;
    POINT                pt[3] ;
    RECT                 rc ;

    switch (message)
```

```

{
case WM_CREATE :
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;

    // Create the owner-draw pushbuttons

    hwndSmaller = CreateWindow (TEXT ("button"), TEXT (""),
        WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
        0, 0, BTN_WIDTH, BTN_HEIGHT,
        hwnd, (HMENU) ID_SMALLER, hInst, NULL) ;

    hwndLarger = CreateWindow (TEXT ("button"), TEXT (""),
        WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
        0, 0, BTN_WIDTH, BTN_HEIGHT,
        hwnd, (HMENU) ID_LARGER, hInst, NULL) ;
    return 0 ;

case WM_SIZE :
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    // Move the buttons to the new center

    MoveWindow (    hwndSmaller,    cxClient / 2 - 3 *
BTN_WIDTH / 2,
                cyClient / 2 -    BTN_HEIGHT / 2,
                BTN_WIDTH, BTN_HEIGHT, TRUE) ;
    MoveWindow (    hwndLarger, cxClient / 2 +    BTN_WIDTH /
2, cyClient / 2 -    BTN_HEIGHT / 2,
                BTN_WIDTH, BTN_HEIGHT, TRUE) ;
    return 0 ;

case WM_COMMAND :
    GetWindowRect (hwnd, &rc) ;

    // Make the window 10% smaller or larger

    switch (wParam)
    {
    case ID_SMALLER :
        rc.left    += cxClient / 20 ;
        rc.right   -= cxClient / 20 ;
        rc.top      += cyClient / 20 ;
        rc.bottom  -= cyClient / 20 ;
        break ;

    case ID_LARGER :
        rc.left    -= cxClient / 20 ;

```

```

        rc.right    += cxClient / 20 ;
        rc.top      -= cyClient / 20 ;
        rc.bottom   += cyClient / 20 ;
        break ;
    }

    MoveWindow (    hwnd, rc.left, rc.top, rc.right - rc.left,
rc.top, TRUE) ;
    return 0 ;

case WM_DRAWITEM :
    pdis = (LPDRAWITEMSTRUCT) lParam ;

    // Fill area with white and frame it black

    FillRect        (pdis->hDC, &pdis->rcItem,
(HBRUSH) GetStockObject (WHITE_BRUSH)) ;

    FrameRect (pdis->hDC, &pdis->rcItem,
(        HBRUSH)                GetStockObject
(BLACK_BRUSH)) ;

    //                Draw inward and outward
black triangles

    cx =            pdis->rcItem.right - pdis->rcItem.left ;
    cy =            pdis->rcItem.bottom - pdis->rcItem.top ;

    switch (pdis->CtlID)
    {
    case ID_SMALLER :
        pt[0].x = 3 * cx / 8 ; pt[0].y = 1 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 1 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 3 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;

        pt[0].x = 7 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 7 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 5 * cx / 8 ; pt[2].y = 4 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;

        pt[0].x = 5 * cx / 8 ; pt[0].y = 7 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 7 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 5 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;
    }

```

```

        pt[0].x = 1 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 1 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 3 * cx / 8 ; pt[2].y = 4 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;
        break ;

    case ID_LARGER :
        pt[0].x = 5 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 1 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;

        pt[0].x = 5 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 7 * cx / 8 ; pt[2].y = 4 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;
        pt[0].x = 3 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 7 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;
        pt[0].x = 3 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 1 * cx / 8 ; pt[2].y = 4 * cy / 8 ;

        Triangle (pdis->hDC, pt) ;
        break ;
    }

    // Invert the rectangle if the button is selected

    if (pdis->itemState & ODS_SELECTED)
        InvertRect (pdis->hDC, &pdis->rcItem) ;

    // Draw a focus rectangle if the button has the focus

    if (pdis->itemState & ODS_FOCUS)
    {
        pdis->rcItem.left += cx / 16 ;
        pdis->rcItem.top += cy / 16 ;
        pdis->rcItem.right -= cx / 16 ;
        pdis->rcItem.bottom -= cy / 16 ;

        DrawFocusRect (pdis->hDC, &pdis->rcItem) ;
    }
}

```



```

        return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

该程式在其显示区域的中央包含了两个按钮，如图 9-2 所示。左边的按钮有四个三角形指向按钮的中央，按下该按钮时，视窗的尺寸将缩小 10%。右边的按钮有四个向外指的三角形，按下此按钮时，视窗的尺寸将增大 10%。

如果您只需要在按钮中显示图示或点阵图，您可以用 BS_ICON 或 BS_BITMAP 样式，并用 BM_SETIMAGE 讯息设定点阵图。但是，对于 BS_OWNERDRAW 样式的按钮，它允许完全自由地绘制按钮。

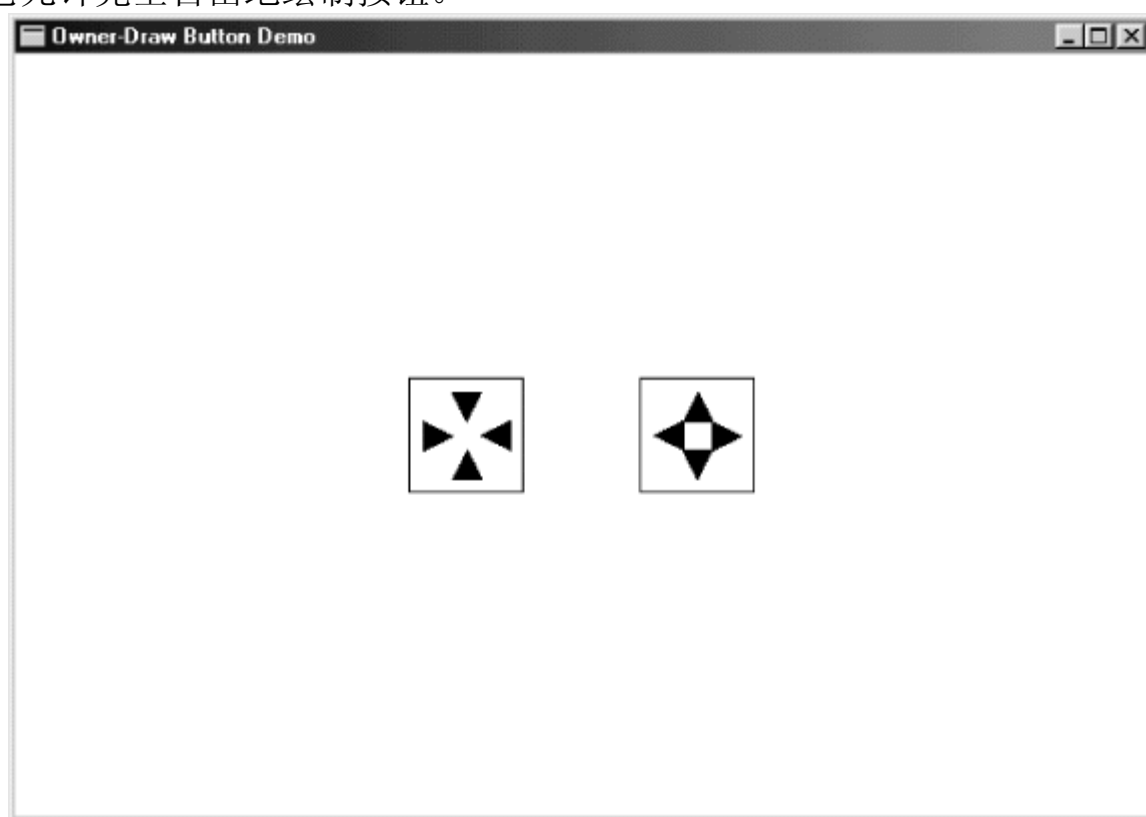


图 9-2 OWNDRAW 的萤幕显示

在处理 WM_CREATE 讯息处理期间，OWNDRAW 建立了两个 BS_OWNERDRAW 样式的按钮；按钮的宽度是系统字体的 8 倍，高度是系统字体的 4 倍（在使用预先定义好的点阵图绘制按钮时，这些尺寸在 VGA 上建立的按钮为 64 图素宽 64 图素高，知道这些资料将非常有用）。这些按钮尚未就定位，在处理 WM_SIZE 讯息处理期间，通过呼叫 MoveWindow 函式，OWNDRAW 将按钮位置放在显示区域的中心。

按下这些按钮时，它们就会产生 WM_COMMAND 讯息。为了处理这些 WM_COMMAND

讯息，OWNDRAW 呼叫 `GetWindowRect`，将整个视窗（不只是显示区域）的位置和尺寸存放在 `RECT`（矩形）结构中，这个位置是相对於萤幕的。然後，根据按下的是左边还是右边的按钮，OWNDRAW 调节这个矩形结构的各个栏位值。程式再通过呼叫 `MoveWindow` 来重新确定位置和尺寸。这将产生另一个 `WM_SIZE` 讯息，按钮被重新定位在显示区域的中央。

如果这是程式所做的全部处理，那么这完全可以，只不过按钮是不可见的。使用 `BS_OWNERDRAW` 样式建立的按钮会在需要重新著色的任何时候都向它的父视窗发送一个 `WM_DRAWITEM` 讯息。这出现在以下几种情况中：当按钮被建立时，当按钮被按下或被放开时，当按钮得到或者失去输入焦点时，以及当按钮需要重新著色的任何时候。

在处理 `WM_DRAWITEM` 讯息处理期间，`lParam` 讯息参数是指向型态 `DRAWITEMSTRUCT` 结构的指标，OWNDRAW 程式将这个指标储存在 `pdis` 变数中，这个结构包含了画该按钮时程式所必需的讯息（这个结构也可以让自绘清单方块和功能表使用）。对按钮而言非常重要的结构栏位有 `hDC`（按钮的装置内容）、`rcItem`（提供按钮尺寸的 `RECT` 结构）、`CtlID`（控制项视窗 ID）和 `itemState`（它说明按钮是否被按下，或者按钮是否拥有输入焦点）。

呼叫 `FillRect` 用白色画刷抹掉按钮的内面，呼叫 `FrameRect` 在按钮的周围画上黑框，由此 OWNDRAW 便启动了 `WM_DRAWITEM` 处理过程。然後，通过呼叫 `Polygon`，OWNDRAW 在按钮上画出 4 个黑色实心的三角形。这是一般的情形。

如果按钮目前被按下，那么 `DRAWITEMSTRUCT` 的 `itemState` 栏位中的某位元将被设为 1。您可以使用 `ODS_SELECTED` 常数来测试这些位元。如果这些位元被设立，那么 OWNDRAW 将通过呼叫 `InvertRect` 将按钮翻转为相反的颜色。如果按钮拥有输入焦点，那么 `itemState` 的 `ODS_FOCUS` 位元将被设立。在这种情况下，OWNDRAW 通过呼叫 `DrawFocusRect`，在按钮的边界内画一个虚线的矩形。

在使用拥有者绘制按钮时，应该注意以下几个方面：Windows 获得装置内容并将其作为 `DRAWITEMSTRUCT` 结构的一个栏位。保持装置内容处於您找到它时所处的状态，任何被选进装置内容的 GDI 物件都必需被释放。另外，当心不要在定义按钮边界的矩形外面进行绘制。

静态类别

在 `CreateWindow` 函式中指定视窗类别为「static」，您就可以建立静态文字的子视窗控制项。这些子视窗非常「文静」。它既不接收滑鼠或键盘输入，也不向父视窗发送 `WM_COMMAND` 讯息。

当您在静态子视窗上移动或者按下滑鼠时，这个子视窗将拦截

WM_NCHITTEST 讯息并将 HTTRANSPARENT 的值传回给 Windows，这将使 Windows 向其下层视窗，通常是它的父视窗，发送相同的 WM_NCHITTEST 讯息。父视窗常常将该讯息传递给 DefWindowProc，在这里，它被转换为显示区域的滑鼠讯息。

前六个静态视窗样式只简单地在子视窗的显示区域内画一个矩形或者边框。在下表的上部，「RECT」静态样式（左列）是填入图样的矩形样式；三个「FRAME」样式（右列）是没有填入图样的矩形轮廓：

「RECT」静态样式	「FRAME」样式
SS_BLACKRECT	SS_BLACKFRAME
SS_GRAYRECT	SS_GRAYFRAME
SS_WHITERECT	SS_WHITEFRAME

「BLACK」、「GRAY」、「WHITE」并不意味着黑、灰和白色，这些颜色是由系统颜色决定的，如表 9-4 所示。

表 9-4

静态控制项	系统颜色
BLACK	COLOR_3DDKSHADOW
GRAY	COLOR_BTNShadow
WHITE	COLOR_BTNHIGHLIGHT

对这些样式，CreateWindow 呼叫中的视窗文字栏位被忽略。矩形的左上角开始於 x 位置坐标和 y 位置坐标，这些坐标都相对於父视窗。您也可以使用 SS_ETCHEDHORZ、SS_ETCHEDVERT 或者 SS_ETCHEDFRAME，采用灰色和白色建立一个形似阴影的边框。

静态类别也包括了三种文字样式：SS_LEFT、SS_RIGHT 和 SS_CENTER。它们建立左对齐、置右对齐和居中文字。文字在 CreateWindow 呼叫的视窗文字参数中给出，并且在以後可以用 SetWindowText 来改变它。当静态控制项的视窗讯息处理程式显示文字时，它使用 DrawText 函式以及 DT_WORDBREAK、DT_NOCLIP 和 DT_EXPANDTABS 参数。文字在子视窗的矩形内可以按文字进行换行。

这三种文字样式子视窗的背景通常为 COLOR_BTNFACE，而文字本身是 COLOR_WINDOWTEXT。在拦截 WM_CTLCOLORSTATIC 讯息时，您可以通过呼叫 SetTextColor 来改变文字颜色，通过 SetBkColor 来改变背景颜色，并传回背景画刷代号。後面的 COLORS1 程式展示了这一点。

最後，静态类别还包括了视窗样式 SS_ICON 和 SS_USERITEM，但是当它们被用作子视窗控制项时却没有任何意义。我们在讨论对话方块时还要提及它们。

卷动列类别

我在第四章首次讨论了卷动列，也讨论了「视窗卷动列」和「卷动列控制项」之间的一些区别。SYSMETS 程式使用视窗卷动列，它出现在视窗的右边和底部。您可以在建立视窗时通过将识别字 WS_VSCROLL、WS_HSCROLL 或者两者都包含在视窗样式中，让视窗加上卷动列。现在我们准备建立一些卷动列控制项，它们是能在父视窗的显示区域的任何地方出现的子视窗。您可以使用预先定义的视窗类别「scrollbar」以及两个卷动列样式 SBS_VERT 和 SBS_HORZ 中的一个来建立子视窗卷动列控制项。

与按钮控制项（以及将在後面讨论的编辑和清单方块控制项）不同，卷动列控制项不向父视窗发送 WM_COMMAND 讯息，而是像视窗卷动列那样发送 WM_VSCROLL 和 WM_HSCROLL 讯息。在处理卷动讯息时，您可以通过 lParam 参数来区分视窗卷动列与卷动列控制项。对于子视窗卷动列其值为 0，对于卷动列控制项其值为卷动列视窗代号。对视窗卷动列和卷动列控制项来说，wParam 参数的高字组和低字组的含义相同。

虽然视窗卷动列有固定的宽度，Windows 使用 CreateWindow 呼叫中（或者在後面的 MoveWindow 呼叫中）给定的矩形尺寸来确定卷动列控制项的尺寸。您可以建立细而长的卷动列控制项，也可以建立短而粗的卷动列控制项。

如果您想建立与视窗卷动列尺寸相同的卷动列控制项，那么可以使用 GetSystemMetrics 取得水平卷动列的高度：

```
GetSystemMetrics (SM_CYHSCROLL) ;
```

或者垂直卷动列的宽度：

```
GetSystemMetrics (SM_CXVSCROLL) ;
```

根据 Windows 文件，卷动列窗样式识别字 SBS_LEFTALIGN、SBS_RIGHTALIGN、SBS_TOPALIGN 和 SBS_BOTTOMALIGN 给出卷动列的标准尺寸，但是这些样式只在对话方块中对卷动列有效。

对视窗卷动列，您可以使用同样的呼叫来建立卷动列控制项的范围和位置：

```
SetScrollRange (hwndScroll, SB_CTL, iMin, iMax, bRedraw) ;
SetScrollPos (hwndScroll, SB_CTL, iPos, bRedraw) ;
SetScrollInfo (hwndScroll, SB_CTL, &si, bRedraw) ;
```

其区别在於：视窗卷动列将父视窗的代号作为第一个参数，并且以 SB_VERT 或者 SB_HORZ 作为第二个参数。

令人吃惊的是，名为 COLOR_SCROLLBAR 的系统颜色不再用於卷动列。两端的按钮和小方块的颜色由 COLOR_BTNFACE、COLOR_BTNHILIGHT、COLOR_BTNSHADOW、COLOR_BTNTEXT（用於小箭头）、COLOR_DKSHADOW 和 COLOR_BTNLIGHT 决定。两端按钮之间区域的颜色由 COLOR_BTNFACE 和

COLOR_BTNHIGHLIGHT 决定。

如果您拦截了 WM_CTLCOLORSCROLLBAR 讯息，那么可以在讯息处理中传回画刷以取代该颜色。让我们来试一下。

COLORS1 程式

为了解卷动列和静态子视窗的一些用法——也为了深入了解颜色——我们将使用 COLORS1 程式，如程式 9-3 所示。COLORS1 在显示区域的左半部显示三种卷动列，并分别标以「Red」、「Green」和「Blue」。当您挪动卷动列时，显示区域的右半部将变为三种原色混合而成的合成色，三种原色的数值显示在三个卷动列的下面。

程式 9-3 COLORS1

```
COLORS1.C
/*-----
COLORS1.C -- Colors Using Scroll Bars
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ScrollProc (HWND, UINT, WPARAM, LPARAM) ;

int idFocus ;
WNDPROC OldScroll[3] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Colors1") ;
    HWND hwnD ;
    MSG msg ;
    WNDCLASS wndclass ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = CreateSolidBrush (0) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;
```

```

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName, MB_ICONERROR) ;

        return 0 ;
    }
    hwnd = CreateWindow (  szAppName, TEXT ("Color Scroll"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static COLORREF crPrim[3] = {    RGB (255, 0, 0), RGB (0, 255, 0),
        RGB (0, 0, 255) } ;
    static HBRUSH      hBrush[3], hBrushStatic ;
    static HWND        hwndScroll[3],    hwndLabel[3],    hwndValue[3],
hwndRect ;
    static int          color[3], cyChar ;
    static RECT          rcColor ;
    static TCHAR *      szColorLabel[] = {    TEXT ("Red"), TEXT ("Green"),
        TEXT ("Blue") } ;
    HINSTANCE            hInstance ;
    int                  i, cxClient, cyClient ;
    TCHAR                szBuffer[10] ;

    switch (message)
    {
    case  WM_CREATE :
        hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

        // Create the white-rectangle window against which the
        // scroll bars will be positioned. The child window ID is 9.

```

```

        hwndRect = CreateWindow (TEXT ("static"), NULL,
            WS_CHILD | WS_VISIBLE | SS_WHITERECT,
            0, 0, 0, 0,
            hwnd, (HMENU) 9, hInstance, NULL) ;

    for (i = 0 ; i < 3 ; i++)
    {
        // The three scroll bars have IDs 0, 1, and 2, with
        // scroll bar ranges from 0 through 255.

        hwndScroll[i] = CreateWindow (TEXT ("scrollbar"), NULL,
            WS_CHILD | WS_VISIBLE |
            WS_TABSTOP | SBS_VERT,
            0, 0, 0, 0,
            hwnd, (HMENU) i, hInstance, NULL) ;

        SetScrollRange (hwndScroll[i], SB_CTL, 0, 255, FALSE) ;
        SetScrollPos (hwndScroll[i], SB_CTL, 0, FALSE) ;

        // The three color-name labels have IDs 3, 4, and 5,
        // and text strings "Red", "Green", and "Blue".

        hwndLabel [i] = CreateWindow (TEXT ("static"), zColorLabel[i],
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            0, 0, 0, 0,
            hwnd, (HMENU) (i + 3),
            hInstance, NULL) ;

        // The three color-value text fields have IDs 6, 7,
        // and 8, and initial text strings of "0".

        hwndValue [i] = CreateWindow (TEXT ("static"), TEXT ("0"),
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            0, 0, 0, 0,
            hwnd, (HMENU) (i + 6),
            hInstance, NULL) ;

        OldScroll[i] = (WNDPROC) SetWindowLong (hwndScroll[i],
            GWL_WNDPROC, (LONG) ScrollProc) ;

        hBrush[i] = CreateSolidBrush (crPrim[i]) ;
    }

    hBrushStatic = CreateSolidBrush (
        GetSysColor (COLOR_BTNHIGHLIGHT)) ;

    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    return 0 ;

```

```
case WM_SIZE :
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    SetRect (&rcColor, cxClient / 2, 0, cxClient, cyClient) ;

    MoveWindow (hwndRect, 0, 0, cxClient / 2, cyClient, TRUE) ;

    for (i = 0 ; i < 3 ; i++)
    {
        MoveWindow (hwndScroll[i],
            (2 * i + 1) * cxClient / 14, 2 * cyChar,
            cxClient / 14, cyClient - 4 * cyChar, TRUE) ;

        MoveWindow (hwndLabel[i],
            (4 * i + 1) * cxClient / 28, cyChar / 2,
            cxClient / 7, cyChar, TRUE)

        MoveWindow (hwndValue[i],
            (4 * i + 1) * cxClient / 28,
            cyClient - 3 * cyChar / 2,
            cxClient / 7, cyChar, TRUE) ;
    }

    SetFocus (hwnd) ;
    return 0 ;

case WM_SETFOCUS :
    SetFocus (hwndScroll[idFocus]) ;
    return 0 ;

case WM_VSCROLL :
    i = GetWindowLong ((HWND) lParam, GWL_ID) ;

    switch (LOWORD (wParam))
    {
    case SB_PAGEDOWN :
        color[i] += 15 ;
        // fall through
    case SB_LINEDOWN :
        color[i] = min (255, color[i] + 1) ;
        break ;

    case SB_PAGEUP :
        color[i] -= 15 ;
        // fall through
    case SB_LINEUP :
        color[i] = max (0, color[i] - 1) ;
        break ;
```



```

        case SB_TOP :
            color[i] = 0 ;
            break ;

        case SB_BOTTOM :
            color[i] = 255 ;
            break ;

        case SB_THUMBPOSITION :
        case SB_THUMBTRACK :
            color[i] = HIWORD (wParam) ;
            break ;

        default :
            break ;
    }
    SetScrollPos (hwndScroll[i], SB_CTL, color[i], TRUE) ;
    wsprintf (szBuffer, TEXT ("%i"), color[i]) ;
    SetWindowText (hwndValue[i], szBuffer) ;

    DeleteObject ((HBRUSH)
        SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
        CreateSolidBrush (RGB (color[0], color[1], color[2])))) ;

    InvalidateRect (hwnd, &rcColor, TRUE) ;
    return 0 ;

case WM_CTLCOLORSCROLLBAR :
    i = GetWindowLong ((HWND) lParam, GWL_ID) ;
    return (LRESULT) hBrush[i] ;

case WM_CTLCOLORSTATIC :
    i = GetWindowLong ((HWND) lParam, GWL_ID) ;

    if (i >= 3 && i <= 8) // static text controls
    {
        SetTextColor ((HDC) wParam, crPrim[i % 3]) ;
        SetBkColor ((HDC) wParam, GetSysColor
(COLOR_BTNHIGHLIGHT));
        return (LRESULT) hBrushStatic ;
    }
    break ;
case WM_SYSCOLORCHANGE :
    DeleteObject (hBrushStatic) ;
    hBrushStatic = CreateSolidBrush
(GetSysColor(COLOR_BTNHIGHLIGHT)) ;
    return 0 ;

```

```

    case WM_DESTROY :
        DeleteObject ((HBRUSH)
            SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
                GetStockObject (WHITE_BRUSH))) ;

        for (i = 0 ; i < 3 ; i++)
            DeleteObject (hBrush[i]) ;

        DeleteObject (hBrushStatic) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ScrollProc (HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    int id = GetWindowLong (hwnd, GWL_ID) ;
    switch (message)
    {
        case WM_KEYDOWN :
            if (wParam == VK_TAB)
                SetFocus (GetDlgItem (GetParent (hwnd),
                    (id + (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3)) ;
            break ;
        case WM_SETFOCUS :
            idFocus = id ;
            break ;
    }
    return CallWindowProc (OldScroll[id], hwnd, message, wParam, lParam) ;
}

```

COLORS1 利用子视窗进行工作，该程式使用 10 个子视窗控制项：3 个滚动列、6 个静态文字视窗和 1 个静态矩形框。COLORS1 拦截 WM_CTLCOLORSCROLLBAR 讯息来给红、绿、蓝 3 个滚动列的内部著色，并拦截 WM_CTLCOLORSTATIC 讯息来著色静态文字。

您可以使用滑鼠或者键盘来挪动滚动列，从而利用 COLORS1 作为一种实验颜色显示的开发工具，为您自己的 Windows 程式选择漂亮的颜色（或者，您可能更喜欢难看的颜色）。COLORS1 的显示如图 9-3 所示。不幸的是，这些颜色在印表纸上被显示为不同深浅的灰色。

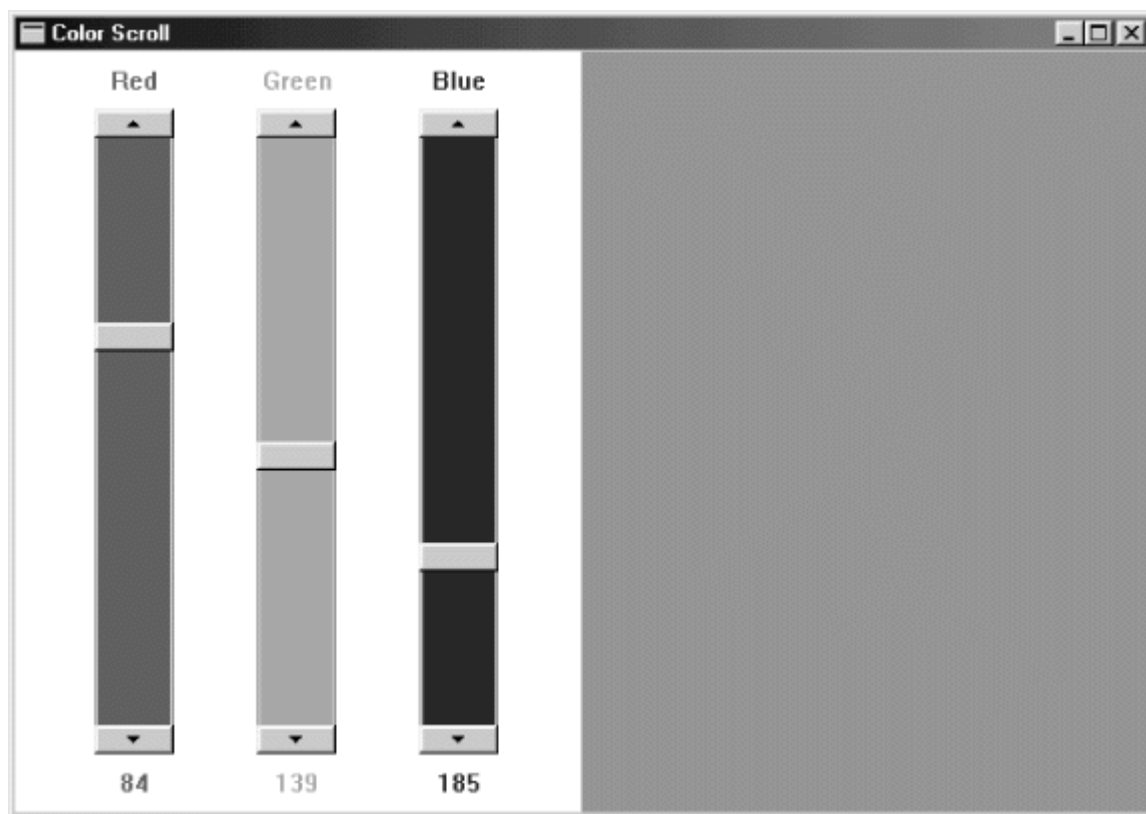


图 9-3 COLORS1 的萤幕显示

COLORS1 不处理 WM_PAINT 讯息，所有的工作几乎都是由子视窗完成的。

显示区域右半部显示的颜色实际上是视窗的背景颜色。SS_WHITERECT 样式的静态子视窗显示在显示区域的左半部。三个卷动列是 SBS_VERT 样式的子视窗控制项，它们被定位在 SS_WHITERECT 子视窗的顶部。另外六个 SS_CENTER 样式（居中文字）的静态子视窗提供标签和颜色值。COLORS1 在 WinMain 函式中用 CreateWindow 建立它的普通重叠式视窗和 10 个子视窗。SS_WHITERECT 和 SS_CENTER 静态视窗使用视窗类别「static」；三个卷动列使用视窗类别「scrollbar」。

CreateWindow 呼叫中的 x 位置、y 位置、宽度和高度参数最初设为 0，因为位置和大小都取决於显示区域的尺寸，而它目前尚未确定。COLORS1 的视窗讯息处理程式在接收到 WM_SIZE 讯息时，就使用 MoveWindow 给 10 个子视窗重新确定大小。所以，每当您对 COLORS1 视窗进行缩放时，卷动列的尺寸就会按比例变化。

当 WndProc 视窗讯息处理程式收到 WM_VSCROLL 讯息时，lParam 参数的高字组就是子视窗的代号。我们可以使用 GetWindowWord 来得到子视窗的 ID：

```
i = GetWindowLong ((HWND) lParam, GWL_ID) ;
```

对于这三个卷动列，我们已经按习惯将其 ID 设为 0、1、2，所以 WndProc 能区别出是哪个卷动列在产生讯息。

由于子视窗的代号在建立时就被储存在阵列中，所以 WndProc 就能对相对

应的卷动列讯息进行处理，并通过呼叫 SetScrollPos 来设定相对应的新值：

```
SetScrollPos (hwndScroll[i], SB_CTL, color[i], TRUE) ;
```

WndProc 也改变卷动列底部子视窗的文字：

```
wsprintf (szBuffer, TEXT ("%i"), color[I]) ;  
SetWindowText (hwndValue[i], szBuffer) ;
```

自动键盘介面

卷动列控制项也能处理键盘输入，但是只有在拥有输入焦点时才行。下表说明怎样将键盘游标键转变为卷动讯息：

表 9-5

游标键	卷动讯息的 wParam 值
Home	SB_TOP
End	SB_BOTTOM
Page Up	SB_PAGEUP
Page Down	SB_PAGEDOWN
左或上	SB_LINEUP
右或下	SB_LINEDOWN

事实上，SB_TOP 和 SB_BOTTOM 卷动讯息只能用键盘产生。在使用滑鼠按动卷动列时，如果想使该卷动列获得输入焦点，那么您必须将 WS_TABSTOP 识别字包含到 CreateWindow 呼叫的视窗类别参数中。当卷动列拥有输入焦点时，在该卷动列的小方框上将显示一个闪烁的灰色块。

为了给卷动列提供全面的键盘介面，还需要另外一些工作。首先，WndProc 视窗讯息处理程式必须使卷动列拥有输入焦点，它是通过处理 WM_SETFOCUS 讯息来完成这一点的，该 WM_SETFOCUS 讯息是当卷动列获得输入焦点时其父视窗接收到的。WndProc 给其中一个卷动列设定输入焦点。

```
SetFocus (hwndScroll[idFocus]) ;
```

其中 idFocus 是一个整体变数。

但是，还需要一些借助键盘尤其是 Tab 键，来从一个卷动列转换到另一个卷动列的方法。这比较困难，因为一旦某个卷动列拥有了输入焦点，它就处理所有的键盘输入，但卷动列只关心游标键，而忽略 Tab 键。解决这一两难处境的方法是「视窗子类别化」。我们将用它来给 COLORS1 增加使用 Tab 键从一个卷动列跳到另一个卷动列的功能。

视窗子类别化 (Window Subclassing)

卷动列控制项的视窗讯息处理程式是 Windows 内部的。但是，将 GWL_WNDPROC

识别字作为参数来呼叫 `GetWindowLong`，您就可以得到这个视窗讯息处理程式的位址。另外，您可以呼叫 `SetWindowLong` 给该卷动列设定一个新的视窗讯息处理程式，这个技术叫做「视窗子类别化」，非常有用。它能让您给现存的视窗讯息处理程式设定「挂勾」，以便在自己的程式中处理一些讯息，同时将所有讯息传递给旧的视窗讯息处理程式。

在 `COLORS1` 中对卷动讯息进行初步处理的视窗讯息处理程式叫做 `ScrollProc`，它在 `COLORS1.C` 档案的尾部。由於 `ScrollProc` 是 `COLORS1` 中的函式，而 Windows 将呼叫 `COLORS1`，所以 `ScrollProc` 必须被定义为 `callback` 函式。

对三个卷动列中的每一个，`COLORS1` 使用 `SetWindowLong` 来设定新的卷动列视窗讯息处理程式的位址，并取得现存卷动列视窗讯息处理程式的位址：

```
OldScroll[i] = (WNDPROC) SetWindowLong (hwndScroll[i], GWL_WNDPROC,
    (LONG) ScrollProc) ;
```

现在，函式 `ScrollProc` 得到了 Windows 发送到 `COLORS1` 中三个卷动列（当然不是其他程式中的卷动列）的卷动列视窗讯息处理程式的全部讯息。`ScrollProc` 视窗讯息处理程式在接收到 `Tab` 或者 `Shift-Tab` 键时，就将输入焦点改变到下一个（或者上一个）卷动列。它使用 `CallWindowProc` 呼叫旧的卷动列视窗讯息处理程式。

给背景著色

当 `COLORS1` 定义它的视窗类别时，也为其显示区域背景定义了一个实心的黑色画刷：

```
wndclass.hbrBackground = CreateSolidBrush (0) ;
```

当您改变 `COLORS1` 的卷动列设定时，程式必须建立一个新的画刷，并将该新画刷代号放入视窗类别结构中。如同使用 `GetWindowLong` 和 `SetWindowLong` 能得到并设定卷动列视窗讯息处理程式一样，用 `GetClassWord` 和 `SetClassWord` 能得到这个画刷的代号。

您可以建立新的画刷并将其代号插入视窗类别结构中，然後删除旧的画刷：

```
DeleteObject ((HBRUSH)
    SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
        CreateSolidBrush (RGB (color[0], color[1], color[2])))) ;
```

Windows 下一次重新为视窗的背景著色时，将使用这个新画刷。为了强迫 Windows 抹掉背景，我们将使整个显示区域无效：

```
InvalidateRect (hwnd, &rcColor, TRUE) ;
```

`TRUE`（非零）值作为第三个参数，表示希望在重新著色之前删去背景。

`InvalidateRect` 使 Windows 在视窗讯息处理程式的讯息佇列中放进一个 `WM_PAINT` 讯息。由於 `WM_PAINT` 讯息的优先等级比较低，所以，如果您还在使用

滑鼠或者游标键移动卷动列的话，这个讯息将不会立即被处理。如果您想在颜色改变之後使该视窗立即变成最新的（目前的），那么您可以在 `InvalidateRect` 之後增加下面的叙述：

```
UpdateWindow (hwnd) ;
```

但这会使得键盘和滑鼠处理变慢。

`COLORS1` 中的 `WndProc` 函式不处理 `WM_PAINT` 讯息，而是将其传给 `DefWindowProc`。Windows 对 `WM_PAINT` 讯息的內定处理只是呼叫 `BeginPaint` 和 `EndPaint` 使视窗生效。因为在 `InvalidateRect` 呼叫中已经指定背景要被抹掉，所以 `BeginPaint` 呼叫使 Windows 发出一个 `WM_ERASEBKGND`（删除背景）讯息，`WndProc` 也将忽略这个讯息。Windows 用视窗类别中指定的画刷将显示区域的背景抹去，这样就处理了这个讯息。

在终止以前进行清除总是一个好主意，因此在处理 `WM_DESTROY` 讯息处理期间，再一次呼叫 `DeleteObject`：

```
DeleteObject ((HBRUSH)
    SetClassLong (hwnd, GCL_HBRBACKGROUND,
        (LONG) GetStockObject (WHITE_BRUSH))) ;
```

给卷动列和静态文字著色

在 `COLORS1` 中，三个卷动列的内部和六个文字栏位中的文字著色为红、绿和蓝色。卷动列的著色是通过处理 `WM_CTLCOLORSCROLLBAR` 讯息来完成的。

在 `WndProc` 中，我们为画刷定义了一个由三个代号组成的静态阵列：

```
static HBRUSH hBrush [3] ;
```

在处理 `WM_CREATE` 期间，我们建立三个画刷：

```
for (I = 0 ; I < 3 ; I++)
    hBrush[0] = CreateSolidBrush (crPrim [I]) ;
```

其中 `crPrim` 阵列中包含三种原色的 RGB 值。在 `WM_CTLCOLORSCROLLBAR` 处理期间视窗讯息处理程式传回这三画刷中的一个：

```
case WM_CTLCOLORSCROLLBAR:
    i = GetWindowLong ((HWND) lParam, GWL_ID) ;
    return (LRESULT) hBrush [i] ;
```

在处理 `WM_DESTROY` 讯息的过程中，这些画刷必须被删除：

```
for (i = 0 ; i < 3 ; i++)
    DeleteObject (hBrush [i]) ;
```

同样地，静态文字栏位中的文字是在处理 `WM_CTLCOLORSTATIC` 讯息中呼叫 `SetTextColor` 来著色的。文字背景用 `SetBkColor` 函式设定为系统颜色 `COLOR_BTNHIGHLIGHT`，这导致文字背景颜色和卷动列与文字後面的静态矩形控制项的颜色一样。对於静态文字控制项，这种文字背景颜色只用於字串中每个

字元後面的矩形，而不会用於整个控制项视窗。为了实作这一点，视窗讯息处理程式还必须传回 COLOR_BTNHIGHLIGHT 颜色画刷的代号。这个画刷被称为 hBrushStatic，它在 WM_CREATE 讯息处理期间建立，在 WM_DESTROY 讯息处理期间清除。

在 WM_CREATE 讯息处理期间依据 COLOR_BTNHIGHLIGHT 颜色建立画刷，并且在执行期间使用这一画刷时，我们遇到了一个小问题。如果程式在执行期间改变了 COLOR_BTNHIGHLIGHT 颜色，那么静态矩形的颜色将发生变化，并且文字背景的颜色也会变化，但是文字视窗控制项的整个背景将保持原有的 COLOR_BTNHIGHLIGHT 颜色。

为了解决这个问题，COLORS1 也简单地通过使用新颜色重新建立 hBrushStatic 来处理 WM_SYSCOLORCHANGE 讯息。

编辑类别

在某些方面，编辑类别是最简单的预先定义视窗类别；在另一方面，它又是最复杂的视窗类别。当您使用类别名称「edit」建立子视窗时，您根据 CreateWindow 呼叫中的 x 位置、y 位置、宽度和高度这些参数定义了一个矩形。此矩形含有可编辑文字。当子视窗控制项拥有输入焦点时，您可以输入文字，移动游标，使用滑鼠或者 Shift 键与一个游标键来选取部分文字，按 Ctrl-X 来删除所选文字或按 Ctrl-C 来复制所选文字、并送到剪贴簿上，按 Ctrl-V 键插入剪贴簿上的文字。

编辑控制项的最简单的应用之一是作为单行输入区域。但是编辑控制项并不仅限于单行，这一点我将在程式 9-4 POPPAD1 中说明。和我们在这本书中所遇到的各种其他问题一样，POPPAD 程式将逐步增强以使用功能表、对话方块(载入与储存档案)和列印。最後的版本将是一个简单而完整的文字编辑器，且其程式码将非常简洁。

程式 9-4 POPPAD1

```
POPPAD1.C
/*-----
   POPPAD1.C -- Popup Editor using child window edit box
               (c) Charles Petzold, 1998
   -----*/

#include <windows.h>
#define ID_EDIT    1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
```

```
TCHAR szAppName[] = TEXT ("PopPad1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, szAppName,
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND hwndEdit ;
    switch (message)
```



```

{
case WM_CREATE :
    hwndEdit = CreateWindow (TEXT ("edit"), NULL,
        WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
        WS_BORDER | ES_LEFT | ES_MULTILINE |
        ES_AUTOHSCROLL | ES_AUTOVSCROLL,
        0, 0, 0, 0, hwnd, (HMENU) ID_EDIT,
        ((LPCREATESTRUCT) lParam) -> hInstance, NULL) ;
    return 0 ;

case WM_SETFOCUS :
    SetFocus (hwndEdit) ;
    return 0 ;

case WM_SIZE :
    MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam), TRUE) ;
    return 0 ;

case WM_COMMAND :
    if (LOWORD (wParam) == ID_EDIT)
        if (HIWORD (wParam) == EN_ERRSPACE ||
            HIWORD (wParam) == EN_MAXTEXT)
            MessageBox (hwnd, TEXT ("Edit control out of space."),
                szAppName, MB_OK | MB_ICONSTOP) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

POPPAD1 是一个多行编辑器 (只是没有档案 I/O), 其 C 语言原始码不到 100 行 (不过, 有一个缺陷, 即预先定义的多行编辑控制项只限于 30,000 字元的文字)。您可以看到, POPPAD1 本身并没有做多少工作, 预先定义的编辑控制项完成了许多工作, 这样, 您可以知道, 无需额外的程式时编辑控制项能做些什么。

编辑类别样式

如前面所提到的, 在 CreateWindow 呼叫中将「edit」作为视窗类别建立了一个编辑控制项, 视窗样式是 WS_CHILD 加上几个选项。如同在静态子视窗控制项中一样, 编辑控制项中的文字可以置左对齐、置右对齐或者居中, 您使用视窗样式 ES_LEFT、ES_RIGHT 和 ES_CENTER 来指定这些格式。

内定状态下, 编辑控制项是单行的。您使用 ES_MULTILINE 视窗样式可以建

立多行编辑控制项。对于单行编辑控制项，您一般只可以在编辑控制项矩形的尾部输入文字。要建立一个自动水平卷动的编辑控制项，您可以采用样式 ES_AUTOHSCROLL。对于一个多行编辑控制项，文字会自动跳行，除非使用 ES_AUTOHSCROLL 样式。在这种情况下，您必须按 Enter 键来开始新的一行。您还可以使用样式 ES_AUTOVSCROLL 来将垂直卷动列包括在多行编辑控制项中。

当您在多行编辑控制项中包括这些卷动样式时，也许还想给编辑控制项增加卷动列。要做到这些，可以对非子视窗使用同一视窗样式识别字 WS_HSCROLL 和 WS_VSCROLL。内定状态下，编辑控制项没有边界，利用样式 WS_BORDER 则可以增加边界。

当您在编辑控制项中选择文字时，Windows 将选择的文字反白显示。但是当编辑控制项失去输入焦点时，被选择的文字将不再被加亮。如果希望在编辑控制项没有输入焦点时被选择的文字仍然被加亮，您可以使用样式 ES_NOHIDESEL。

在 POPPAD1 建立其编辑控制项时，CreateWindow 呼叫依如下形式给出样式：

```
WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |  
    WS_BORDER | ES_LEFT | ES_MULTILINE |  
    ES_AUTOHSCROLL | ES_AUTOVSCROLL
```

在 POPPAD1 中，编辑控制项的大小是后来当 WndProc 接收到 WM_SIZE 讯息时通过呼叫 MoveWindow 来定义的。编辑控制项的尺寸被简单地设定为主视窗的尺寸：

```
MoveWindow (hwndEdit, 0, 0, LOWORD (lParam),  
            HIWORD (lParam), TRUE) ;
```

对于单行编辑控制项，控制项的高度必须可以容纳一个字元。如果编辑控制项有边界（大多数都有），那么使用一个字元高度的 1.5 倍（包括外部间距）。

编辑控制项通知

编辑控制项给父视窗讯息处理程式发送 WM_COMMAND 讯息，对按钮控制项来说，wParam 和 lParam 变数的含义是相同的：

LOWORD (wParam)	子视窗 ID
HIWORD (wParam)	通知码
lParam	子视窗代号

通知码如下所示：

EN_SETFOCUS	编辑控制项已经获得输入焦点
EN_KILLFOCUS	编辑控制项已经失去输入焦点
EN_CHANGE	编辑控制项的内容将改变
EN_UPDATE	编辑控制项的内容已经改变
EN_ERRSPACE	编辑控制项执行已经超出中间

EN_MAXTEXT	编辑控制项在插入时执行超出空间
EN_HSCROLL	编辑控制项的水平卷动列已经被按下
EN_VSCROLL	编辑控制项的垂直卷动列已经被按下

POPPAD1 只拦截 EN_ERRSPACE 和 EN_MAXTEXT 通知码, 并显示一个讯息方块。

使用编辑控制项

如果在您的主视窗上使用了几个单行编辑控制项, 那么您需要将视窗子类别化以便把输入焦点从一个控制项转移到另一个控制项。您可以通过拦截 Tab 键和 Shift-Tab 键来完成这种移动, 非常像 COLORS1 中所做的 (视窗子类别化的另一个例子在后面的 HEAD 程式中说明)。如何处理 Enter 键取决於您, 可以像 Tab 键那样使用, 也可以当成给程式的信号, 表示所有的编辑栏位都准备好了。

如果您想在编辑区中插入文字, 那么可以使用 SetWindowText 来做到。将文字从编辑控制项中取出涉及了 GetWindowTextLength 和 GetWindowText, 我们将在 POPPAD 程式的修订版本中看到这些操作的实例。

发送给编辑控制项的讯息

因为用 SendMessage 发送给编辑控制项的讯息很多, 并且其中的几个还将在后面 POPPAD 修订版本中用到, 所以这里不解说所有用 SendMessage 发送给编辑控制项的讯息, 只概要地说明一下。

这些讯息允许您剪下、复制或者清除目前被选择的文字。使用者使用滑鼠或者 Shift 键加上游标控制项键来选择文字并进行上面的操作, 这样, 在编辑控制项中选中的文字将被加亮:

```
SendMessage (hwndEdit, WM_CUT, 0, 0) ;
SendMessage (hwndEdit, WM_COPY, 0, 0) ;
SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
```

WM_CUT 将目前选择的文字从编辑控制项中移走, 并将其发送到剪贴簿中; WM_COPY 将选择的文字复制到剪贴簿上并保持编辑控制项中的内容完好无损; WM_CLEAR 将选择的内容从编辑控制项中删除, 但是不向剪贴簿中发送。

您也可以将剪贴簿上的文字插入到编辑控制项中的游标位置:

```
SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
```

您可以取得目前选择的起始位置和末尾位置:

```
SendMessage (hwndEdit, EM_GETSEL, (LPARAM) &iStart,
              (LPARAM) &iEnd) ;
```

结束位置实际上是最後一个选择字元的位置加 1。

您可以选择文字:

```
SendMessage (hwndEdit, EM_SETSEL, iStart, iEnd) ;
```

您还可以使用别的文字来置换目前的选择内容：

```
SendMessage (hwndEdit, EM_REPLACESEL, 0, (LPARAM) szString) ;
```

对多行编辑控制项，您可以取得行数：

```
iCount = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0) ;
```

对任何特定的行，您可以取得距离编辑缓冲区文字开头的偏移量：

```
iOffset = SendMessage (hwndEdit, EM_LINEINDEX, iLine, 0) ;
```

行数从 0 开始计算，iLine 值为-1 时传回包含游标所在行的偏移量。您可以取得行的长度：

```
iLength = SendMessage (hwndEdit, EM_LINELENGTH, iLine, 0) ;
```

并将行本身复制到一个缓冲区中：

```
iLength = SendMessage (hwndEdit, EM_GETLINE, iLine, (LPARAM) szBuffer) ;
```

清单方块类别

我在本章讨论的最后一个预先定义子视窗控制项是清单方块。一个清单方块是字串的集合，这些字串是一个矩形中可以卷动显示的清单。——程式通过向清单方块视窗讯息处理程式发送讯息，可以在清单中增加或者删除字串。当清单方块中的某项被选择时，清单方块控制项就向其父视窗发送 WM_COMMAND 讯息，父视窗也就可以确定选择的是哪一项。

一个清单方块可以是单选，也可以是多选的，后者允许使用者从清单方块中选择多个项目。当清单方块拥有输入焦点时，其中项目的周围显示有虚线。在清单方块中，游标位置并不指明被选择的项目。被选择的项目被加亮显示，并且是反白显示的。

在单项选择的清单方块中，使用者按 Spacebar 键就可以选择游标所在位置的项目。方向键移动游标和目前选择指示，并且能够滚动清单方块的内容。Page Up 和 Page Down 键也能滚动清单方块，但它移动的是游标而不是选择指示。按字母键能将游标和选择指示移到以此字母开头的第一个（或下一个）选项。也可以使用滑鼠在要选择的项目上单击或者双击来选择它。

在多项选择清单方块中，Spacebar 键可以切换游标所在位置的项目的选择状态（如果该项已经被选择，则取消选择）。如同在单项选择清单方块中一样，方向键取消前面选择过的项目，并且移动游标和选择指示。但是，Ctrl 键和方向键能够在移动游标的同时不移动选择，Shift 键加方向键能扩展一个选择。

在多项选择清单方块中，单击或者双击滑鼠按键能取消之前所有的选择，而选择被点中的项目。但是，如果在滑鼠点中某一项的同时也按下 Shift 键，则只能切换该项的选择状态，而不会改变任何其他项的选择状态。

清单方块样式

当您使用 `CreateWindow` 建立清单方块子视窗时，您应该将「`listbox`」作为视窗类别，将 `WS_CHILD` 作为视窗样式。但是，这个内定清单方块样式不向其父视窗发送 `WM_COMMAND` 讯息，这样一来，程式必须向清单方块询问其中的项目的选择状态（借助於发送给清单方块控制项的讯息）。所以，清单方块控制项通常都包括清单方块样式识别字 `LBS_NOTIFY`，它允许父视窗接收来自清单方块的 `WM_COMMAND` 讯息。如果您希望清单方块控制项对清单方块中的项目进行排序，那么您可以使用另一种常用的样式 `LBS_SORT`。

内定情况下，清单方块是单项选择的。多项选择的清单方块相当少。如果您想建立一个多项选择清单方块，那么您可以使用样式 `LBS_MULTIPLESEL`。通常，当给有卷动列的清单方块增加新项目时，清单方块本身会自己重画。您可以通过将样式 `LBS_NOREDRA` 包含进去来防止这种现象。但是您也许不想使用这种样式，这时可以使用 `WM_SETREDRAW` 讯息来暂时防止清单方块控制项重新画过，我将在稍後讨论 `WM_SETREDRAW` 讯息。

内定状态下，清单方块视窗讯息处理程式只显示列表项目，它的周围没有任何边界。您可以使用视窗样式识别字 `WS_BORDER` 来加上边界。另外，您可以使用视窗样式识别字 `WS_VSCROLL` 来增加垂直卷动列，以便使用滑鼠来卷动列表项目。

Windows 表头档案定义了一个清单方块样式，叫做 `LBS_STANDARD`，它包含了最常用的样式，其定义如下：

```
(LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER)
```

您也可以采用 `WS_SIZEBOX` 和 `WS_CAPTION` 识别字，但是这两个识别字允许您重新定义清单方块的大小，也允许您在清单方块父视窗的显示区域中移动清单方块。

清单方块的宽度应该能够容纳最长字串的宽度加上卷动列的宽度。您可以使用：

```
GetSystemMetrics (SM_CXVSCROLL) ;
```

来获得垂直卷动列的宽度。您用一个字元的高度乘以想要在视埠中显示的项目数来计算出清单方块的高度。

将字串放入清单方块

建立清单方块之後，下一步是将字串放入其中，您可以通过呼叫 `SendMessage` 为清单方块视窗讯息处理程式发送讯息来做到这一点。字串通常通过以 0 开始计数的索引数来引用，其中 0 对应於最顶上的项目。在下面的例子

中, hwndList 是子视窗清单方块控制项的代号, 而 iIndex 是索引值。在使用 SendMessage 传递字串的情况下, lParam 参数是指向以 null 字元结尾字串的指标。

在大多数例子中, 当视窗讯息处理程式储存的清单方块内容超过了可用记忆体空间时, SendMessage 将传回 LB_ERRSPACE (定义为-2)。如果是因为其他原因而出错, 那么 SendMessage 将传回 LB_ERR (-1)。如果操作成功, 那么 SendMessage 将传回 LB_OKAY (0)。您可以通过测试 SendMessage 的非零值来判断这两种错误。

如果您采用 LBS_SORT 样式 (或者如果您在清单方块中按照想要呈现的顺序排列字串), 那么填入清单方块最简单的方法是借助 LB_ADDSTRING 讯息:

```
SendMessage (hwndList, LB_ADDSTRING, 0, (LPARAM) szString) ;
```

如果您没有采用 LBS_SORT, 那么可以使用 LB_INSERTSTRING 指定一个索引值, 将字串插入到清单方块中:

```
SendMessage (hwndList, LB_INSERTSTRING, iIndex, (LPARAM) szString) ;
```

例如, 如果 iIndex 等於 4, 那么 szString 将变为索引值为 4 的字串——从顶头开始算起的第 5 个字串 (因为是从 0 开始计数的), 位於这个点後面的所有字串都将向後推移。索引值为-1 时, 将字串增加在最後。您可以对样式为 LBS_SORT 的清单方块使用 LB_INSERTSTRING, 但是这个清单方块的内容不能被重新排序 (您也可以使用 LB_DIR 讯息将字串插入到清单方块中, 这将在本章的最後进行讨论)。

您可以在指定索引值的同时使用 LB_DELETETESTRING 参数, 这就可以从清单方块中删除字串:

```
SendMessage (hwndList, LB_DELETETESTRING, iIndex, 0) ;
```

您可以使用 LB_RESETCONTENT 清除清单方块中的内容:

```
SendMessage (hwndList, LB_RESETCONTENT, 0, 0) ;
```

当在清单方块中增加或者删除字串时, 清单方块视窗讯息处理程式将更新显示。如果您有许多字串需要增加或者删除, 那么您也许希望暂时阻止这一动作, 其方法是关掉控制项的重画旗标:

```
SendMessage (hwndList, WM_SETREDRAW, FALSE, 0) ;
```

当您完成後, 可以再打开重画旗标:

```
SendMessage (hwndList, WM_SETREDRAW, TRUE, 0) ;
```

使用 LBS_NOREDRAW 样式建立的清单方块开始时其重画旗标是关闭的。

选择和取得项

SendMessage 完成了下面所描述的任务之後, 通常传回一个值。如果出错, 那么这个值将被设定为 LB_ERR (定义为-1)。

当清单方块中放入一些项目之後，您可以弄清楚清单方块中有多少项目：

```
iCount = SendMessage (hwndList, LB_GETCOUNT, 0, 0) ;
```

其他一些呼叫对单项选择清单方块和多项选择清单方块是不同的。让我们先来看看单项选择清单方块。

通常，您让使用者在清单方块中选择条目。但是如果您想加亮显示一个内定选择，则可以使用：

```
SendMessage (hwndList, LB_SETCURSEL, iIndex, 0) ;
```

将 iParam 设定为-1 则取消所有选择。

您也可以根据项目的第一个字母来选择：

```
iIndex = SendMessage (hwndList, LB_SELECTSTRING, iIndex,  
                      (LPARAM) szSearchString) ;
```

在 SendMessage 呼叫中将 iIndex 作为 iParam 参数时，iIndex 是索引，可以根据它搜索其开头字元与 szSearchString 相匹配的项目。iIndex 的值等於-1 时从头开始搜索，SendMessage 传回被选中项目的索引。如果没有开头字元与 szSearchString 相匹配的项目时，SendMessage 传回 LB_ERR。

当您得到来自清单方块的 WM_COMMAND 讯息时（或者在任何其他时候），您可以使用 LB_GETCURSEL 来确定目前选项的索引：

```
iIndex = SendMessage (hwndList, LB_GETCURSEL, 0, 0) ;
```

如果没有项目被选中，那么从呼叫中传回的 iIndex 值为 LB_ERR。

您可以确定清单方块中字串的长度：

```
iLength = SendMessage (hwndList, LB_GETTEXTLEN, iIndex, 0) ;
```

并可以将某项目复制到文字缓冲区中：

```
iLength = SendMessage (      hwndList, LB_GETTEXT, iIndex,  
                        (LPARAM) szBuffer) ;
```

在这两种情况下，从呼叫传回的 iLength 值是字串的长度。对以 NULL 字元终结的字串长度来说，szBuffer 阵列必须够大。您也许想用 LB_GETTEXTLEN 先分配一些局部记忆体来存放字串。

对于一个多项选择清单方块，您不能使用 LB_SETCURSEL、LB_GETCURSEL 或者 LB_SELECTSTRING，但是您可以使用 LB_SETSEL 来设定某特定项目的选择状态，而不影响有可能被选择的其他项：

```
SendMessage (hwndList, LB_SETSEL, wParam, iIndex) ;
```

wParam 参数不为 0 时，选择并加亮某一项；wParam 为 0 时，取消选择。如果 wParam 等於-1，那么将选择所有项目或者取消所有被选中的项目。您可以如下确定某特定项目的选择状态：

```
iSelect = SendMessage (hwndList, LB_GETSEL, iIndex, 0) ;
```

其中，如果由 iIndex 指定的项目被选中，iSelect 被设为非 0，否则被设为 0。

接收来自清单方块的信息

当使用者用鼠标单击清单方块时，清单方块将接收输入焦点。下面的操作可以使父视窗将输入焦点转交给清单方块控制项：

```
SetFocus (hwndList) ;
```

当清单方块拥有输入焦点时，光标移动键、字母键和 Spacebar 键都可以用来在该清单方块中选择某项。

清单方块控制项向其父视窗发送 WM_COMMAND 讯息，对按钮和编辑控制项来说，wParam 和 lParam 变数的含义是相同的：

LOWORD (wParam)	子视窗 ID
HWORD (wParam)	通知码
lParam	子视窗代号

通知码及其值如下所示：

LBN_ERRSPACE	-2
LBN_SELCHANGE	1
LBN_DBLCLK	2
LBN_SELCANCEL	3
LBN_SETFOCUS	4
LBN_KILLFOCUS	5

只有清单方块视窗样式包括 LBS_NOTIFY 时，清单方块控制项才会向父视窗发送 LBN_SELCHANGE 和 LBN_DBLCLK。

LBN_ERRSPACE 表示清单方块已经超出执行空间。LBN_SELCHANGE 表示目前选择已经被改变。这些讯息出现在下列情况下：使用者在清单方块中移动加亮的项目时，使用者使用 Spacebar 键切换选择状态或者使用鼠标单击某项时。LBN_DBLCLK 说明某项目已经被鼠标双击（LBN_SELCHANGE 和 LBN_DBLCLK 通知码的值表示鼠标按下的次数）。

根据应用的需要，您也许要使用 LBN_SELCHANGE 或 LBN_DBLCLK，也许二者都要使用。您的程式会收到许多 LBN_SELCHANGE 讯息，但是 LBN_DBLCLK 讯息只有当使用者双击鼠标时才会出现。如果您的程式使用双击，那么您需要提供一个复制 LBN_DBLCLK 的键盘介面。

一个简单的清单方块应用程序

既然您知道了如何建立清单方块，如何使用文字项目填入清单方块，如何接收来自清单方块的控制项以及如何取得字串，现在是到了写一个应用程序的时候了。如程式 9-5 中所示，ENVIRON 程式在显示区域中使用清单方块来显示目

前作业系统环境变数（例如 PATH 和 WINDIR）。当您选择一个环境变数时，其内容将显示在显示区域的顶部。

程式 9-5 ENVIRON

```
ENVIRON.C
/*-----
    ENVIRON.C -- Environment List Box
                    (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
#define ID_LIST      1
#define ID_TEXT      2

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Environ") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                        szAppName,
MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Environment List Box"),
                          WS_OVERLAPPEDWINDOW,
```

```

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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void FillListBox (HWND hwndList)
{
    int    iLength ;
    TCHAR * pVarBlock, * pVarBeg, * pVarEnd, * pVarName ;

    pVarBlock = GetEnvironmentStrings () ; // Get pointer to environment
block

    while (*pVarBlock)
    {
        if (*pVarBlock != '=')           // Skip variable names beginning with
'='
        {
            pVarBeg = pVarBlock ;        // Beginning of variable name
            while (*pVarBlock++ != '=') ; // Scan until '='
            pVarEnd = pVarBlock - 1 ;     // Points to '=' sign
            iLength = pVarEnd - pVarBeg ; // Length of variable name

            // Allocate memory for the variable name and terminating
            // zero. Copy the variable name and append a zero.

            pVarName = calloc (iLength + 1, sizeof (TCHAR)) ;
            CopyMemory (pVarName, pVarBeg, iLength * sizeof (TCHAR)) ;
            pVarName[iLength] = '\0' ;

            // Put the variable name in the list box and free memory.
            SendMessage (hwndList, LB_ADDSTRING, 0, (LPARAM) pVarName) ;
            free (pVarName) ;
        }
        while (*pVarBlock++ != '\0') ; // Scan until terminating zero
    }
    FreeEnvironmentStrings (pVarBlock) ;
}

```

```

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static HWND hwndList, hwndText ;
    int          iIndex, iLength, cxChar, cyChar ;
    TCHAR        *      pVarName, * pVarValue ;

    switch (message)
    {
    case WM_CREATE :
        cxChar = LOWORD (GetDialogBaseUnits ()) ;
        cyChar = HIWORD (GetDialogBaseUnits ()) ;
                                // Create listbox and static text windows.

        hwndList = CreateWindow (TEXT ("listbox"), NULL,
            WS_CHILD | WS_VISIBLE | LBS_STANDARD,
            cxChar, cyChar * 3,
            cxChar * 16 + GetSystemMetrics (SM_CXVSCROLL),
            cyChar * 5,
            hwnd, (HMENU) ID_LIST,
            (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
                                NULL) ;

        hwndText = CreateWindow (TEXT ("static"), NULL,
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            cxChar, cyChar,
            GetSystemMetrics (SM_CXSCREEN), cyChar,
            hwnd, (HMENU) ID_TEXT,
            (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
                                NULL) ;

        FillListBox (hwndList) ;
        return 0 ;

    case WM_SETFOCUS :
        SetFocus (hwndList) ;
        return 0 ;

    case WM_COMMAND :
        if (LOWORD (wParam) == ID_LIST && HIWORD (wParam) ==
LBN_SELCHANGE)
        {
                                // Get current selection.

            iIndex = SendMessage (hwndList, LB_GETCURSEL, 0, 0) ;
            iLength = SendMessage (hwndList, LB_GETTEXTLEN, iIndex, 0) + 1 ;
            pVarName = calloc (iLength, sizeof (TCHAR)) ;
            SendMessage (hwndList, LB_GETTEXT, iIndex, (LPARAM) pVarName) ;

```

```

// Get environment string.

iLength = GetEnvironmentVariable (pVarName, NULL, 0) ;
pVarValue = calloc (iLength, sizeof (TCHAR)) ;
GetEnvironmentVariable (pVarName, pVarValue, iLength) ;

// Show it in window.

SetWindowText (hwndText, pVarValue) ;
free (pVarName) ;
free (pVarValue) ;
}
return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

ENVIRON 建立两个子视窗：一个是 LBS_STANDARD 样式的清单方块，另一个是 SS_LEFT 样式（置左对齐文字）的静态视窗。ENVIRON 使用函式 GetEnvironmentStrings 来获得一个指标，该指标指向存有全部环境变数名及其值的记忆体区块。ENVIRON 用 FillListBox 函式来分析此记忆体区块，并使用 LB_ADDSTRING 讯息来指定清单方块视窗讯息处理程式将每个字串放入清单方块中。

当您执行 ENVIRON 时，可以使用滑鼠或者键盘来选择环境变数。每次您改变选择时，清单方块都会给其父视窗 WndProc 发送一个 WM_COMMAND 讯息。当 WndProc 收到 WM_COMMAND 讯息时，它就检查 wParam 的低字组是否为 ID_LIST（清单方块的子视窗 ID）和 wParam 的高字组（通知码）是否等於 LBN_SELCHANGE。如果是的，那么它就使用 LB_GETCURSEL 讯息来获得选中项目的索引，并使用 LB_GETTEXT 来获得外部环境变数名的字串本身。ENVIRON 程式使用 C 语言函式 GetEnvironmentVariable 来获得与变数相对应的环境字串，使用 SetWindowText 将该字串传递到静态子视窗控制项中，这个静态子视窗控制项被用来显示文字。

档案列表

我将最好的留在最後：LB_DIR，这是功能最强的清单方块讯息。它用档案目录列表填入清单方块，并且可以选择将子目录和有效的磁碟机也包括进来：

```
SendMessage (hwndList, LB_DIR, iAttr, (LPARAM) szFileSpec) ;
```

使用档案属性码

iAttr 参数是档案属性代码，其最低位元组是档案属性代码，该代码可以是表 9-6 资料的组合：

表 9-6

iAttr	值	属性
DDL_READWRITE	0x0000	普通档案
DDL_READONLY	0x0001	唯读档案
DDL_HIDDEN	0x0002	隐藏档案
DDL_SYSTEM	0x0004	系统档案
DDL_DIRECTORY	0x0010	子目录
DDL_ARCHIVE	0x0020	归档位元设立的档案

高位元组提供了一些对所要求项目的附加控制：

表 9-7

iAttr	值	属性
DDL_DRIVES	0x4000	包括磁碟机代号
DDL_EXCLUSIVE	0x8000	互斥搜索

字首 DDL 表示「对话目录列表」。

当 LB_DIR 讯息的 iAttr 值为 DDL_READWRITE 时，清单方块列出普通档案、唯读档案和归档位元设立的档案。当值为 DDL_DIRECTORY 时，清单方块除了列出上述档案之外，还列出子目录，目录位於中括号之内。当值为 DDL_DRIVES | DDL_DIRECTORY 时，那么列表将扩展到包括所有有效的磁碟机，而磁碟机代号显示在虚线之间。

将 iAttr 的最高位元设立就可以只列出符合条件的档案，而不包括其他档案。例如，对 Windows 的档案备份程式，也许您只想列出最後一次备份後修改过的档案，这种档案的归档位元设立，因此您可以使用 DDL_EXCLUSIVE | DDL_ARCHIVE。

档案列表的排序

lParam 参数是指向档案指定字符串如「*.」的指标，这个档案指定字符串不影响清单方块中的子目录。

您也许希望给列有档案清单的清单方块使用 LBS_SORT 讯息。清单方块首先列出符合档案指定要求的档案，再（可选择）列出子目录名。列出的第一个子目录名将采用下面的格式：

[. .]

这一个「两个点」的子目录项允许使用者向根目录回溯一层（在根目录下列出档案名时此项目不会出现）。最後，具体的子目录名称采用下面的形式：

[SUBDIR]

再来是以下列形式列出的有效磁碟机（也是可选择的）：

[-A-]

Windows 的 head 程式

UNIX 中有一个著名的实用程式叫做 head，它显示档案开始的几行。让我们使用清单方块为 Windows 编写一个类似的程式。如程式 9-6 所示，HEAD 将所有档案和子目录列在清单方块中。您可以挑选某个被选择的档案来显示，方法是在该档案上使用滑鼠双击或者使用 Enter 键按下要选的档案。您也可以使用这两种方法之一来改变子目录。这个程式在 HEAD 视窗显示区域的右边，从档案的开头开始显示，它最多能够显示 8 KB 的内容。

程式 9-6 HEAD

```
HEAD.C
/*-----
    HEAD.C -- Displays beginning (head) of file
              (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#define ID_LIST      1
#define ID_TEXT      2

#define MAXREAD      8192
#define DIRATTR      (DDL_READWRITE | DDL_READONLY | DDL_HIDDEN | DDL_SYSTEM | \
                      DDL_DIRECTORY | DDL_ARCHIVE | DDL_DRIVES)
#define DTFLAGS      (DT_WORDBREAK | DT_EXPANDTABS | DT_NOCLIP | DT_NOPREFIX)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ListProc (HWND, UINT, WPARAM, LPARAM) ;

WNDPROC OldList ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("head") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;
    wndclass.style     = CS_HREDRAW | CS_VREDRAW ;
```

```

    wndclass.lpfnWndProc      = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) (COLOR_BTNFACE + 1) ;
    wndclass.lpszMenuName    = NULL ;
    wndclass.lpszClassName   = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("head"),
                          WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static BOOL          bValidFile ;
    static BYTE          buffer[MAXREAD] ;
    static HWND          hwndList, hwndText ;
    static RECT          rect ;
    static TCHAR          szFile[MAX_PATH + 1] ;
    HANDLE               hFile ;
    HDC                  hdc ;
    int                  i, cxChar, cyChar ;
    PAINTSTRUCT          ps ;
    TCHAR                szBuffer[MAX_PATH + 1] ;
    switch (message)

```

```
{
case WM_CREATE :
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;

    rect.left = 20 * cxChar ;
    rect.top  = 3 * cyChar ;

    hwndList = CreateWindow (TEXT ("listbox"), NULL,
        WS_CHILDWINDOW | WS_VISIBLE | LBS_STANDARD,
        cxChar, cyChar * 3,
        cxChar * 13 + GetSystemMetrics (SM_CXVSCROLL),
        cyChar * 10,
        hwnd, (HMENU) ID_LIST,
        (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
        NULL) ;

    GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;

    hwndText = CreateWindow (TEXT ("static"), szBuffer,
        WS_CHILDWINDOW | WS_VISIBLE | SS_LEFT,
        cxChar, cyChar, cxChar * MAX_PATH, cyChar,
        hwnd, (HMENU) ID_TEXT,
        (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
        NULL) ;

    OldList = (WNDPROC) SetWindowLong (hwndList, GWL_WNDPROC,
        (LPARAM) ListProc) ;

    SendMessage (hwndList, LB_DIR, DIRATTR, (LPARAM) TEXT ("*..*")) ;
    return 0 ;

case WM_SIZE :
    rect.right      = LOWORD (lParam) ;
    rect.bottom     = HIWORD (lParam) ;
    return 0 ;

case WM_SETFOCUS :
    SetFocus (hwndList) ;
    return 0 ;

case WM_COMMAND :
    if (LOWORD (wParam) == ID_LIST && HIWORD (wParam) == LBN_DBLCLK)
    {
        if (LB_ERR == (i = SendMessage (hwndList, LB_GETCURSEL, 0, 0)))
            break ;

        SendMessage (hwndList, LB_GETTEXT, i, (LPARAM) szBuffer) ;
    }
}
```



```
if (INVALID_HANDLE_VALUE != (hFile = CreateFile (szBuffer,
    GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, 0, NULL)))
{
    CloseHandle (hFile) ;
    bValidFile = TRUE ;
    lstrcpy (szFile, szBuffer) ;
    GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;

    if (szBuffer [lstrlen (szBuffer) - 1] != '\\')
        lstrcat (szBuffer, TEXT ("\\")) ;
    SetWindowText (hwndText, lstrcat (szBuffer, szFile)) ;
}
else
{
    bValidFile = FALSE ;
    szBuffer [lstrlen (szBuffer) - 1] = '\\0' ;

    // If setting the directory doesn't work, maybe it's
    // a drive change, so try that.

    if (!SetCurrentDirectory (szBuffer + 1))
    {
        szBuffer [3] = ':' ;
        szBuffer [4] = '\\0' ;
        SetCurrentDirectory (szBuffer + 2) ;
    }

    // Get the new directory name and fill the list box.

    GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;
    SetWindowText (hwndText, szBuffer) ;
    SendMessage (hwndList, LB_RESETCONTENT, 0, 0) ;
    SendMessage (hwndList, LB_DIR, DIRATTR,
        (LPARAM) TEXT ("*. *")) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
}
return 0 ;

case WM_PAINT :
    if (!bValidFile)
        break ;

    if (INVALID_HANDLE_VALUE == (hFile = CreateFile (szFile,
        GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL)))
    {
        bValidFile = FALSE ;
        break ;
    }
```

```

    }

    ReadFile (hFile, buffer, MAXREAD, &i, NULL) ;
    CloseHandle (hFile) ;

    // i now equals the number of bytes in buffer.
    // Commence getting a device context for displaying text.

    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    SetTextColor (hdc, GetSysColor (COLOR_BTNTEXT)) ;
    SetBkColor (hdc, GetSysColor (COLOR_BTNFACE)) ;

    // Assume the file is ASCII

    DrawTextA (hdc, buffer, i, &rect, DTFLAGS) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ListProc (HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    if (message == WM_KEYDOWN && wParam == VK_RETURN)
        SendMessage (GetParent (hwnd), WM_COMMAND,
            MAKELONG (1, LBN_DBLCLK), (LPARAM) hwnd) ;
    return CallWindowProc (OldList, hwnd, message, wParam, lParam) ;
}

```

在 ENVIRON 中，当我们选择一个环境变数时——无论是使用滑鼠还是键盘——程式都将显示一个环境字串。但是，如果我们在 HEAD 中使用这种选择显示方法，那么程式回应会很慢，这是因为在清单方块中移动选择时，程式仍然要不断地打开和关闭档案。然而，HEAD 要求档案或者子目录被双击，从而引起一些问题，这是因为清单方块控制项没有滑鼠双击的自动键盘介面。前面讲过，如果可能，应该尽量提供键盘介面。

解决的方法是什么呢？当然是视窗子类别化。HEAD 中的清单方块子类则函数叫做 ListProc，它寻找 wParam 参数等於 VK_RETURN 的 WM_KEYDOWN 讯息，并给其父视窗发送一条带有 LBN_DBLCLK 通知码的 WM_COMMAND 讯息。在 WndProc

中，对 WM_COMMAND 的处理使用了 Windows 函式的 CreateFile 来检查清单方块中的选择。如果 CreateFile 传回一个错误资讯，则表示该选择不是档案，而可能是一个子目录。然後 HEAD 使用 SetCurrentDirectory 来改变这个子目录。如果 SetCurrentDirectory 不能执行，程式将假定使用者已经选择了一个磁碟机代号。改变磁碟机也需要呼叫 SetCurrentDirectory，作为该函式参数的字串则为是选择字串中拿掉开头的斜线，并加上一个冒号。它向清单方块发送一条 LB_RESETCONTENT 讯息来清除其中的内容，再发送一条 LB_DIR 讯息，使用新子目录中的档案来填入清单方块。

WndProc 中的 WM_PAINT 讯息是用 Windows 的 CreateFile 函式来打开档案的，这将传回一个档案代号，该代号可以传递给 Windows 的 ReadFile 和 CloseHandle 函式。

现在，在本章中，我们第一次碰到这个问题：Unicode。我们所希望最完美的方式大概就是让作业系统辨认文字档案的种类，使 ReadFile 能将 ASCII 档案转换成 Unicode 文字，或者将 Unicode 档案转换成 ASCII 文字。但现实并非如此完美。ReadFile 的功能只是读取档案中未经转换的位元组，也就是说，DrawTextA（在编译好的可执行档中没有定义 UNICODE 识别字）会把文字解释为 ASCII，而 DrawTextW（Unicode 版）会假设文字是 Unicode 的。

因此程式真正应该做的是去判别档案所包含的是 ASCII 文字还是 Unicode 文字，然後再恰当地呼叫 DrawTextA 或者 DrawTextW。实际上，HEAD 采用一个比较简单的方式，它只呼叫了 DrawTextA。

第十章 功能表及其他资源

大多数 Windows 程式都包含一个自订的图示, Windows 将该图示显示在应用程式视窗标题列的左上角。当程式被列在「开始」功能表中, 被显示在萤幕底部的工作列中, 被列在 Windows Explorer 中, 或者作为快捷方式显示在桌面上时, Windows 也显示该程式的图示。有些程式——大部分是像小画家一类的图形绘制工具——也使用自订滑鼠游标来表示程式的不同操作。还有许多 Windows 程式使用功能表和对话方块。功能表、对话方块加上卷动列, 这是标准 Windows 使用者界面的卖点。

图示、游标、功能表和对话方块都是相互关联的, 它们是 Windows 的全部资源型态。资源即资料, 它们被储存在程式的 .EXE 档案中, 但是它们并非驻留在程式的资料区域中。也就是说, 资源不能从程式原始码中定义的变数直接存取, Windows 提供函式直接或间接地把它们载入记忆体以备使用。我们已经遇到了两个这样的函式, 即 LoadIcon 和 LoadCursor, 它们出现在范例程式, 定义视窗类别结构的内容设定叙述中。它们从 Windows 中载入二进位图示和游标映象, 并传回该图示或游标的代号。在本章中, 我们先建立自己的图示, 它会从程式自己的 .EXE 档案中载入。

在本书中, 我们将讨论这些资源:

- 图示
- 游标
- 字串
- 自订资源
- 功能表
- 键盘加速键
- 对话方块
- 点阵图

前六个资源在本章讨论, 对话方块在第十一章讨论, 而点阵图在第十四章讨论。

图示、游标、字串和自订资源

使用资源的好处之一, 在於程式的许多元件能够连结编译进程式的 .EXE 档案中。如果没有资源这一个概念, 如图示图像之类的二进位档案可能会存放在单独的档案中, .EXE 会把它读入记忆体中使用。或者图示不得不在程式中以位

元组阵列的形式定义（这样就无法看到实际的图示图像了）。作为资源，图示储存在开发者电脑上可单独编辑的档案中，但在编译程序中被连结编译进 EXE 档案中。

将图示添加到程式

将资源添加到程式中需要 Visual C++ Developer Studio 的一些附加功能。对於图示来说，可以使用「Image Editor」（也称为「Graphics Editor」）来绘制图示的图像。该图像被储存在副档名为 .ICO 的图示档案中。Developer Studio 还产生一个资源描述档（副档名为 .RC 的档案，有时也称作资源定义档案），它列出了程式的所有资源和一个让程式引用资源的表头档案（RESOURCE.H）。

因此，您可以看到这些新档案是如何组织在一起的，让我们以建立名为 ICONDEMO 的新专案开始。像往常一样，在 Developer Studio 中从 **File** 功能表中选择 **New**，然後依次选择 **专案** 页面标签和 **Win32 Application**。在 **Project Name** 栏中键入 **ICONDEMO** 并单击 **OK**。这时，Developer Studio 建立了用於支援工作区和专案的五个档案。这些档案包括文字档案 ICONDEMO.DSW、ICONDEMO.DSP 和 ICONDEMO.MAK（假设当您从 **Tools** 功能表选择 **Open** 後，在显示的 **Open** 对话方块中，从 **Build** 页面标签中选 **Export makefile when saving project file**）。现在，让我们像通常那样所做的建立 C 原始码档案。从 **File** 功能表上选择 **New**，选择 **Files** 页面标签，并单击 **C++Source File**。在 **File Name** 栏中键入 **ICONDEMO.C** 并单击 **OK**。此时，Developer Studio 就建立了一个空的 **ICONDEMO.C** 档案。键入程式 10-1 中的程式，或选择 **Insert** 功能表，然後选择 **File As Text** 选项，从本书附上的光碟中复制原始码。

程式 10-1 ICONDEMO

```

ICONDEMO.C
/*-----
    ICONDEMO.C --          Icon Demonstration Program
                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{

```

```

TCHAR          szAppName[]          = TEXT ("IconDemo") ;
HWND           hwnd ;
MSG            msg ;
WNDCLASS       wndclass ;

wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc     = WndProc ;
wndclass.cbClsExtra      = 0 ;
wndclass.cbWndExtra      = 0 ;
wndclass.hInstance      = hInstance ;
wndclass.hIcon           = LoadIcon (hInstance, MAKEINTRESOURCE
(IDI_ICON)) ;
wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground  = GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName    = NULL ;
wndclass.lpszClassName  = szAppName ;
if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Icon Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HICON hIcon ;
    static int   cxIcon, cyIcon, cxClient, cyClient ;
    HDC          hdc ;

```

```

HINSTANCE    hInstance ;
PAINTSTRUCT  ps ;
int          x, y ;

switch (message)
{
case WM_CREATE :
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
        hIcon      = LoadIcon (hInstance, MAKEINTRESOURCE
(IDI_ICON)) ;
        cxIcon      = GetSystemMetrics (SM_CXICON) ;
        cyIcon      = GetSystemMetrics (SM_CYICON) ;
        return 0 ;

case WM_SIZE :
        cxClient    = LOWORD (lParam) ;
        cyClient    = HIWORD (lParam) ;
        return 0 ;

case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;

        for (y = 0 ; y < cyClient ; y += cyIcon)
                for (x = 0 ; x < cxClient ; x += cxIcon)
                        DrawIcon (hdc, x, y, hIcon) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

如果您试著编译该程式,因为在程式开头引用的 RESOURCE.H 档案并不存在,所以会产生错误。然而,您不必直接建立 RESOURCE.H 档案,而是由 Developer Studio 为您建立一个。

您可以通过将资源描述档添加到专案中来做到这一点。从「File」功能表中选择「New」,选择「Files」页面标签,单击「Resource Script」,在「File Name」栏中键入「ICONDEMO」,单击 OK。此时,Developer Studio 会建立两个文字档案:ICONDEMO.RC (资源描述档)和 RESOURCE.H (允许 C 原始码档案和资源描述档引用相同的已定义识别字)。不必直接编辑这两个档案,只要让 Developer Studio 来维护它们就可以。如果您想查看资源描述档和 RESOURCE.H

而不希望对 Developer Studio 产生干扰，可以用记事本打开它们。除非您对所做的动作很有把握，否则不要轻易地更改它们。请记住，只有在您下达明确的操作命令或重新编译专案时，Developer Studio 才会储存这些档案的新版本。

资源描述档是文字档案。它包括这些资源的可用文字形式表达的描述，例如功能表和对话方块。资源描述档也包括对非文字资源的二进位档案的引用，例如图示和自订的滑鼠游标。

现在，已经存在 RESOURCE.H 档案，您可以试著重新编译一下 ICONDEMO。现在会出现一条错误讯息，指出 IDI_ICON 还没被定义。这个识别字第一次出现在下面的叙述中：

```
wndclass.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

在本书前面的程式中，这个叙述是由下面的叙述代替的：

```
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
```

之所以改变叙述，是因为以前我们为应用程式使用的是标准的图示，而这里我们的目的是使用自订图示。

那么让我们建立一个图示吧！在 Developer Studio 的「File View」视窗中，您会看到两个档案——ICONDEMO.C 和 ICONDEMO.RC。您开启 CONDEMO.C 後，就可以编辑原始码。开启 ICONDEMO.RC 後，就可以把资源添加到档案中或编辑已存在的资源。要添加图示的话，请从「[Insert](#)」功能表上选择「[Resource](#)」选择您想添加的资源，也就是图示，然後再按下「[New](#)」按钮。

现在呈现的是一个空白的 32×32 图素的图示，您可以在其中填入颜色。您会看到带有一组绘图工具和可用颜色的浮动工具列。注意颜色工具列中包括两个与颜色无关的选项，这两种颜色选项有时被称为「萤幕颜色」跟「反萤幕颜色」。当一个图素在著色时选择了「萤幕颜色」时，它实际上是透明的。不管图示在什么表面上显示，图示未著色的部分会显示出底色。这样我们就可以建立非矩形的图示。

双击围绕图示的区域，会出现「Icon Properties」对话方块，该对话方块使您能够更改图示的 ID 和档案名称。Developer Studio 可能已经将 ID 设定为 IDI_ICON1，将它改为 IDI_ICON，这样 ICONDEMO 就可以引用图示（字首 IDI 代表「图示的 ID」）。同样地，将档案名改为 ICONDEMO.ICO。

现在选择一种有特色的颜色（如红色）并在图示上画一个大的 B（代表 BIG），请注意不必像图 10-1 那么整齐。

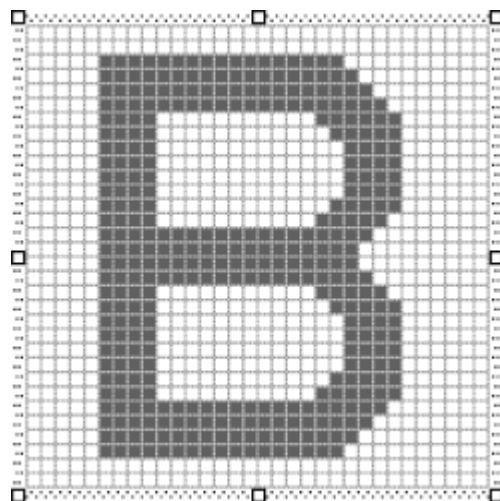


图 10-1 显示在 Developer Studio 中的标准 (32×32) ICONDEMO 档案

此时程式应该能够编译并执行得很好了。Developer Studio 将在 ICONDEMO.RC 资源描述档中划一条横线，表示下面是带有识别字 (IDI_ICON) 的图示档案 (ICONDEMO.ICO)。RESOURCE.H 表头档案中会包含 IDI_ICON 识别字的定义。

Developer Studio 通过资源编译器 RC.EXE 编译资源。文字资源描述档被转化为二进位形式，也就是具有副档名.RES 的档案。然後，该已编译的资源档案随同.OBJ 和.LIB 档案一起在 LINK 步骤中被指定连结。这就是资源被添加到最後产生出来的.EXE 档案中的方式。

当您执行 ICONDEMO 时，程式图示显示在标题列的左上角和工作列中。如果您将程式添加到「开始」功能表中，或在桌面上放置捷径，您也会在那儿看到该图示。

ICONDEMO 也在显示区域水平和垂直地重复显示该图示。程式使用叙述

```
hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

取得图示的代号。使用叙述

```
cxIcon = GetSystemMetrics (SM_CXICON) ;  
cyIcon = GetSystemMetrics (SM_CYICON) ;
```

取得图示的大小。然後，程式通过多次呼叫

```
DrawIcon (hdc, x, y, hIcon) ;
```

显示图示，其中 x 和 y 是被显示图示其左上角的座标。

在目前使用的大多数视讯显示卡上，带有 SM_CXICON 和 SM_CYICON 索引的 GetSystemMetrics 会回报图示的大小为 32×32 图素。这是我们在 Developer Studio 中建立的图示大小，它也是图示出现在桌面上和显示在 ICONDEMO 程式显示区域的大小。然而，这个大小并非显示在程式的标题列或工作列中的图示大小。小图示的大小可以由带有 SM_CXSMSIZE 和 SM_CYSMSIZE 索引的 GetSystemMetrics 获得（第一个 SM 表示「system metrics (系统度量)」，被包含的 SM 表示「small (小)」）。对于目前使用的大多数显示卡来说，小图

示的大小为 16×16 图素。

这会产生问题。当 Windows 将 32×32 的图示缩小为 16×16 的图示时，必需减少图素的行和列。这样，对于某些比较复杂的图示，就会失真。因此，我们应该为那些图像缩小就会变形的图示建立特殊的 16×16 图素的图示。在 Developer Studio 中图示图像的上面是标识为「Device」的下拉式清单方块，在它的右边有一个按钮，按下该按钮会弹出「New Icon Image」对话方块，此时选择「Small (16×16)」。现在您可以画另一个图示。如图 10-2 所示，画一个「S」（表示「小」）。

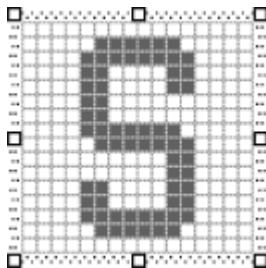


图 10-2 在 Developer Studio 中显示的小 (16×16) ICONDEMO 档案

在该程式中您不必做任何事情。第二个图示图像被储存在相同的 ICONDEMO.ICO 档案中，并以相同的 IDI_ICON 识别字引用。在适当的时候，Windows 会自动使用该较小的图示，例如在标题列或工作列中。当在桌面上显示快捷方式，以及程式呼叫 DrawIcon 装饰显示区域时，Windows 会使用大图示。

在掌握这些知识之后，让我们看一看使用图示的详细情况。

取得图示代号

如果您仔细阅读 ICONDEMO.RC 和 RESOURCE.H 档案，会看到由 Developer Studio 产生用于维护档案的一些标记。然而，当编译资源描述档时，只有少数几行是重要的。这些从 ICONDEMO.RC 和 RESOURCE.H 档案中摘录下来的关键部分被列在程式 10-2 中。

ICONDEMO.RC (摘录)

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Icon
IDI_ICON          ICON    DISCARDABLE    "icondemo.ico"
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by IconDemo.rc
```

```
#define IDI_ICON 101
```

程式 10-2 ICONDEMO.RC 和 RESOURCE.H 档案的摘录

程式 10-2 所显示的 ICONDEMO.RC 和 RESOURCE.H 档案与您在普通的文字编辑器中手动建立的很相似，80 年代的 Windows 程式写作者就是这样做的。唯一不同的是 AFXRES.H，它是个表头档案，包含了在建立由机器产生的 MFC 专案时由 Developer Studio 使用的常用识别字。在本书中，我们不会用到 AFXRES.H。

ICONDEMO.RC 中的这行

```
IDI_ICON ICON DISCARDABLE "icondemo.ico"
```

是资源描述档的 ICON 叙述。该图示有一个数值识别字 IDI_ICON，等於 101。由 Developer Studio 添加的 DISCARDABLE 关键字指出，必要时 Windows 可以从记忆体中丢弃图示，以获得额外的空间。之後不需要程式任何特定的操作，Windows 就能够重新载入图示。DISCARDABLE 属性是内定的，不需要指定。只有在名称和目录路径包含空格时，Developer Studio 才将档案名加上引号。

当资源编译程序将编译的资源储存在 ICONDEMO.RES 中，并且由连结程式将资源添加到 ICONDEMO.EXE 中以後，该资源就可以经由一个资源型态 (RT_ICON) 和一个识别字 (IDI_ICON 或 101) 来标识。程式可以通过呼叫 LoadIcon 函式取得此图示的代号：

```
hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

请注意 ICONDEMO 在两个地方呼叫这个函式，一次在定义视窗类别时，另一次在视窗讯息处理程式中取得图示的代号用於绘制。LoadIcon 传回 HICON 型态的值，它是图示的代号。

LoadIcon 的第一个参数，是指出资源来自哪个档案的执行实体代号。使用 hInstance 表示它来自程式自己的 .EXE 档案。LoadIcon 的第二个参数实际上被定义为指向字串的指标。待会将会看到，可以使用字串而不是用数值识别字标识资源。巨集 MAKEINTRESOURCE (把整数转换成资源字串) 生成指向非数字的指标，如下所示：

```
#define MAKEINTRESOURCE(i) (LPTSTR) ((DWORD) ((WORD) (i)))
```

LoadIcon 知道，如果第二个参数的高字组为 0，那么低字组就为图示的数值识别字。图示的识别字必须为 16 位元值。

本书前面的范例程式使用了预先定义的图示：

```
LoadIcon (NULL, IDI_APPLICATION) ;
```

hInstance 参数被设定为 NULL，因此 Windows 知道这是预先定义的图示。IDI_APPLICATION 也在 WINUSER.H 中用 MAKEINTRESOURCE 定义：

```
#define IDI_APPLICATION MAKEINTRESOURCE(32512)
```

LoadIcon 的第二个参数带来了一个有趣的问题：图示的识别字能可以为字串吗？答案是可以。方法如下：在 [Developer Studio](#) 中，在 [ICONDEMO](#) 专案

的档案列表上, 选择 **IDONDEMO.RC**。您会看到顶端为「IconDemo Resource」的树状结构, 然後是资源型态「Icon」, 再下来是「IDI_ICON」。如果用滑鼠右键单击图示识别字, 并从功能表上选择「**Properties**」, 您就能改变 ID。实际上, 您可以把名称放在引号内将其更改为字串。我用这种方法指定资源名称, 并在本书的其他地方也使用该方法。

我喜欢为图示 (以及一些其他资源) 使用文字名称, 因为名称可以是程式的名称。例如, 假定档案被命名为 MYPROG。如果您使用「Icon Properties」对话方块将图示的 ID 指定为「MyProg」(包括引号), 资源描述档将包含下列叙述:

```
MYPROG ICON DISCARDABLE myprog.ico
```

然而, 在 RESOURCE.H 中并没有#define 叙述, 来指出 MYPROG 是数值识别字。资源描述档将假定 MYPROG 是字串识别字。

在 C 程式中, 使用 LoadIcon 函式来取得图示代号。您可能已经有了表示程式名的字串:

```
static TCHAR szAppName [] = TEXT ("MyProg") ;
```

这意味著程式可以使用叙述:

```
hIcon = LoadIcon (hInstance, szAppName) ;
```

来载入图示, 这比巨集 MAKEINTRESOURCE 更清晰一些。

但是如果您确实想用数字来命名, 那么您可以用数字代替识别字或字串。在「Icon Properties」对话方块中, 在 ID 栏中输入数字。资源描述档将有一个类似下面的 ICON 叙述:

```
125 ICON DISCARDABLE myprog.ico
```

可以使用两种方法之一引用图示。明显易读的方式是:

```
hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (125)) ;
```

另一个不易阅读的方式是:

```
hIcon = LoadIcon (hInstance, TEXT ("#125")) ;
```

Windows 识别初始字元#作为 ASCII 形式中字元数值的开头。

在程式中使用图示

虽然 Windows 以几种方式用图示来代表程式, 但是许多 Windows 程式仅在用 WNDCLASS 结构和 RegisterClass 定义视窗类别时指定一个图示。如我们所看到的, 这样作用得很好, 尤其当图示档案包含标准和较小的图像大小时, 更是如此。Windows 在显示图示图像时, 它会在图示档案中选择最合适的图像大小。

RegisterClass 有一个改进版本叫做 RegisterClassEx, 它使用名为 WNDCLASSEX 的结构。WNDCLASSEX 有两个附加的栏位: cbSize 和 hIconSm。cbSize 栏位指出了 WNDCLASSEX 结构的大小, 假设 hIconSm 被设定为小图示的图示代号。

这样，在 WNDCLASSEX 结构中，您可以设定与两个图示档案相关的两个图示代号——一个用于标准图示，一个用于小图示。

有这种必要吗？没有。正如我们看到的，Windows 已经从单个图示档案中提取了大小合适的图示图像。RegisterClassEx 似乎没有 RegisterClass 聪明。如果 hIconSm 栏位使用了包含多个图像的图示档案，则只有第一个图像能被利用。它可能是标准大小的图示，使用时才被缩小。RegisterClassEx 似乎是为了使用多个图示图像而设计的，每个图像只包含一种图示大小。因为现在可以将多个图示大小包括在同一个图示档案中，所以我建议使用 WNDCLASS 和 RegisterClass。

如果您想在程式执行的时候，动态地更改程式的图示，可以使用 SetClassLong 来达到目的。例如，如果您有与识别字 IDI_ALTICON 相关的第二个图示档案，则您可以使用以下的叙述将其切换到那个图示：

```
SetClassLong (hwnd, GCL_HICON,
    LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ALTICON))) ;
```

如果不想储存程式图示的代号，但要使用 DrawIcon 函式在别处显示它，可以使用 GetClassLong 获得代号。例如：

```
DrawIcon (hdc, x, y, GetClassLong (hwnd, GCL_HICON)) ;
```

在 Windows 文件的某些部分，LoadIcon 被称为「过时的」，并推荐使用 LoadImage（LoadIcon 在 /Platform SDK/User Interface Services/Resources/Icons 中说明，LoadImage 在 /Platform SDK/User Interface Services/Resources/Resources 中说明）。当然 LoadImage 更为灵活，但它没有 LoadIcon 简单。您会注意到，在 ICONDEMO 中对同一个图示呼叫了 LoadIcon 两次。这不会产生问题，也没有使用额外的记忆体。LoadIcon 是取得代号但不需要清除代号的少数几个函式之一。实际上有一个 DestroyIcon 函式，但它与 CreateIcon、CreateIconIndirect 和 CreateIconFromResource 连在一起使用。这些函式使程式能够动态地建立图示图像。

使用自订游标

在程式中使用自订的滑鼠游标与使用自订的图示相似，只是大多数程式写作者总是使用 Windows 提供的游标。自订游标一般为单色，大小为 32 32 图素。在 Developer Studio 中建立游标与建立图示的方法相同（从「[Insert](#)」功能表上选择「[Resource](#)」，然后单击「[Cursor](#)」），但不要忘记定义热点。

可以在物件类别定义中设定自订游标，叙述为：

```
wndclass.hCursor = LoadCursor (hInstance, MAKEINTRESOURCE (IDC_CURSOR)) ;
```

如果游标用文字名称定义，则为：


```
wndclass.hCursor = LoadCursor (hInstance, szCursor) ;
```

每当滑鼠位於根据这个类别建立的视窗上时，就会显示与 IDC_CURSOR 或 szCursor 相对应的滑鼠标标。

如果使用了子视窗，那么您可能希望游标随著所在视窗的不同而有所区别。如果程式为这些子视窗定义了视窗类别，就可以在每个视窗类别中适当地设定 hCursor 栏位，让每个视窗类别使用不同的游标。如果使用了预先定义子视窗控制项，就可以使用以下方法改变视窗类别的 hCursor 栏位：

```
SetClassLong (hwndChild, GCL_HCURSOR,  
    LoadCursor (hInstance, TEXT ("childcursor")) ;
```

如果您将显示区域划分为较小的逻辑区域而不使用子视窗，就可以使用 SetCursor 来改变滑鼠标标：

```
SetCursor (hCursor) ;
```

在处理 WM_MOUSEMOVE 讯息处理期间，您应该呼叫 SetCursor；否则，当游标移动时，Windows 将使用视窗类别中定义的游标来重画游标。文件指出，如果没有改变游标，则 SetCursor 速度将会很快。

字串资源

把字串当成资源的观念一开始可能令人觉得诡异。因为我们在使用原始码中定义为变数的一般字串时，并没有碰到任何问题。

字串资源主要是为了让程式转换成其他语言时更为方便。正如後面两章中将看到的一样，功能表和对话方块也是资源描述档的一部分。如果使用字串资源而不是将字串直接放入原始码中，那么程式所使用的所有文字将在同一档案——资源描述档中。如果转换了资源描述档中的文字，那么建立程式的另一种语言版本所需做的一切就是重新连结程式。这种方法比重新组织原始码安全得多（然而，除了下一个范例程式，我在本书的其他程式中不使用字串表，原因是字串表使程式码看起来更为模糊和复杂）。

您可以在「 [Insert](#) 」功能表中选择「 [Resource](#) 」，再选择「 [String Table](#) 」，建立一个字串表。字串会显示在萤幕右边的列表中。通过双击字串就可以选中它。针对每个字串，您可以指定识别字和字串的内容。

在资源描述中，字串显示在一个多行的叙述中，如下所示：

```
STRINGTABLE DISCARDABLE  
BEGIN  
    IDS_STRING1, "character string 1"  
    IDS_STRING2, "character string 2"  
    其他字串定义  
END
```

如果您在替早期版本的 Windows 写程式，并在文字编辑器中手动建立这个

字符串表（用 Developer Studio 来做这件事当然更容易得多了），您可以用左右大括弧代替 BEGIN 和 END 叙述。

资源描述可以包含多个字符串表，但是每个 ID 必须唯一表示一个字符串。每个字符串占一行，最多 4097 个字节。\\t 可以作为跳位字节，\\n 则作为 linefeed 字节。DrawText 和 MessageBox 函式能够识别这些控制符号。

您的程式可以使用 LoadString 呼叫把字符串复制到程式资料段的缓冲区中：

```
LoadString (hInstance, id, szBuffer, iMaxLength) ;
```

参数 id 是 ID，它加在资源描述档中每个字符串的前面；szBuffer 是指向接收字符串的字节数组的指标；iMaxLength 是送入 szBuffer 中的最大字节数。函式传回字符串中的字节数。

每个字符串前面的 ID 一般是定义在表头档案中的巨集识别字。许多 Windows 程式写作者使用字首 IDS_ 来表示字符串的 ID。有时，档案名称或其他资讯需要在字符串显示时插入到字符串中。在这种情况下，您可以将 C 的格式化字节放入字符串，并把它用于 sprintf 中作为一个格式化字符串。

所有资源文字——包括字符串表中的文字——以 Unicode 格式储存在 .RES 编译资源档案以及最终的 .EXE 档案中。LoadStringW 函式直接载入 Unicode 文字。LoadStringA 函式（仅在 Windows 98 下有效）完成由 Unicode 到本地内码表的文字转换。

让我们来看一个程式，它使用三个字符串，在讯息方块中显示三条错误资讯。RESOURCE.H 表头档案为这些资讯定义了三个识别字：

```
#define IDS_FILENOTFOUND      1
#define IDS_FILETOOBIG       2
#define IDS_FILEREADONLY     3
```

资源描述档具有此字符串表：

```
STRINGTABLE
BEGIN
    IDS_FILENOTFOUND,          "File %s not found."
    IDS_FILETOOBIG,           "File %s too large to edit."
    IDS_FILEREADONLY,         "File %s is read-only."
END
```

C 原始码档案也包含这个表头档案，并定义了一个显示讯息方块的函式（我假定 szAppName 是一个包含程式名称的整体变数）。

```
OkMessage (HWND hwnd, int iErrorNumber, TCHAR *szFileName)
{
    TCHAR szFormat [40] ;
    TCHAR szBuffer [60] ;

    LoadString (hInst, iErrorNumber, szFormat, 40) ;
    sprintf (szBuffer, szFormat, szFilename) ;
```

```
return MessageBox (    hwnd, szBuffer, szAppName,
                      MB_OK | MB_ICONEXCLAMATION) ;
}
```

为了显示包含「file not found」资讯的讯息方块，程式呼叫：

```
OkMessage (hwnd, IDS_FILENOTFOUND, szFileName) ;
```

自订的资源

Windows 也定义了「自订资源」，这又称为「使用者定义的资源」（使用者就是您——程式写作者，而不是那个使用您程式的幸运者）。自订资源让连结.EXE 档案中的各种资料更为方便，对取得程式中的资料也是如此。资料可以是您需要的任何格式。程式用於存取自订资源的 Windows 函式促使 Windows 将资料载入记忆体并传回指向它的指标。然後您就可以对程式做任何操作。您会发现对於储存和存取各种自己的资料，这要比把资料储存在外部档案中，再使用档案输入函式存取它要方便得多。

例如，您有一个档案叫做 BINDATA.BIN，它包含程式需要显示的一些资料。您可以选择这个档案的格式。如果在 MYPROG 专案中有 MYPROG.RC 资源描述档，您就可以在 Developer Studio 中从「[Insert](#)」功能表中选择「[Resource](#)」并按「[Custom](#)」按钮，来建立自订的资源。键入表示资源的名称：例如，BINTYPE。然後，Developer Studio 会生成资源名称（在这种情况下是 IDR_BINTYPE1）并显示让您输入二进位资料的视窗。但是您不必输入什么，用滑鼠右键单击 IDR_BINTYPE1 名称，并选择 [Properties](#)，然後就可以输入一个档案名称：例如，BINDATA.BIN。

资源描述档就会包含以下的一行叙述：

```
IDR_BINTYPE1 BINTYPE BINDATA.BIN
```

除了我们刚刚生成的 BINTYPE 资源型态外，这个叙述与 ICONDEMO 中的 ICON 叙述一样。有了图示後，您可以对资源名称使用文字的名称，而不是数字的识别字。

当您编译并连结程式，整个 BINDATA.BIN 档案会被并入 MYPROG.EXE 档案中。

在程式的初始化（比如，在处理 WM_CREATE 讯息时）期间，您可以获得资源的代号：

```
hResource = LoadResource (    hInstance,
                             FindResource (    hInstance, TEXT ("BINTYPE"),
                                                MAKEINTRESOURCE
(IDR_BINTYPE1))) ;
```

变数 hResource 定义为 HGLOBAL 型态，它是指向记忆体区块的代号。不管它的名称是什么，LoadResource 不会立即将资源载入记忆体。把 LoadResource

和 FindResource 函式如上例般合在一起使用，在实质上就类似於 LoadIcon 和 LoadCursor 函式的做法。事实上，LoadIcon 和 LoadCursor 函式就用到了 LoadResource 和 FindResource 函式。

当您需要存取文字时，呼叫 LockResource：

```
pData = LockResource (hResource) ;
```

LockResource 将资源载入记忆体（如果还没有载入的话），然後它会传回一个指向资源的指标。当结束对资源的使用时，您可以从记忆体中释放它：

```
FreeResource (hResource) ;
```

当您的程式终止时，也会释放资源，即使您没有呼叫 FreeResource.。

让我们看一个使用三种资源——一个图示、一个字串表和一个自订的资源——的范例程式。程式 10-3 所示的 POEPOEM 程式在其显示区域显示 Edgar Allan Poe 的「Annabel Lee」文字。自订的资源是档案 POEPOEM.TXT，它包含了一段诗文，此文字档案以反斜线（\）结束。

程式 10-3 POEPOEM

```
POEPOEM.C
/*-----
-
POEPOEM.C -- Demonstrates Custom Resource
              (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    TCHAR          szAppName [16], szCaption [64], szErrMsg [64] ;
    HWND           hwnd ;
    MSG            msg ;
    WNDCLASS        wndclass ;

    LoadString (    hInstance, IDS_APPNAME, szAppName,
                    sizeof (szAppName) / sizeof (TCHAR)) ;

    LoadString (    hInstance, IDS_CAPTION, szCaption,
                    sizeof (szCaption) / sizeof (TCHAR)) ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
```

```

    wndclass.cbClsExtra          = 0 ;
    wndclass.cbWndExtra          = 0 ;
    wndclass.hInstance          = hInstance ;
    wndclass.hIcon               = LoadIcon (hInstance, szAppName) ;
    wndclass.hCursor             = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName        = NULL ;
    wndclass.lpszClassName       = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        LoadStringA (hInstance, IDS_APPNAME, (char *) szAppName,
                    sizeof (szAppName)) ;
        LoadStringA (hInstance, IDS_ERRMSG, (char *) szErrMsg,
                    sizeof (szErrMsg)) ;
        MessageBoxA (NULL, (char *) szErrMsg,
                    (char *) szAppName,
MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, szCaption,
        WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static char          * pText ;
    static HGLOBAL        hResource ;
    static HWND          hScroll ;
    static int           iPosition, cxChar, cyChar, cyClient,
iNumLines, xScroll ;
    HDC                   hdc ;
    PAINTSTRUCT           ps ;

```

```

RECT                                rect ;
TEXTMETRIC                          tm ;

switch (message)
{
case WM_CREATE :
    hdc                                = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar                            = tm.tmAveCharWidth ;
    cyChar                            = tm.tmHeight + tm.tmExternalLeading ;
    ReleaseDC (hwnd, hdc) ;

    xScroll                            = GetSystemMetrics (SM_CXVSCROLL) ;
    hScroll                            = CreateWindow (TEXT ("scrollbar"), NULL,
        WS_CHILD | WS_VISIBLE | SBS_VERT,
        0, 0, 0, 0,
        hwnd, (HMENU) 1, hInst, NULL) ;

    hResource = LoadResource (hInst,
        FindResource (hInst, TEXT ("AnnabelLee"),
            TEXT ("TEXT"))) ;

    pText = (char *) LockResource (hResource) ;
    iNumLines = 0 ;

    while (*pText != '\\\\' && *pText != '\\0')
    {
        if (*pText == '\\n')
            iNumLines ++ ;
        pText = AnsiNext (pText) ;
    }
    *pText = '\\0' ;

    SetScrollRange (hScroll, SB_CTL, 0, iNumLines, FALSE) ;
    SetScrollPos (hScroll, SB_CTL, 0, FALSE) ;
    return 0 ;

case WM_SIZE :
    MoveWindow (hScroll, LOWORD (lParam) - xScroll, 0,
        xScroll, cyClient = HIWORD (lParam), TRUE) ;
    SetFocus (hwnd) ;
    return 0 ;

case WM_SETFOCUS :
    SetFocus (hScroll) ;
    return 0 ;

case WM_VSCROLL :

```

```
switch (wParam)
{
case SB_TOP :
    iPosition = 0 ;
    break ;
case SB_BOTTOM :
    iPosition = iNumLines ;
    break ;
case SB_LINEUP :
    iPosition -= 1 ;
    break ;
case SB_LINEDOWN :
    iPosition += 1 ;
    break ;
case SB_PAGEUP :
    iPosition -= cyClient / cyChar ;
    break ;
case SB_PAGEDOWN :
    iPosition += cyClient / cyChar ;
    break ;
case SB_THUMBPOSITION :
    iPosition = LOWORD (lParam) ;
    break ;
}
iPosition = max (0, min (iPosition, iNumLines)) ;

if (iPosition != GetScrollPos (hScroll, SB_CTL))
{
    SetScrollPos (hScroll, SB_CTL, iPosition,
TRUE) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
}
return 0 ;

case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    pText = (char *) LockResource (hResource) ;

    GetClientRect (hwnd, &rect) ;
    rect.left += cxChar ;
    rect.top += cyChar * (1 - iPosition) ;
    DrawTextA (hdc, pText, -1, &rect, DT_EXTERNALLEADING) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
```

```

        FreeResource (hResource) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

POEPOEM.RC (摘录)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// TEXT
ANNABELLEE                TEXT    DISCARDABLE    "poepoem.txt"

////////////////////////////////////
/
// Icon
POEPOEM                    ICON    DISCARDABLE    "poepoem.ico"

////////////////////////////////////
/
// String Table
STRINGTABLE DISCARDABLE
BEGIN
    IDS_APPNAME            "PoePoem"
    IDS_CAPTION            "" "Annabel Lee" " by Edgar Allan Poe"
    IDS_ERRMSG              "This program requires Windows NT!"
END

```

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by PoePoem.rc

#define IDS_APPNAME    1
#define IDS_CAPTION    2
#define IDS_ERRMSG     3

```

POEPOEM.TXT

```

It was many and many a year ago,
    In a kingdom by the sea,
That a maiden there lived whom you may know
By the name of Annabel Lee;
And this maiden she lived with no other thought
    Than to love and be loved by me.
I was a child and she was a child
    In this kingdom by the sea,
But we loved with a love that was more than love --

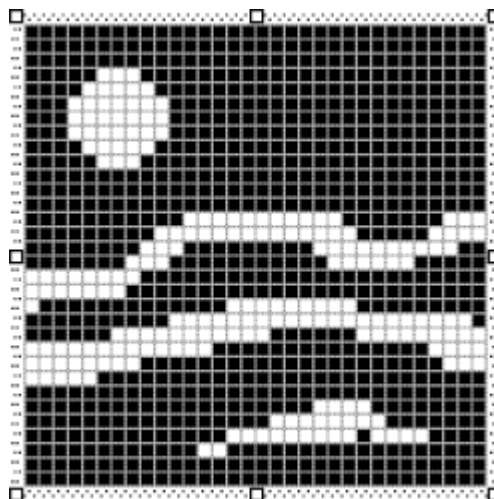
```

I and my Annabel Lee --
With a love that the winged seraphs of Heaven
Coveted her and me.
And this was the reason that, long ago,
In this kingdom by the sea,
A wind blew out of a cloud, chilling
My beautiful Annabel Lee;
So that her highborn kinsmen came
And bore her away from me,
To shut her up in a sepulchre
In this kingdom by the sea.
The angels, not half so happy in Heaven,
Went envying her and me --
Yes! that was the reason (as all men know,
In this kingdom by the sea)
That the wind came out of the cloud by night,
Chilling and killing my Annabel Lee.
But our love it was stronger by far than the love Of those who were older than
we -- Of many far wiser than we --
And neither the angels in Heaven above
Nor the demons down under the sea
Can ever dissever my soul from the soul
Of the beautiful Annabel Lee:
For the moon never beams, without bringing me dreams
Of the beautiful Annabel Lee;
And the stars never rise, but I feel the bright eyes
Of the beautiful Annabel Lee:
And so, all the night-tide, I lie down by the side
Of my darling -- my darling -- my life and my bride,
In her sepulchre there by the sea --
In her tomb by the sounding sea.

[May, 1849]

\

POEPOEM. ICO



在 POEPOEM.RC 资源描述档中, 使用者定义的资源被定义为 TEXT 型态, 取名为 AnnabelleLee:

```
ANNABELLEE TEXT POEPOEM.TXT
```

在 WndProc 处理 WM_CREATE 时, 使用 FindResource 和 LoadResource 取得资源代号。使用 LockResource 锁定资源, 并且使用一个小程式将档案末尾的反斜线 (\) 换成 0, 这有利於後面 WM_PAINT 讯息处理期间使用的 DrawText 函式。

注意, 这里使用的是子视窗的卷动列, 而不是视窗卷动列, 这是因为子视窗卷动列有一个自动的键盘介面, 因此在 POEPOEM 中没有处理 WM_KEYDOWN。

POEPOEM 还使用三个字串, 它们的 ID 在 RESOURCE.H 表头档案中定义。在程式的开始, IDS_APPNAME 和 IDS_CAPTIONPOEPOEM 字串由 LoadString 载入记忆体:

```
LoadString (hInstance, IDS_APPNAME, szAppName,      sizeof (szAppName) /
                                                    sizeof (TCHAR)) ;

LoadString (hInstance, IDS_CAPTION, szCaption,      sizeof (szCaption) /
                                                    sizeof (TCHAR)) ;
```

注意 RegisterClass 前面的两个呼叫。如果您在 Windows 98 下执行 Unicode 版本的 POEPOEM, 这两个呼叫就都会失败。因此, LoadStringA 比 LoadStringW 要复杂得多 (LoadStringA 必须将资源字串由 Unicode 转化为 ANSI, 而 LoadStringW 仅是直接载入它), LoadStringW 在 Windows 98 下不被支援。这意味著在 Windows 98 下, 当 RegisterClassW 函式失败时, MessageBoxW 函式 (Windows 98 支援) 就不能使用 LoadStringW 载入程式的字串。由於这个原因, 程式使用 LoadStringA 载入 IDS_APPNAME 和 IDS_ERRMSG 字串, 并使用 MessageBoxA 显示自订的讯息方块:

```
if (!RegisterClass (&wndclass))
{
    LoadStringA (hInstance, IDS_APPNAME, (char *) szAppName,
                  sizeof (szAppName)) ;
    LoadStringA (hInstance, IDS_ERRMSG, (char *) szErrMsg,
                  sizeof (szErrMsg)) ;
    MessageBoxA (NULL, (char *) szErrMsg,
                  (char *) szAppName, MB_ICONERROR) ;
    return 0 ;
}
```

注意, TCHAR 字串变数是指向 char 的指标。

既然我们已经定义了用於 POEPOEM 的所有字串资源, 那么翻译者将程式转换成外语版本就很容易了。当然, 它们将不得不翻译「Annabel Lee」这个名字——我想, 这会是一项困难得多的工作。

功能表

您还记得 Monty Python 有关乳酪店的幽默短剧吗？那故事内容是这样的：一个客人走进乳酪店想买某种乳酪。当然，店里没有这种乳酪。因此他又问有没有另一种乳酪，然後再问另一种，再问另一种，不断的问店家有没有另一种乳酪（最後总共问了 40 种的乳酪），回答仍然是没有，没有，没有，没有，没有。

这个不幸的事件可以通过功能表的使用来避免。一个功能表是一列可用的选项，它告诉饥饿的用餐者，厨房可以提供哪些服务，并且——对於 Windows 程式来说——还告诉使用者一个應用程式能够执行哪些操作。

功能表可能是 Windows 程式提供的一致使用者介面中最重要的部分，而在您的程式中增加功能表，是 Windows 程式设计中相对简单的部分。您在 Developer Studio 中定义功能表。每个可选的功能表项被赋予唯一的 ID。您在视窗类别结构中指定功能表名称。当使用者选择一个功能表项时，Windows 给您的程式发送包含该 ID 的 WM_COMMAND 讯息。

讨论完功能表後，我还将讨论键盘加速键，它们是一些键的组合，主要用於启动功能表功能。

功能表概念

视窗的功能表列紧接在标题列的下方显示，这个功能表列有时被称为「主功能表」或「顶层功能表」。列在顶层功能表的项目通常是下拉式功能表，也叫做「突现式功能表」或「子功能表」。您也可以定义多重嵌套的突现式功能表，也就是说，在突现式功能表上的项目可以存取另一个突现式功能表。有时突现式功能表上的项目呼叫对话方块以获得更多的资讯（对话方块在下一章介绍）。在标题列的最左端，很多父视窗都显示程式的小图示，这个图示可以启动系统功能表。它实际上是另一个突现式功能表。

突现式功能表的各项可以是「被选中的」，这意味著 Windows 在功能表文字的左端显示一个小的选中标记，选中标记让使用者知道从功能表中选中了哪些选项。这些选项之间可以是互斥的，也可以不互斥。顶层功能表项不能被选中。

顶层功能表或突现式功能表项可以被「启用」、「禁用」或「无效化」。「启动」和「不启动」有时候被当作「启用」和「禁用」的同义词。被启用或禁用的功能表项在使用者看来是一样的，但是无效化的功能表项是使用灰色文字来显示的。

从使用者的角度来看，启用、禁用和无效化的功能表项都是可以「选择的」（被选择的功能表项目会被加高亮度显示），也就是说，使用者可以使用滑鼠选择被禁用的功能表项，将反相显示游标列移动到禁用的功能表项上，或者使用功能表项的关键字母来选择该功能表项。然而，从程式写作者的角度来看，启用、禁用和无效化功能表项的功能是不同的。Windows 只为启用的功能表项向程式发送 WM_COMMAND 讯息。要让选项变得无效，可以把那些功能表项禁用和无效化。如果您想让使用者知道选择是无效的，那么您可以让一个功能表项无效化。

功能表结构

当您建立或改变程式中的功能表时，把顶层功能表和每一个突现式功能表想像成各自独立的功能表是有用的。顶层功能表有一个功能表代号，在顶层功能表中的每一个突现式功能表也有它自己的功能表代号。系统功能表（也是一个突现式功能表）也有功能表代号。

功能表中的每一项都有三个特性。第一个特性是功能表中显示什么，它可以是字串或点阵图。第二个特性是 WM_COMMAND 讯息中 Windows 发送给程式的功能表 ID，或者是在使用者选择功能表项时 Windows 显示的突现式功能表的代号。第三个特性是功能表项的属性，包括是否被禁用、无效化或被选中。

定义功能表

要使用 Developer Studio 来给程式资源描述档添加功能表，可以从 **Insert** 功能表中选择 **Resource** 并选择 **Menu**（或者您可能已经知道了）。然後，您可以用交谈式的方式定义功能表。功能表中每一项都有一个相关的 **Menu Item Properties** 对话方块，指出该项目的字串。如果选中了 **Pop-up** 核取方块，该项目就会呼叫一个突现式功能表，并且没有 ID 与此项目相联系。如果没有选中 **Pop-up** 核取方块，该项目被选中时就会产生带有特定 ID 的 WM_COMMAND 讯息。这两类功能表项分别出现在资源描述档的 POPUP 和 MENUITEM 叙述中。

当您为功能表中的项目键入文字时，可以键入一个「&」符号，指出後面一个字元在 Windows 显示功能表时要加底线。这种底线字元是在您使用 Alt 键选择功能表项时 Windows 要寻找的比对字元。如果在文字中不包括「&」符号，就不显示任何底线，Windows 会将功能表项文字的第一个字母用於 Alt 键查找。

如果在 **Menu Items Properties** 对话方块中选中 **Grayed** 选项，则功能表项是不能启动的，它的文字是灰色的，该项不产生 WM_COMMAND 讯息。如果选

中 **Inactive** 选项, 则功能表项也是不能启动的, 也不产生 WM_COMMAND 讯息, 但是它的文字显示正常。 **Checked** 选项在功能表项边上放置一个选中标记。 **Separator** 选项在突现式功能表上产生一个分栏的横线。

在突现式功能表的项目上, 可以在字串中使用跳位字元\t。紧接著\t 的文字被放置在距离突现式功能表的第一列右边新的一列上。在本章後面, 会看到在使用键盘加速键时它起的作用。字串中的\a 使跟著它的文字向右对齐。

您指定的 ID 值是 Windows 发送给视窗讯息处理程式中功能表讯息中的数值。在功能表中 ID 值应该是唯一的。按照惯例, 我使用以 IDM(「ID for a Menu」) 开头的识别字。

在程式中引用功能表

大多数 Windows 應用程式在资源描述档中只有一个功能表。您可以给功能表起一个与程式名称相同的文字的名称。程式写作者经常将程式名用於功能表名称, 以便相同的字串可以用於视窗类别、程式的图示名称和功能表名称。然後, 程式在视窗的定义中为功能表引用该名称:

```
wndclass.lpszMenuName = szAppName ;
```

虽然存取功能表资源的最常用方法是在视窗类别中指定功能表, 您也可以使用其他方法。Windows 應用程式可以使用 LoadMenu 函式将功能表资源载入记忆体中, 如同 LoadIcon 和 LoadCursor 函式一样。LoadMenu 传回一个功能表代号。如果您在资源描述档中为功能表使用了名称, 叙述如下:

```
hMenu = LoadMenu (hInstance, TEXT ("MyMenu")) ;
```

如果使用了数值, 那么 LoadMenu 呼叫采用如下的形式:

```
hMenu = LoadMenu (hInstance, MAKEINTRESOURCE (ID_MENU)) ;
```

然後, 您可以将这个功能表代号作为 CreateWindow 的第九个参数:

```
hwnd = CreateWindow ( TEXT ("MyClass"), TEXT ("Window Caption"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, hMenu, hInstance, NULL) ;
```

在这种情况下, CreateWindow 呼叫中指定的功能表可以覆盖视窗类别中指定的任何功能表。如果 CreateWindow 的第九个参数是 NULL, 那么您可以把视窗类别中的功能表看作是这种视窗类别的视窗内定使用的功能表。这样, 您可以为依据同一视窗类别建立的几个视窗使用不同的功能表。

您也可以在视窗类别中指定 NULL 功能表, 并且在 CreateWindow 呼叫中也指定 NULL 功能表, 然後在视窗被建立後再给视窗指定一个功能表:

```
SetMenu (hwnd, hMenu) ;
```

这种形式使您可以动态地修改视窗的功能表。在本章後面的 NOPOPUPS 程式

中我们将会看到这方面的例子。

当视窗被清除时，与视窗相关的所有功能表都将被清除。与视窗不相关的功能表在程式结束前通过呼叫 DestroyMenu 主动清除。

功能表和讯息

当使用者选择一个功能表项时，Windows 通常向视窗讯息处理程式发送几个不同的讯息。在大多数情况下，您的程式可以忽略大部分讯息，只需把它们传递给 DefWindowProc 即可。WM_INITMENU 就是这一类的讯息，它具有下列参数：

- wParam: 主功能表代号
- lParam: 0

wParam 值是您的主功能表代号，即使使用者选择的是系统功能表中的项目。Windows 程式通常忽略 WM_INITMENU 讯息。尽管在选中该项之前的讯息已经给程式提供了修改功能表的机会，但是我们觉得此刻改变顶层功能表是会扰乱使用者的。

程式也会接收到 WM_MENUSELECT 讯息。随著使用者在功能表项中移动游标或者滑鼠，程式会收到许多 WM_MENUSELECT 讯息。这对实作那些包含对功能表项的文字描述的状态列是很有帮助的。WM_MENUSELECT 的参数如下所示：

- LOWORD (wParam): 被选中项目：功能表 ID 或者突现式功能表代号
- HIWORD (wParam): 选择旗标
- lParam: 包含被选中项目的功能表代号

WM_MENUSELECT 是一个功能表追踪讯息，wParam 的值告诉您目前选择的是功能表中的哪一项（加高亮度显示的那个），wParam 的高字组中的「选择旗标」可以是下列这些旗标的组合：MF_GRAYED、MF_DISABLED、MF_CHECKED、MF_BITMAP、MF_POPUP、MF_HELP、MF_SYSMENU 和 MF_MOUSESELECT。如果您需要根据对功能表项的选择来改变视窗显示区域的内容，那么您可以使用 WM_MENUSELECT 讯息。许多程式把该讯息发送给 DefWindowProc。

当 Windows 准备显示一个突现式功能表时，它给视窗讯息处理程式发送一个 WM_INITMENUPOPUP 讯息，参数如下：

- wParam: 突现式功能表代号
- LOWORD (lParam): 突现式功能表索引
- HIWORD (lParam): 系统功能表为 1，其他为 0

如果您需要在显示突现式功能表之前启用或者禁用功能表项，那么这个讯息就很重要。例如，假定程式使用突现式功能表上的 **Paste** 命令从 **剪贴簿** 复制文字，当您收到突现式功能表中的 WM_INITMENUPOPUP 讯息时，应确定剪贴簿

内是否有文字存在。如果没有，那么应该使 **Paste** 功能表项无效化。我们将在本章後面修改的 POPPAD 程式中看到这样的例子。

最重要的功能表讯息是 WM_COMMAND，它表示使用者已经从功能表中选中了一个被启用的功能表项。第八章中的 WM_COMMAND 讯息也可以由子视窗控制项产生。如果您碰巧为功能表和子视窗控制项使用同一 ID 码，那么您可以通过 lParam 的值来区别它们，功能表项的 lParam 其值为 0，请参见表 10-1。

表 10-1

	功能表	控制项
LOWORD (wParam):	功能表 ID	控制项 ID
HIWORD (wParam):	0	通知码
lParam:	0	子视窗代号

WM_SYSCOMMAND 讯息类似於 WM_COMMAND 讯息，只是 WM_SYSCOMMAND 表示使用者从系统功能表中选择一个启用的功能表项：

- wParam: 功能表 ID
- lParam: 0

然而，如果 WM_SYSCOMMAND 讯息是由按滑鼠按键产生的，LOWORD (lParam) 和 HIWORD (lParam) 将包含滑鼠游标位置的 x 和 y 萤幕座标。

對於 WM_SYSCOMMAND，功能表 ID 指示系统功能表中的哪一项被选中。對於预先定义的系统功能表项，较低的那四个位元应该和 0xFFF0 进行 AND 运算来遮罩掉，结果值应该为下列之一：SC_SIZE、SC_MOVE、SC_MINIMIZE、SC_MAXIMIZE、SC_NEXTWINDOW、SC_PREVWINDOW、SC_CLOSE、SC_VSCROLL、SC_HSCROLL、SC_ARRANGE、SC_RESTORE 和 SC_TASKLIST。此外，wParam 可以是 SC_MOUSEMENU 或 SC_KEYMENU。

如果您在系统功能表中添加功能表项，那么 wParam 的低字组将是您定义的功能表 ID。为了避免与预先定义的功能表 ID 相冲突，應用程式应该使用小於 0xF000 的值，这對於将一般的 WM_SYSCOMMAND 讯息发送给 DefWindowProc 是很重要的。如果您不这样做，那么您实际上就是禁用了正常的系统功能表命令。

我们将讨论的最後一个讯息是 WM_MENUCHAR。实际上，它根本不是功能表讯息。在下列两种情况之一发生时，Windows 会把这个讯息发送到视窗讯息处理程式：如果使用者按下 Alt 和一个与功能表项不匹配的字元时，或者在显示突现式功能表而使用者按下一个与突现式功能表里的项目不匹配的字元键时。随 WM_MENUCHAR 讯息一起发送的参数如下所示：

- LOWORD (wParam): 字元代码 (ASCII 或 Unicode)
- HIWORD (wParam): 选择码

- lParam: 功能表代号

选择码是:

- 0 不显示突现式功能表
- MF_POPUP 显示突现式功能表
- MF_SYSMENU 显示系统突现式功能表

Windows 程式通常把该讯息传递给 DefWindowProc, 它一般给 Windows 传回 0, 这会使 Windows 发出哔声。在第十四章 GRAFMENU 程式中会看到 WM_MENUCHAR 讯息的使用。

范例程式

让我们来看一个简单的例子。程式 10-4 所示的 MENUDEMO 程式, 在主功能表中有五个选择项——File、Edit、Background、Timer 和 Help, 每一项都与一个突现式功能表相连。MENUDEMO 只完成了最简单、最通用的功能表处理操作, 包括拦截 WM_COMMAND 讯息和检查 wParam 的低字组。

程式 10-4 MENUDEMO

```
MENUDEMO.C
/*-----
MENUDEMO.C -- Menu Demonstration
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName[] = TEXT ("MenuDemo") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND      hwnd ;
    MSG       msg ;
    WNDCLASS  wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
```

```

    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName   = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Menu Demonstration"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int idColor [5] = {  WHITE_BRUSH, LTGRAY_BRUSH, GRAY_BRUSH,
                                DKGRAY_BRUSH, BLACK_BRUSH } ;
    static int iSelection = IDM_BKGND_WHITE ;
    HMENU      hMenu ;

    switch (message)
    {
    case WM_COMMAND:
        hMenu = GetMenu (hwnd) ;

        switch (LOWORD (wParam))
        {
            case IDM_FILE_NEW:
            case IDM_FILE_OPEN:
            case IDM_FILE_SAVE:
            case IDM_FILE_SAVE_AS:

```

```

        MessageBeep (0) ;
        return 0 ;

case  IDM_APP_EXIT:
        SendMessage (hwnd, WM_CLOSE, 0, 0) ;
        return 0 ;

case  IDM_EDIT_UNDO:
case  IDM_EDIT_CUT:
case  IDM_EDIT_COPY:
case  IDM_EDIT_PASTE:
case  IDM_EDIT_CLEAR:
        MessageBeep (0) ;
        return 0 ;

case  IDM_BKGND_WHITE:           // Note: Logic below
case  IDM_BKGND_LTGRAY:         // assumes that IDM_WHITE
case  IDM_BKGND_GRAY:           // through IDM_BLACK are
case  IDM_BKGND_DKGRAY:         // consecutive numbers in
case  IDM_BKGND_BLACK:          // the order shown here.

        CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;
        iSelection = LOWORD (wParam) ;
        CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;

        SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
                                GetStockObject
(idColor [LOWORD (wParam) - IDM_BKGND_WHITE])) ;

        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

case  IDM_TIMER_START:
        if (SetTimer (hwnd, ID_TIMER, 1000, NULL))
        {
        EnableMenuItem (hMenu, IDM_TIMER_START, MF_GRAYED) ;
        EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_ENABLED) ;
        }
        return 0 ;

case  IDM_TIMER_STOP:
        KillTimer (hwnd, ID_TIMER) ;
        EnableMenuItem (hMenu, IDM_TIMER_START, MF_ENABLED) ;
        EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_GRAYED) ;
        return 0 ;

case  IDM_APP_HELP:
        MessageBox (hwnd, TEXT ("Help not yet implemented!"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;

```

```

        return 0 ;

    case  IDM_APP_ABOUT:
        MessageBox (hwnd,TEXT ("Menu Demonstration Program\n")
            TEXT ("(c) Charles Petzold, 1998"),
            szAppName, MB_ICONINFORMATION | MB_OK) ;
        return 0 ;
    }
    break ;

    case  WM_TIMER:
        MessageBeep (0) ;
        return 0 ;

    case  WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

MENUDEMO.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
MENUDEMO MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New",                IDM_FILE_NEW
        MENUITEM "&Open",                IDM_FILE_OPEN
        MENUITEM "&Save",                IDM_FILE_SAVE
        MENUITEM "Save &As...",          IDM_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                IDM_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo",                IDM_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "C&ut",                IDM_EDIT_CUT
        MENUITEM "&Copy",                IDM_EDIT_COPY
        MENUITEM "&Paste",                IDM_EDIT_PASTE
        MENUITEM "De&lete",              IDM_EDIT_CLEAR
    END
END

```



```

    POPUP "&Background"
    BEGIN
        MENUITEM "&White",      IDM_BKGND_WHITE, CHECKED
        MENUITEM "&Light Gray", IDM_BKGND_LTGRAY
        MENUITEM "&Gray",       IDM_BKGND_GRAY
        MENUITEM "&Dark Gray",  IDM_BKGND_DKGRAY
        MENUITEM "&Black",     IDM_BKGND_BLACK
    END
    POPUP "&Timer"
    BEGIN
        MENUITEM "&Start",      IDM_TIMER_START
        MENUITEM "S&top",       IDM_TIMER_STOP, GRAYED
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help...",    IDM_APP_HELP
        MENUITEM "&About MenuDemo...", IDM_APP_ABOUT
    END
END

```

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by MenuDemo.rc

```

```

#define IDM_FILE_NEW          40001
#define IDM_FILE_OPEN         40002
#define IDM_FILE_SAVE         40003
#define IDM_FILE_SAVE_AS      40004
#define IDM_APP_EXIT          40005
#define IDM_EDIT_UNDO         40006
#define IDM_EDIT_CUT          40007
#define IDM_EDIT_COPY         40008
#define IDM_EDIT_PASTE        40009
#define IDM_EDIT_CLEAR        40010
#define IDM_BKGND_WHITE       40011
#define IDM_BKGND_LTGRAY      40012
#define IDM_BKGND_GRAY        40013
#define IDM_BKGND_DKGRAY      40014
#define IDM_BKGND_BLACK       40015
#define IDM_TIMER_START       40016
#define IDM_TIMER_STOP        40017
#define IDM_APP_HELP          40018
#define IDM_APP_ABOUT         40019

```

MENUDEMO.RC 资源描述档给了您定义功能表的提示。功能表的名称为「MenuDemo」。大多数项目有底线字母，这就是说您必须在字母前键入『&』。MENUITEM SEPARATOR 叙述是在「[Menu Item Properties](#)」对话方块中选中「[Separator](#)」框产生的。注意功能表中有一个项目具有「[Checked](#)」选项，

另一个具有「 **Grayed** 」选项。还有，「 **Background** 」突现式功能表中的五个项目应该按顺序输入，确保识别字是以数值的顺序，本程式需要这样。所有功能表项的识别字定义在 RESOURCE.H 中。

当收到突现式功能表「 **File** 」和「 **Edit** 」各项有关的 WM_COMMAND 讯息时，MENUDEMO 程式只使系统发出哔声。「 **Background** 」突现式功能表列出 MENUDEMO 用来给背景著色的五种现有画刷。在 MENUDEMO.RC 资源描述档中，「 **White** 」功能表项（功能表 ID 为 IDM_BKGND_WHITE）被标以「 **CHECKED** 」，它在功能表项旁边设定选中标记。在 MENUDEMO.C 中，iSelection 的值被初始化为 IDM_BKGND_WHITE。

「 **Background** 」突现式功能表上的五种画刷相互排斥。当 MENUDEMO.C 收到一个 WM_COMMAND 讯息，而该讯息中的 wParam 是「 **Background** 」突现式功能表上的五项之一时，它必须从先前选中的背景颜色中除掉选中标记，并把标记加到新的背景颜色上。为此，首先要得到功能表代号：

```
hMenu = GetMenu (hwnd) ;
```

CheckMenuItem 函式用来取消目前被选中的项目：

```
CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;
```

iSelection 的值被设定为 wParam 的值，新的背景颜色被选中：

```
iSelection = wParam ;
```

```
CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;
```

视窗类别中的背景颜色於是被替换为新的背景颜色，视窗显示区域变为无效状态，Windows 使用新的背景颜色清除视窗。

Timer 突现式功能表列出了两个选项——「Start」和「Stop」。开始时，「Stop」选项变为灰色的（就像在资源描述档中的功能表定义一样）。当您选择「Start」选项时，MENUDEMO 试图启动一个计时器，如果成功，则无效化「Start」选项，并启用「Stop」选项：

```
EnableMenuItem (hMenu, IDM_TIMER_START, MF_GRAYED) ;
```

```
EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_ENABLED) ;
```

当收到一条 WM_COMMAND 讯息，并且 wParam 等於 IDM_TIMER_STOP 时，MENUDEMO 程式会停止计数，启用「 **Start** 」项，然後无效化「 **Stop** 」选项：

```
EnableMenuItem (hMenu, IDM_TIMER_START, MF_ENABLED) ;
```

```
EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_GRAYED) ;
```

请注意，在计时器执行时，MENUDEMO 程式不可能收到 wParam 等於 IDM_TIMER_START 的 WM_COMMAND 讯息。同样地，在计时器关闭时，MENUDEMO 程式也不可能收到 wParam 等於 IDM_TIMER_STOP 的 WM_COMMAND 讯息。

当 MENUDEMO 收到一个 WM_COMMAND 讯息，而该讯息的参数 wParam 等於 IDM_APP_ABOUT 或 IDM_APP_HELP 时，MENUDEMO 程式显示一个讯息方块（在下一

章中，我们将把讯息方块变为对话方块）。

当 MENUDEMO 程式收到一个 WM_COMMAND 讯息，其参数 wParam 等於 IDM_APP_EXIT 时，它给自己发送一个 WM_CLOSE 讯息。这个讯息与 DefWindowProc 收到 WM_SYSCOMMAND 讯息且 wParam 等於 SC_CLOSE 时发送给视窗讯息处理程式的讯息相同。我们将在本章後面介绍 POPPAD2 时再仔细研究这个问题。

功能表设计规范

在 MENUDEMO 中的「File」和「Edit」突现式功能表的格式与其他 Windows 程式中的格式非常类似。Windows 的目的之一是为使用者提供一种易懂的介面，而不要求使用者为每个程式重新学习基本操作方式。如果「File」和「Edit」功能表在每个 Windows 程式中看起来都一样，并且都使用同样的字母和 Alt 键来进行选择，那么当然有助於减轻使用者的学习负担。

除了「File」和「Edit」突现式功能表外，大多数 Windows 程式的功能表都是不同的。当设计一个功能表时，您应该看一看现有的 Windows 程式以尽量保持一致。当然，如果您认为别的程式是不对的，而您知道正确的方法，那么没有人能够阻止您。同时记住，修改一个功能表，通常只需要修改资源描述档而不必修改您的程式码。即使以後要改变功能表项的位置，也不会有多大的问题。

虽然您的程式功能表在顶层可以有 MENUITEM 叙述，但这是不合规范的，因为这样会很容易导致错误的选择。如果您要这样做，那么请在字串後面加一个惊叹号，表示功能表项不会启动突现式功能表。

较难的一种功能表定义方法

在程式的资源描述档中定义功能表，通常是在您的视窗中添加功能表的最简单方法，但不是唯一的方法。如果您没有使用资源描述档，那么可以使用 CreateMenu 和 AppendMenu 两个函式在程式中建立功能表。在您定义完功能表後，您可以将功能表代号发送给 CreateWindow，或者使用 SetMenu 来设定视窗的功能表。

以下是具体的做法。CreateMenu 简单地把一个代号传回给新功能表：

```
hMenu = CreateMenu ();
```

功能表一开始为空。AppendMenu 将功能表项插入功能表中。您必须为顶层功能表项和每一个突现式功能表提供不同的功能表代号。突现式功能表是单独构成的，然後将突现式功能表代号插入顶层功能表。程式 10-5 中所示的程式码就是用这种方法建立功能表的，实际上，这个功能表与 MENUDEMO 程式中的功能

表相同。为了简化说明，代码使用 ASCII 字串。

程式 10-5 不使用资源描述档建立与 MENUDEMO 程式相同功能表的 C 程式码

```

hMenu = CreateMenu () ;
hMenuPopup = CreateMenu () ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_FILE_NEW, "&New");
AppendMenu      (hMenuPopup,      MF_STRING, IDM_FILE_OPEN, "&Open...");
AppendMenu      (hMenuPopup,      MF_STRING, IDM_FILE_SAVE, "&Save");
AppendMenu      (hMenuPopup,      MF_STRING, IDM_FILE_SAVE_AS, "Save &As...");
AppendMenu      (hMenuPopup,      MF_SEPARATOR, 0, NULL) ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_APP_EXIT, "E&xit") ;

AppendMenu      (hMenu, MF_POPUP, hMenuPopup, "&File") ;

hMenuPopup = CreateMenu () ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_EDIT_UNDO, "&Undo") ;
AppendMenu      (hMenuPopup,      MF_SEPARATOR, 0, NULL) ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_EDIT_CUT, "Cu&t") ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_EDIT_COPY, "&Copy") ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_EDIT_PASTE, "&Paste") ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_EDIT_CLEAR, "De&lete") ;
AppendMenu      (hMenu,      MF_POPUP,      hMenuPopup, "&Edit") ;

hMenuPopup = CreateMenu () ;
AppendMenu      (hMenuPopup,      MF_STRING| MF_CHECKED, IDM_BKGND_WHITE,
"&White");
AppendMenu      (hMenuPopup,      MF_STRING,      IDM_BKGND_LTGRAY, "&Light
Gray");
AppendMenu      (hMenuPopup,      MF_STRING,      IDM_BKGND_GRAY,
"&Gray") ;
AppendMenu      (hMenuPopup,      MF_STRING,      IDM_BKGND_DKGRAY, "&Dark
Gray");
AppendMenu      (hMenuPopup,      MF_STRING,      IDM_BKGND_BLACK, "&Black") ;

AppendMenu      (hMenu, MF_POPUP, hMenuPopup, "&Background") ;
hMenuPopup = CreateMenu () ;
AppendMenu      (hMenuPopup,      MF_STRING,      IDM_TIMER_START, "&Start") ;
AppendMenu      (hMenuPopup,      MF_STRING | MF_GRAYED, IDM_TIMER_STOP, "S&top") ;

AppendMenu      (hMenu,      MF_POPUP, hMenuPopup, "&Timer") ;

hMenuPopup =      CreateMenu () ;

AppendMenu      (hMenuPopup,      MF_STRING, IDM_HELP_HELP, "&Help") ;
AppendMenu      (hMenuPopup,      MF_STRING, IDM_APP_ABOUT, "&About
MenuDemo...") ;

AppendMenu      (hMenu, MF_POPUP, hMenuPopup, "&Help") ;

```

我认为您会同意底下这个观点：使用资源描述档功能表模板来制作功能表，会更容易而且更清楚。我并不鼓励您使用这里的方法定义功能表，而只是提供了一种实作功能表的方法。当然，您可以使用包含所有功能表项字串、ID 和旗标等的结构阵列来压缩程式码大小。不过，如果您这么做了，那么您还可以利用 Windows 定义功能表的第三种方法。LoadMenuIndirect 函式接受一个指向 MENUITEMTEMPLATE 型态的结构指标，并传回功能表的代号，该函式在载入资源描述档中的常规功能表模板後，在 Windows 中构造功能表，读者不妨自己尝试一下。

浮动突现式功能表

您还可以在没有顶层功能表列的情况下使用功能表，也就是说，您可以使突现式功能表出现在萤幕顶层的任何位置。一种方法是使用滑鼠右键来启动突现式功能表。程式 10-6 所示的 POPMENU 说明了这种方法。

程式 10-6 POPMENU

```
POPMENU.C
/*-----
   POPMENU.C -- Popup Menu Demonstration
                                   (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
TCHAR      szAppName[] = TEXT ("PopMenu") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, szAppName) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
```

```

    wndclass.lpszMenuName      = NULL ;
    wndclass.lpszClassName    = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hInst = hInstance ;
    hwnd = CreateWindow (  szAppName, TEXT ("Popup Menu Demonstration"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HMENU      hMenu ;
    static int        idColor [5] = {  WHITE_BRUSH,          LTGRAY_BRUSH,
GRAY_BRUSH,
                                DKGRAY_BRUSH, BLACK_BRUSH } ;
    static int        iSelection = IDM_BKGND_WHITE ;
    POINT
                        point ;

    switch (message)
    {
    case WM_CREATE:
        hMenu = LoadMenu (hInst, szAppName) ;
        hMenu = GetSubMenu (hMenu, 0) ;
        return 0 ;

    case WM_RBUTTONDOWN:
        point.x = LOWORD (lParam) ;
        point.y = HIWORD (lParam) ;
        ClientToScreen (hwnd, &point) ;

```

```

        TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y, 0,
hwnd, NULL) ;

        return 0 ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_FILE_NEW:
            case IDM_FILE_OPEN:
            case IDM_FILE_SAVE:
            case IDM_FILE_SAVE_AS:
            case IDM_EDIT_UNDO:
            case IDM_EDIT_CUT:
            case IDM_EDIT_COPY:
            case IDM_EDIT_PASTE:
            case IDM_EDIT_CLEAR:
                MessageBeep (0) ;
                return 0 ;

            case IDM_BKGND_WHITE:           // Note: Logic below
            case IDM_BKGND_LTGRAY:          // assumes that IDM_WHITE
            case IDM_BKGND_GRAY:            // through IDM_BLACK are
            case IDM_BKGND_DKGRAY:          // consecutive numbers in
            case IDM_BKGND_BLACK:           // the order shown here.

                CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;
                iSelection = LOWORD (wParam) ;
                CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;

                SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
                                GetStockObject
                                (idColor [LOWORD (wParam) - IDM_BKGND_WHITE])) ;

                InvalidateRect (hwnd, NULL, TRUE) ;
                return 0 ;

            case IDM_APP_ABOUT:
                MessageBox (hwnd, TEXT ("Popup Menu Demonstration
Program\n"),

                TEXT ("(c) Charles Petzold, 1998"),
                szAppName, MB_ICONINFORMATION | MB_OK) ;
                return 0 ;

            case IDM_APP_EXIT:
                SendMessage (hwnd, WM_CLOSE, 0, 0) ;
                return 0 ;

```

```

        case  IDM_APP_HELP:
            MessageBox (hwnd, TEXT ("Help not yet implemented!"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;
        }
        break ;

    case  WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

POPMENU.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////

/

// Menu

POPMENU MENU DISCARDABLE

BEGIN

POPUP "MyMenu"

BEGIN

POPUP "&File"

BEGIN

MENUITEM "&New", IDM_FILE_NEW

MENUITEM "&Open", IDM_FILE_OPEN

MENUITEM "&Save", IDM_FILE_SAVE

MENUITEM "Save &As", IDM_FILE_SAVE_AS

MENUITEM SEPARATOR

MENUITEM "E&xit", IDM_APP_EXIT

END

POPUP "&Edit"

BEGIN

MENUITEM "&Undo", IDM_EDIT_UNDO

MENUITEM SEPARATOR

MENUITEM "Cu&t", IDM_EDIT_CUT

MENUITEM "&Copy", IDM_EDIT_COPY

MENUITEM "&Paste", IDM_EDIT_PASTE

MENUITEM "De&lete", IDM_EDIT_CLEAR

END

POPUP "&Background"

BEGIN

MENUITEM "&White", IDM_BKGND_WHITE, CHECKED

MENUITEM "&Light Gray", IDM_BKGND_LTGRAY

MENUITEM "&Gray", IDM_BKGND_GRAY


```

MENUITEM "&Dark Gray",          IDM_BKGND_DKGRAY
MENUITEM "&Black",              IDM_BKGND_BLACK
END

    POPUP "&Help"
BEGIN
MENUITEM "&Help...",            IDM_APP_HELP
    MENUITEM "&About PopMenu...", IDM_APP_ABOUT
END
    END
END

```

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by PopMenu.rc

```

```

#define IDM_FILE_NEW          40001
#define IDM_FILE_OPEN        40002
#define IDM_FILE_SAVE        40003
#define IDM_FILE_SAVE_AS     40004
#define IDM_APP_EXIT         40005
#define IDM_EDIT_UNDO        40006
#define IDM_EDIT_CUT         40007
#define IDM_EDIT_COPY        40008
#define IDM_EDIT_PASTE       40009
#define IDM_EDIT_CLEAR       40010
#define IDM_BKGND_WHITE      40011
#define IDM_BKGND_LTGRAY    40012
#define IDM_BKGND_GRAY       40013
#define IDM_BKGND_DKGRAY    40014
#define IDM_BKGND_BLACK      40015
#define IDM_APP_HELP         40016
#define IDM_APP_ABOUT        40017

```

资源描述档 POPMENU.RC 定义的功能表与 MENUDEMO.RC 中的功能表非常相似。不同的是，在顶层功能表中只包含一项——一个突现式功能表「MyMenu」，它呼叫「File」、「Edit」、「Background」和「Help」选项。这四个选项垂直一行地出现在突现式功能表上，而不是水平一列地出现在主功能表上。

在 WndProc 中的 WM_CREATE 处理期间，POPMENU 取得此突现式功能表的代号，就是带有文字「MyMenu」的那个突现式功能表：

```

hMenu = LoadMenu (hInst, szAppName) ;
hMenu = GetSubMenu (hMenu, 0) ;

```

在 WM_RBUTTONDOWN 讯息处理期间，POPMENU 提供了滑鼠指标的位置，将此位置转换为萤幕座标，再将座标值传递给 TrackPopupMenu：

```

point.x = LOWORD (lParam) ;
point.y = HIWORD (lParam) ;
ClientToScreen (hwnd, &point) ;

```

```
TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y,
                0, hwnd, NULL) ;
```

然後, Windows 显示出具有「File」、「Edit」、「Background」和「Help」项的突现式功能表。选择其中任何一项都可以使嵌套的突现式功能表显示在右边, 功能表函式与一般的功能表一样。

如果要使用与该程式的主功能表相同的功能表并带有 TrackPopupMenu, 您会遇到一些问题, 因为函式需要突现式功能表代号。在「Microsoft Knowledge Base」文章 ID Q99806 有提供一些资讯。

使用系统功能表

使用 WS_SYSMENU 样式建立的父视窗, 在其标题列的左侧有一个系统功能表按钮。如果您愿意, 可以修改这个功能表。在 Windows 程式设计的早期, 程式写作者一般把「About」功能表项放入系统功能表。虽然这种方法不常见, 但是修改系统功能表往往是一种在短程式中添加功能表的快速偷懒方法。这里唯一的限制是: 在系统功能表中增加的命令其 ID 值必须小於 0xF000; 否则它们将会与 Windows 系统功能表命令所使用的 ID 值相冲突。还要记住, 当您为这些新功能表项在视窗讯息处理程式中处理 WM_SYSCOMMAND 讯息时, 您必须把其他的 WM_SYSCOMMAND 讯息发送给 DefWindowProc。如果您不这样做, 那么实际上是禁用了系统功能表上的所有正常选项。

程式 10-7 中所示的 POORMENU (「设计不当的个人功能表」) 在系统功能表中加入了一个分隔条和三个命令, 最後一个命令将删除这些附加的功能表项。

程式 10-7 POORMENU

```
POORMENU.C
/*-----
    POORMENU.C --          The Poor Person's Menu
                           (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
#define IDM_SYS_ABOUT      1
#define IDM_SYS_HELP      2
#define IDM_SYS_REMOVE    3

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
static TCHAR szAppName[] = TEXT ("PoorMenu") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdLine, int iCmdShow)
{
    HMENU      hMenu ;
    HWND       hwnd ;
```

```

MSG                                msg ;
WNDCLASS                          wndclass ;

wndclass.style                     = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc               = WndProc ;
wndclass.cbClsExtra                = 0 ;
wndclass.cbWndExtra               = 0 ;
wndclass.hInstance                = hInstance ;
wndclass.hIcon                    = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground            = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName             = NULL ;
wndclass.lpszClassName            = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("The Poor-Person's Menu"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

hMenu = GetSystemMenu (hwnd, FALSE) ;
AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_ABOUT, TEXT ("About...")) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_HELP, TEXT ("Help...")) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_REMOVE, TEXT ("Remove
Additions")) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM

```

```

lParam)
{
    switch (message)
    {
        case WM_SYSCOMMAND:
            switch (LOWORD (wParam))
            {
                case IDM_SYS_ABOUT:
                    MessageBox (    hwnd,          TEXT      ("A
Poor-Person's Menu Program\n")
                                TEXT ("(c) Charles Petzold, 1998"),
                                szAppName, MB_OK | MB_ICONINFORMATION) ;
                    return 0 ;

                case IDM_SYS_HELP:
                    MessageBox (    hwnd, TEXT ("Help not yet
implemented!"),
                                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                    return 0 ;

                case IDM_SYS_REMOVE:
                    GetSystemMenu (hwnd, TRUE) ;
                    return 0 ;

            }
            break ;

        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;

    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

三个功能表 ID 在 POORMENU.C 的开始部分定义:

```

#define IDM_ABOUT          1
#define IDM_HELP           2
#define IDM_REMOVE        3

```

在程式视窗建立之後, POORMENU 得到一个系统功能表的代号:

```
hMenu = GetSystemMenu (hwnd, FALSE) ;
```

第一次呼叫 GetSystemMenu 时, 您应该为修改功能表作准备, 将第二个参数设定为 FALSE。

使用四个 AppendMenu 呼叫来实作对功能表的修改:

```

AppendMenu (hMenu,      MF_SEPARATOR, 0,
            NULL) ;
AppendMenu (hMenu,      MF_STRING, IDM_SYS_ABOUT,      TEXT ("About...")) ;
AppendMenu (hMenu,      MF_STRING, IDM_SYS_HELP,       TEXT ("Help...")) ;
AppendMenu (hMenu,      MF_STRING, IDM_SYS_REMOVE,     TEXT      ("Remove

```

```
Additions"));
```

第一个 AppendMenu 呼叫是添加分隔条。选择「Remove Additions」功能表项将使 POORMENU 删除这些附加的功能表项，这只要把第二个参数设定为 TRUE，再次呼叫 GetSystemMenu 即可：

```
GetSystemMenu (hwnd, TRUE) ;
```

标准系统功能表有下列选项：Restore、Move、Size、Minimize、Maximize 和 Close。它们产生 wParam 分别等於 SC_RESTORE、SC_MOVE、SC_SIZE、SC_MINIMUM、SC_MAXIMUM 和 SC_CLOSE 的 WM_SYSCOMMAND 讯息。尽管 Windows 程式一般不这样做，但是您可以自己处理这些讯息，而不把它们留给 DefWindowProc。您也可以使用下面所述的方法来禁止或者除掉系统功能表的标准选项。Windows 文件中还介绍了一些系统功能表的标准附加项目，这些附加项目使用识别字 SC_NEXTWINDOW、SC_PREVWINDOW、SC_VSCROLL、SC_HSCROLL 和 SC_ARRANGE。您也许会发现，在一些应用程式中将这些命令加入系统功能表是合适的。

改变功能表

我们已经看到了如何使用 AppendMenu 函式为程式定义功能表以及将功能表项加入到系统功能表中。在 Windows 3.0 之前，您不得被迫使用 ChangeMenu 函式来完成这种工作。ChangeMenu 函式有很多功能，至少在当时，整个 Windows 中它是最复杂的函式之一。现在，许多函式都比 ChangeMenu 函式还要复杂，并且 ChangeMenu 的功能被分解为五个新的函式：

- **AppendMenu** 在功能表尾部添加一个新的功能表项目
- **DeleteMenu** 删除功能表中一个现有的功能表项并清除该项目
- **InsertMenu** 在功能表中插入一个新项目
- **ModifyMenu** 修改一个现有的功能表项目
- **RemoveMenu** 从功能表中移走某一项目

如果功能表项是一个突现式功能表，那么 DeleteMenu 和 RemoveMenu 之间的区别就很重要。DeleteMenu 清除突现式功能表，但 RemoveMenu 不清除它。

其他功能表命令

下面是在使用功能表时一些有用的函式。

当您改变顶层功能表项时，直到 Windows 重画功能表列时才显示所做的改变。您可以通过下列呼叫来强迫执行功能表更新：

```
DrawMenuBar (hwnd) ;
```

注意，DrawMenuBar 的参数是视窗代号而不是功能表代号。

您可以使用下列命令来获得突现式功能表的代号：

```
hMenuPopup = GetSubMenu (hMenu, iPosition) ;
```

其中 iPosition 是 hMenu 指示的顶层功能表中突现式功能表项的索引（开始为 0）。然後您可以在其他函式中使用突现式功能表代号（例如在 AppendMenu 函式中）。

您可以使用下列命令获得顶层功能表或者突现式功能表中目前的项数：

```
iCount = GetMenuItemCount (hMenu) ;
```

您可以取得突现式功能表项的功能表 ID：

```
id = GetMenuItemID (hMenuPopup, iPosition) ;
```

其中 iPosition 是功能表项在突现式功能表中的位置（以 0 开始）。

在 MENUDEMO 中您已经看到如何选中、或者取消选中突现式功能表中的某一项：

```
CheckMenuItem (hMenu, id, iCheck) ;
```

在 MENUDEMO 中，hMenu 是顶层功能表的代号，id 是功能表 ID，而 iCheck 的值是 MF_CHECKED 或 MF_UNCHECKED。如果 hMenu 是突现式功能表代号，那么参数 id 是位置索引而不是功能表 ID。如果使用索引会更方便的话，那么您可以在第三个参数中包含 MF_BYPOSITION，例如：

```
CheckMenuItem (hMenu, iPosition, MF_CHECKED | MF_BYPOSITION) ;
```

除了第三个参数是 MF_ENABLED、MF_DISABLED 或 MF_GRAYED 外，EnableMenuItem 函式与 CheckMenuItem 函式所完成的工作类似。如果您在具有突现式功能表的顶层功能表项上使用 EnableMenuItem，那么必须在第三个参数中使用 MF_BYPOSITION 识别字，因为功能表项没有功能表 ID。我们将在本章後面所示的 POPPAD2 程式中看到 EnableMenuItem 的一个例子。HiliteMenuItem 也类似於 CheckMenuItem 和 EnableMenuItem，但是它使用的是 MF_HILITE 和 MF_UNHILITE。当您在功能表项之间移动时，Windows 使用反白显示方式加亮显示功能表项。您通常不需要使用 HiliteMenuItem。

您还需要对您的功能表做些什么呢？还记得我们在功能表中使用了哪些字符串吗？您可以透过下面的呼叫来回顾一下：

```
iCharCount = GetMenuString (hMenu, id, pString, iMaxCount, iFlag) ;
```

iFlag 可以是 MF_BYCOMMAND(其中 id 是功能表 ID)，也可以是 MF_BYPOSITION（其中的 id 是位置索引）。函式将字符串的 iMaxCount 个位元组复制到 pString 中，并传回复制的位元组数。

或许您也想知道功能表项目目前的属性是什么：

```
iFlags = GetMenuState (hMenu, id, iFlag) ;
```

同样地，iFlag 可以是 MF_BYCOMMAND 或 MF_BYPOSITION。传回值 iFlags 是目前所有属性的组合，您可以通过对 MF_DISABLED、MF_GRAYED、MF_CHECKED、

MF_MENUBREAK、MF_MENUBARBREAK 和 MF_SEPARATOR 识别字的检测来决定目前的属性。

也许现在您对功能表有了一些了解。这时您可能想知道，如果您不再需要功能表时又应该如何处理。您可以使用下面的命令来清除功能表：

```
DestroyMenu (hMenu) ;
```

从而使功能表代号无效。

建立功能表的非正统方法

现在让我们稍微偏离我们所讨论的主题。如果在您的程式中没有下拉式功能表，而是建立了多个没有突现式功能表的顶层功能表，并呼叫 SetMenu 在顶层功能表之间切换，那会是什么样的情形呢？就像 Lotus 1-2-3 中老式的文字模式功能表那样。程式 10-8 中的 NOPOPUPS 程式展示了处理这种情况。在这个程式中，「File」和「Edit」项与 MENUDEMO 程式中的类似，但是却以另一种顶层功能表显示出来。

程式 10-8 NOPOPUPS

```
NOPOPUPS.C
/*-----
    NOPOPUPS.C --          Demonstrates No-Popup Nested Menu
                           (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("NoPopUps") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style     = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
```

```
wndclass.lpszClassName = szAppName ;
if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName,
    TEXT ("No-Popup Nested Menu Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static HMENU      hMenuMain, hMenuEdit, hMenuFile ;
    HINSTANCE          hInstance ;
    switch (message)
    {
        case  WM_CREATE:
            hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

            hMenuMain = LoadMenu (hInstance, TEXT ("MenuMain")) ;
            hMenuFile = LoadMenu (hInstance, TEXT ("MenuFile")) ;
            hMenuEdit = LoadMenu (hInstance, TEXT ("MenuEdit")) ;

            SetMenu (hwnd, hMenuMain) ;
            return 0 ;

        case  WM_COMMAND:
            switch (LOWORD (wParam))
            {
                case  IDM_MAIN:
                    SetMenu (hwnd, hMenuMain) ;
```



```

        return 0 ;

    case  IDM_FILE:
        SetMenu (hwnd, hMenuFile) ;
        return 0 ;

    case  IDM_EDIT:
        SetMenu (hwnd, hMenuEdit) ;
        return 0 ;

    case  IDM_FILE_NEW:
    case  IDM_FILE_OPEN:
    case  IDM_FILE_SAVE:
    case  IDM_FILE_SAVE_AS:
    case  IDM_EDIT_UNDO:
    case  IDM_EDIT_CUT:
    case  IDM_EDIT_COPY:
    case  IDM_EDIT_PASTE:
    case  IDM_EDIT_CLEAR:
        MessageBeep (0) ;
        return 0 ;

    }
    break ;

case  WM_DESTROY:
    SetMenu (hwnd, hMenuMain) ;
    DestroyMenu (hMenuFile) ;
    DestroyMenu (hMenuEdit) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

NOPOPUPS.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

MENUMAIN MENU DISCARDABLE

BEGIN

 MENUITEM "MAIN:", 0, INACTIVE

 MENUITEM "&File...", IDM_FILE

 MENUITEM "&Edit...", IDM_EDIT

END

```

MENUFILE MENU DISCARDABLE
BEGIN
    MENUITEM "FILE:",          0, INACTIVE
    MENUITEM "&New",
    IDM_FILE_NEW
    MENUITEM "&Open...",          IDM_FILE_OPEN
    MENUITEM "&Save",
    IDM_FILE_SAVE
    MENUITEM "Save &As",
    IDM_FILE_SAVE_AS
    MENUITEM "(&Main)",
    IDM_MAIN
END
MENUEEDIT MENU DISCARDABLE
BEGIN
    MENUITEM "EDIT:",          0, INACTIVE
    MENUITEM "&Undo",          IDM_EDIT_UNDO
    MENUITEM "Cu&t",          IDM_EDIT_CUT
    MENUITEM "&Copy",          IDM_EDIT_COPY
    MENUITEM "&Paste",          IDM_EDIT_PASTE
    MENUITEM "De&lete",          IDM_EDIT_CLEAR
    MENUITEM "(&Main)",          IDM_MAIN
END

```

[RESOURCE.H \(摘录\)](#)

```

// Microsoft Developer Studio generated include file.
// Used by NoPopups.rc

#define IDM_FILE          40001
#define IDM_EDIT          40002
#define IDM_FILE_NEW      40003
#define IDM_FILE_OPEN     40004
#define IDM_FILE_SAVE     40005
#define IDM_FILE_SAVE_AS  40006
#define IDM_MAIN          40007
#define IDM_EDIT_UNDO     40008
#define IDM_EDIT_CUT      40009
#define IDM_EDIT_COPY     40010
#define IDM_EDIT_PASTE    40011
#define IDM_EDIT_CLEAR    40012

```

在 Microsoft Developer Studio 中，您建立了三个功能表，而不是一个。从「Insert」中选择「Resource」三次，每个功能表有一个不同的名称。当视窗讯息处理程式处理 WM_CREATE 讯息时，Windows 将每个功能表资源载入记忆体：

```

hMenuMain = LoadMenu (hInstance, TEXT ("MenuMain")) ;
hMenuFile = LoadMenu (hInstance, TEXT ("MenuFile")) ;
hMenuEdit = LoadMenu (hInstance, TEXT ("MenuEdit")) ;

```

开始时，程式只显示主功能表：

```
SetMenu (hwnd, hMenuMain) ;
```

主功能表使用字串「MAIN:」、「File...」和「Edit...」列出这三个选项。然而,「MAIN:」是禁用的,因此它不能使 WM_COMMAND 讯息被发送到视窗讯息处理程式。「File」和「Edit」功能表项以「FILE:」和「EDIT:」开始,表示它们是子功能表。每个功能表的最後一项都是字串「(Main)」,表示传回到主功能表。在这三个功能表之间进行切换是很简单的:

```
case WM_COMMAND :
    switch (wParam)
    {
        case IDM_MAIN :
            SetMenu (hwnd, hMenuMain) ;
            return 0 ;

        case IDM_FILE :
            SetMenu (hwnd, hMenuFile) ;
            return 0 ;

        case IDM_EDIT :
            SetMenu (hwnd, hMenuEdit) ;
            return 0 ;

        其他行程式
    }
    break ;
```

在 WM_DESTROY 讯息处理期间,NOPOPUPS 将程式的功能表设定为主功能表,并呼叫 DestroyMenu 来清除「File」和「Edit」功能表。当视窗被清除时,主功能表将被自动清除。

键盘加速键

加速键是产生 WM_COMMAND 讯息(有些情况下是 WM_SYSCOMMAND)的键组合。许多时候,程式使用加速键来重复常用功能表项的动作(然而,加速键还可以用於执行非功能表功能)。例如,许多 Windows 程式都有一个包含「Delete」或「Clear」选项的「Edit」功能表,这些程式习惯上都将 Del 键指定为该选项的加速键。使用者可以通过「Alt 键」从功能表中选择「Delete」选项,或者只需按下加速键 Del。当视窗讯息处理程式收到一个 WM_COMMAND 讯息时,它不必确定使用的是功能表还是加速键。

为什么要使用加速键

您也许会问:为什么我应该使用加速键?为什么不能直接拦截 WM_KEYDOWN

或 WM_CHAR 讯息而自己实作同样的功能表功能呢？好处又在哪里呢？对于一个单视窗應用程式，您当然可以拦截键盘讯息，但是使用加速键可以得到一些好处：您不需要把功能表和加速键的处理方式重写一遍。

对于有多个视窗和多个视窗讯息处理程式的應用程式来说，加速键是非常重要的。正如我们所看到的，Windows 将键盘讯息发送给目前活动视窗的视窗讯息处理程式。然而对于加速键，Windows 把 WM_COMMAND 讯息发送给视窗讯息处理程式，该视窗讯息处理程式的代号在 Windows 函式 TranslateAccelerator 中给出。通常这是主视窗，也是拥有功能表的视窗，这意味著无须每个视窗讯息处理程式都把加速键的操作处理程式重写一遍。

如果您在主视窗的显示区域中，使用了非系统模态对话方块（在下一章中会讨论）或者子视窗，那么这种好处就变得非常重要。如果定义一个特定的加速键以便在不同的视窗之间移动，那么，只需要一个视窗讯息处理程式有这个处理程式。子视窗就不会收到加速键引发的 WM_COMMAND 讯息。

安排加速键的几条规则

理论上，您可以使用任何虚拟键或者字元键连同 Shift 键、Ctrl 键或 Alt 键来定义加速键。然而，您应该尽力使應用程式之间协调一致，并且尽量避免干扰 Windows 的键盘使用。在加速键中，应该避免使用 Tab、Enter、Esc 和 Spacebar 键，因为这些键常常用于完成系统功能。

加速键最经常的用途是操作程式的「Edit」功能表中的各项。为这些功能表项推荐的加速键在 Windows 3.0 和 Windows 3.1 之间已有不同，因此通常都要支援如下所列的新旧两套加速键：

表 10-2

功能	旧加速键	新加速键
Undo	Alt+Backspace	Ctrl+Z
Cut	Shift+Del	Ctrl+X
Copy	Ctrl+Ins	Ctrl+C
Paste	Shift+Ins	Ctrl+V
Delete 或 Clear	Del	Del

另一种常用的虚拟键是启动辅助资讯的功能键 F1。应该避免使用 F4、F5 和 F6 键，因为这些键常用在多重文件介面（MDI）程式中来完成特殊的功能（将在第十九章中讨论）。

加速键表

您可以在 Developer Studio 中定义加速键表。为了让程式中载入加速键表更为容易，给它和程式名相同的名称（与功能表和图示名也相同）。

每个加速键都有在 **Accel Properties** 对话方块中定义的 ID 和按键组合。如果您已经定义了功能表，则功能表 ID 会出现在下拉式清单方块中，因此不需要键入它们。

加速键可以是虚拟键或 ASCII 字元与 Shift、Ctrl 或 Alt 键的组合。可以通过在字母前键入『^』来指定带有 Ctrl 键的 ASCII 字元。也可以从下拉式清单方块中选取虚拟键。

当您为功能表项定义加速键时，应该将键的组合包含到功能表项的文字中。跳位字元（\t）将文字与加速键分割开，将加速键列在第二列。为了在功能表中为加速键做上标记，可以在文字「Ctrl」、「Shift」或「Alt」之後跟上一个「+」号和一个键名（例如，「Shift+F6」或「Ctrl+F6」）。

加速键表的载入

在您的程式中，您使用 LoadAccelerators 函式把加速键表载入记忆体，并获得该表的代号。LoadAccelerators 叙述非常类似於 LoadIcon、LoadCursor 和 LoadMenu 叙述。

首先，把加速键表的代号定义为型态 HANDLE：

```
HANDLE hAccel ;
```

然後载入加速键表：

```
hAccel = LoadAccelerators (hInstance, TEXT ("MyAccelerators")) ;
```

正如图示、游标和功能表一样，您可以使用一个数值代替加速键表的名称，然後在 LoadAccelerators 叙述中和 MAKEINTRESOURCE 巨集一起使用该数值，或者把它放在双引号内，前面冠以字元「#」。

键盘代码转换

现在我们将讨论底下这三行程式码，在本书中，截至目前为止建立的所有 Windows 程式中都使用过它们。这些程式码是标准的讯息回圈：

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
```

下面把上头那段程式码加以修改，以便使用加速键：

```
while (GetMessage (&msg, NULL, 0, 0))
```

```
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
```

TranslateAccelerator 函式确认存放在 msg 讯息结构中的讯息是否为键盘讯息。如果是，该函式将找寻代号为 hAccel 的加速键表。如果找到了一个符合的，则呼叫代号为 hwnd 的视窗讯息处理程式。如果加速键 ID 与系统功能表的功能表项一致，则讯息就是 WM_SYSCOMMAND；否则，讯息为 WM_COMMAND。

当 TranslateAccelerator 传回时，如果讯息已经被转换（并且已经被发送给视窗讯息处理程式），那么传回值为非零；否则，传回值为 0。如果 TranslateAccelerator 传回一个非零值，则不呼叫 TranslateMessage 和 DispatchMessage，而是经过回圈回到 GetMessage 呼叫中。

TranslateMessage 中的参数 hwnd 看起来有点累赘，因为讯息回圈中的其他三个函式都没有要求这个参数。此外，讯息结构本身（结构变数 msg）有一个叫做 hwnd 的成员，它是视窗代号。

该函式有些不同的原因在於：msg 结构的栏位由 GetMessage 呼叫填入。当 GetMessage 的第二个参数为 NULL 时，函式会找寻应用程式所有视窗的讯息。当 GetMessage 传回时，msg 结构的 hwnd 是将要获得讯息之视窗的视窗代号。然而，当 TranslateAccelerator 把键盘讯息转换为 WM_COMMAND 或 WM_SYSCOMMAND 讯息时，它使用函式的第一个参数指定的视窗代号 hwnd 来代替视窗代号 msg.hwnd。Windows 就是这样把所有加速键讯息发送给同一视窗讯息处理程式的，即使另一个应用视窗目前拥有输入焦点。当系统模态对话方块或者讯息方块拥有输入焦点时，TranslateAccelerator 不会转换键盘讯息，因为这些视窗的讯息是不经过程式的讯息回圈的。

在某些情况下，当您程式的另一个视窗（比如一个非系统模态对话方块）拥有输入焦点时，您也许不想转换加速键。您将在下一章中看到如何处理这种情况。

接收加速键讯息

当加速键与系统功能表中的功能表项相对应时，TranslateAccelerator 给视窗讯息处理程式发送一个 WM_SYSCOMMAND 讯息，否则，TranslateAccelerator 给视窗讯息处理程式发送一个 WM_COMMAND 讯息。下表所示为几种可能接收到的 WM_COMMAND 讯息，这些讯息用於加速键、功能表命令以及子视窗控制项：

表 10-3

	加速键	功能表	控制项
LOWORD (wParam)	加速键 ID	功能表 ID	控制项 ID
HIWORD (wParam)	1	0	通知码
lParam	0	0	子视窗代号

如果加速键与一个功能表项对应，那么视窗讯息处理程式还会收到 WM_INITMENU、WM_INITMENUPOPUP 和 WM_MENUSELECT 讯息，就好像选中了功能表选项一样。在处理 WM_INITMENUPOPUP 时，程式往往启用和禁用突现式功能表中的功能表项，因此，在使用加速键时，您仍然能够实作这类功能。如果加速键与一个禁用或者无效化的功能表项相对应，那么，TranslateAccelerator 函式就不会向视窗讯息处理程式发送 WM_COMMAND 或 WM_SYSCOMMAND 讯息。

如果活动视窗已经被最小化，那么 TranslateAccelerator 将为与启用的系统功能表项相对应的加速键向视窗讯息处理程式发送 WM_SYSCOMMAND 讯息，而不是 WM_COMMAND 讯息。TranslateAccelerator 也会为没有任何功能表项与之对应的加速键，来向视窗讯息处理程式发送 WM_COMMAND 讯息。

功能表与加速键应用程式 POPPAD

在第九章，我们建立了一个叫做 POPPAD1 的程式，它使用了子视窗编辑控制项来实作基本的笔记本功能。在这一章中，我们将加入「File」和「Edit」功能表，并称此程式为 POPPAD2。「Edit」功能表的功能表项的功能全部可用；我们将在第十一章中完成「File」功能，在第十三章中完成「Print」功能。POPPAD2 如程式 10-9 所示。

程式 10-9 POPPAD2

```
POPPAD2.C
/*-----
-
      POPPAD2.C --          Popup Editor Version 2 (includes menu)
                              (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

#define ID_EDIT              1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
TCHAR szAppName[] = TEXT ("PopPad2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```

                                PSTR szCmdLine, int iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS         wndclass ;
    wndclass.style           = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon           = LoadIcon (hInstance,
szAppName) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName     = szAppName ;
    wndclass.lpszClassName    = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, szAppName,
        WS_OVERLAPPEDWINDOW,
        GetSystemMetrics (SM_CXSCREEN) / 4,
        GetSystemMetrics (SM_CYSCREEN) / 4,
        GetSystemMetrics (SM_CXSCREEN) / 2,
        GetSystemMetrics (SM_CYSCREEN) / 2,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    hAccel = LoadAccelerators (hInstance, szAppName) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

```



```

}

AskConfirmation (HWND hwnd)
{
    return MessageBox (    hwnd, TEXT ("Really want to close PopPad2?"),
                        szAppName, MB_YESNO | MB_ICONQUESTION) ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static HWND        hwndEdit ;
    int                iSelect, iEnable ;

    switch (message)
    {
    case WM_CREATE:
        hwndEdit = CreateWindow (TEXT ("edit"), NULL,
        WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
        WS_BORDER | ES_LEFT | ES_MULTILINE |
        ES_AUTOHSCROLL | ES_AUTOVSCROLL,
        0, 0, 0, 0, hwnd, (HMENU) ID_EDIT,
        ((LPCREATESTRUCT) lParam)->hInstance, NULL) ;
        return 0 ;

    case WM_SETFOCUS:
        SetFocus (hwndEdit) ;
        return 0 ;

    case WM_SIZE:
        MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam),
TRUE) ;

        return 0 ;

    case WM_INITMENUPOPUP:
        if (lParam == 1)
        {
            EnableMenuItem        ((HMENU)        wParam,
IDM_EDIT_UNDO,

            SendMessage (hwndEdit, EM_CANUNDO, 0, 0) ?
MF_ENABLED : MF_GRAYED) ;

            EnableMenuItem        ((HMENU)        wParam,
IDM_EDIT_PASTE,

            IsClipboardFormatAvailable (CF_TEXT) ?
MF_ENABLED : MF_GRAYED) ;

            iSelect = SendMessage (hwndEdit, EM_GETSEL,

```

```

0, 0) ;

        if (HIWORD (iSelect) == LOWORD (iSelect))
            iEnable = MF_GRAYED ;
        else
            iEnable = MF_ENABLED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CLEAR, iEnable) ;
    return 0 ;
}
break ;
case WM_COMMAND:
    if (lParam)
    {
        if (LOWORD (lParam) == ID_EDIT &&
            (HIWORD (wParam) == EN_ERRSPACE ||
             HIWORD (wParam) == EN_MAXTEXT))
            MessageBox (hwnd, TEXT ("Edit control out of space."),
                szAppName, MB_OK | MB_ICONSTOP) ;
        return 0 ;
    }

    else switch (LOWORD (wParam))
    {
    case IDM_FILE_NEW:
    case IDM_FILE_OPEN:
    case IDM_FILE_SAVE:
    case IDM_FILE_SAVE_AS:
    case IDM_FILE_PRINT:
        MessageBeep (0) ;
        return 0 ;

    case IDM_APP_EXIT:
        SendMessage (hwnd, WM_CLOSE, 0, 0) ;
        return 0 ;

    case IDM_EDIT_UNDO:
        SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
        return 0 ;

    case IDM_EDIT_CUT:
        SendMessage (hwndEdit, WM_CUT, 0, 0) ;
        return 0 ;

    case IDM_EDIT_COPY:
        SendMessage (hwndEdit, WM_COPY, 0, 0) ;
        return 0 ;
    }
}

```

```

        case IDM_EDIT_PASTE:
            SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
            return 0 ;

        case IDM_EDIT_CLEAR:
            SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
            return 0 ;

        case IDM_EDIT_SELECT_ALL:
            SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
            return 0 ;

        case IDM_HELP_HELP:
            MessageBox (hwnd, TEXT ("Help not yet
implemented!"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            return 0 ;

        case IDM_APP_ABOUT:
            MessageBox (hwnd, TEXT ("POPPAD2 (c) Charles
Petzold, 1998"),
                szAppName, MB_OK | MB_ICONINFORMATION) ;
            return 0 ;
    }
    break ;

case WM_CLOSE:
    if (IDYES == AskConfirmation (hwnd))
        DestroyWindow (hwnd) ;
    return 0 ;

case WM_QUERYENDSESSION:
    if (IDYES == AskConfirmation (hwnd))
        return 1 ;
    else
        return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

POPPAD2.RC (摘录)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

```

```

////////////////////////////////////
/
// Icon
POPPAD2          ICON    DISCARDABLE    "poppad2.ico"
////////////////////////////////////
/
// Menu
POPPAD2 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New",                      IDM_FILE_NEW
        MENUITEM "&Open...",                  IDM_FILE_OPEN
        MENUITEM "&Save",                      IDM_FILE_SAVE
        MENUITEM "Save &As...",              IDM_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print",
        IDM_FILE_PRINT
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                      IDM_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",              IDM_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                IDM_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",              IDM_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",             IDM_EDIT_PASTE
        MENUITEM "De&lete\tDel",
        IDM_EDIT_CLEAR
        MENUITEM SEPARATOR
        MENUITEM "&Select All",
        IDM_EDIT_SELECT_ALL
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help...",
        IDM_HELP_HELP
        MENUITEM "&About PopPad2...",          IDM_APP_ABOUT
    END
END

////////////////////////////////////
/
// Accelerator
POPPAD2 ACCELERATORS DISCARDABLE
BEGIN

```

```

VK_BACK,          IDM_EDIT_UNDO,          VIRTKEY, ALT, NOINVERT
VK_DELETE,        IDM_EDIT_CLEAR,         VIRTKEY, NOINVERT
VK_DELETE,        IDM_EDIT_CUT,           VIRTKEY, SHIFT, NOINVERT
VK_F1,            IDM_HELP_HELP,          VIRTKEY, NOINVERT
VK_INSERT,        IDM_EDIT_COPY,          VIRTKEY, CONTROL, NOINVERT
VK_INSERT,        IDM_EDIT_PASTE,         VIRTKEY, SHIFT, NOINVERT
"^C",            IDM_EDIT_COPY,           ASCII, NOINVERT
"^V",            IDM_EDIT_PASTE,           ASCII, NOINVERT
"^X",            IDM_EDIT_CUT,             ASCII, NOINVERT
"^Z",            IDM_EDIT_UNDO,            ASCII, NOINVERT
END

```

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by POPPAD2.RC

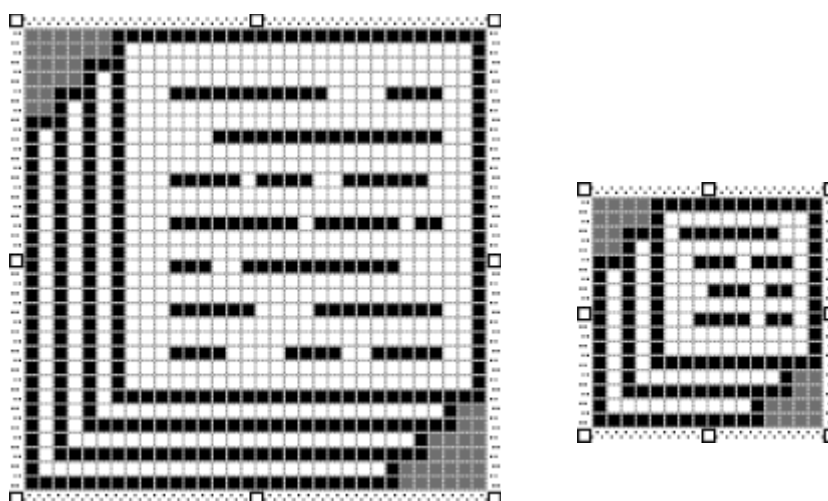
```

```

#define IDM_FILE_NEW          40001
#define IDM_FILE_OPEN         40002
#define IDM_FILE_SAVE         40003
#define IDM_FILE_SAVE_AS      40004
#define IDM_FILE_PRINT        40005
#define IDM_APP_EXIT          40006
#define IDM_EDIT_UNDO         40007
#define IDM_EDIT_CUT          40008
#define IDM_EDIT_COPY         40009
#define IDM_EDIT_PASTE        40010
#define IDM_EDIT_CLEAR        40011
#define IDM_EDIT_SELECT_ALL    40012
#define IDM_HELP_HELP         40013
#define IDM_APP_ABOUT         40014

```

POPPAD2.ICO



POPPAD2.RC 资源描述档包含功能表和加速键。您将注意到，所有加速键都表示在跳位字元 (\t) 後的「Edit」突现式功能表的字串中。

启用功能表项

视窗讯息处理程式的工作包括启用和无效化「Edit」功能表中的选项，这项工作在处理 WM_INITMENUPOPUP 时完成。首先，程式检查是否要显示「Edit」突现式功能表。因为功能表里「Edit」的位置索引（「File」从 0 开始）是 1，因此如果即将显示「Edit」突现式功能表，那么 lParam 应该等於 1。

为了确定是否启用「Undo」选项，POPPAD2 给编辑控制项发送一条 EM_CANUNDO 讯息。如果编辑控制项能够执行「Undo」动作，那么 SendMessage 呼叫传回非零值。在这种情况下，选项被启用；否则，选项无效化：

```
EnableMenuItem (wParam, IDM_UNDO,
    SendMessage (hwndEdit, EM_CANUNDO, 0, 0) ?
        MF_ENABLED : MF_GRAYED) ;
```

只有当剪贴簿中包含文字时，「Paste」选项才能够被启用。我们可以使用 CF_TEXT 识别字通过 IsClipboardFormatAvailable 呼叫来确定这一点：

```
EnableMenuItem (wParam, IDM_PASTE,
    IsClipboardFormatAvailable (CF_TEXT) ? MF_ENABLED : MF_GRAYED) ;
```

只有选择了编辑控制项中的文字，「Cut」、「Copy」和「Delete」选项才能够被启用。给编辑控制项发送一条 EM_GETSEL 讯息，并传回包含此资讯的整数：

```
iSelect = SendMessage (hwndEdit, EM_GETSEL, 0, 0) ;
```

iSelect 的低位元字是第一个被选中字元的位置，iSelect 的高字组是下一个被选中字元的位置。如果这两个字相等，则表示没有选中文字：

```
if (HIWORD (iSelect) == LOWORD (iSelect))
    iEnable = MF_GRAYED ;
else
    iEnable = MF_ENABLED ;
```

然後可以将 iEnable 的值用於「Cut」、「Copy」和「Delete」选项：

```
EnableMenuItem (wParam, IDM_CUT, iEnable) ;
EnableMenuItem (wParam, IDM_COPY, iEnable) ;
EnableMenuItem (wParam, IDM_DEL, iEnable) ;
```

处理功能表项

当然，如果 POPPAD2 程式不使用子视窗编辑控制项，那么我们将面临一些问题，这涉及如何完成「Edit」功能表中的「Undo」、「Cut」、「Copy」、「Paste」、「Clear」和「Select All」选项。正是编辑控制项使得这种处理变得容易，因为对於每一个选项我们只需向编辑控制项发送一个讯息即可：

```
case IDM_UNDO :
    SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
    return 0 ;
```

```

case IDM_CUT :
    SendMessage (hwndEdit, WM_CUT, 0, 0) ;
    return 0 ;

case IDM_COPY :
    SendMessage (hwndEdit, WM_COPY, 0, 0) ;
    return 0 ;

case IDM_PASTE :
    SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
    return 0 ;

case IDM_DEL :
    SendMessage (hwndEdit, WM_DEL, 0, 0) ;
    return 0 ;

case IDM_SELALL :
    SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
    return 0 ;

```

注意，我们可以更进一步简化这些处理——只要使 IDM_UNDO、IDM_CUT 等的值等於相对应的视窗讯息 WM_UNDO、WM_CUT 的值。

「File」突现式功能表上的「About」选项启动一个简单的讯息方块：

```

case IDM_ABOUT :
    MessageBox (hwnd, TEXT ("POPPAD2 (c) Charles Petzold, 1998"),
                szAppName, MB_OK |
MB_ICONINFORMATION) ;
    return 0 ;

```

在下一章中，我们将把它变成一个对话方块。当您从功能表中选择「Help」选项或者按下 F1 加速键时，同样可以启动一个讯息方块。

「Exit」选项向视窗讯息处理程式发送一个 WM_CLOSE 讯息：

```

case IDM_EXIT :
    SendMessage (hwnd, WM_CLOSE, 0, 0) ;
    return 0 ;

```

这正是 DefWindowProc 收到一个 wParam 等於 SC_CLOSE 的 WM_SYSCOMMAND 讯息时所完成的工作。

在前面的那些程式中，我们没有在视窗讯息处理程式中处理 WM_CLOSE 讯息，而只是简单地把它送给 DefWindowProc。DefWindowProc 对 WM_CLOSE 的处理非常简单：呼叫 DestroyWindow 函式。可以不把 WM_CLOSE 讯息送给 DefWindowProc，而让 POPPAD2 来处理它。这个事实到目前为止并不重要，但是在第十一章中当 POPPAD 可以真正编辑文字时，它就变得非常重要了。

```

case WM_CLOSE :
    if (IDYES == AskConfirmation (hwnd))
        DestroyWindow (hwnd) ;

```

```
    return 0 ;  
AskConfirmation 是 POPPAD2 中的一个函式，它显示一个请求确认关闭程式的讯息方块：  
AskConfirmation (HWND hwnd)  
{  
    return MessageBox (hwnd, TEXT ("Really want to close Poppad2?"),  
                        szAppName, MB_YESNO | MB_ICONQUESTION) ;  
}
```

如果选择了 Yes 按钮的话，讯息方块（以及 AskConfirmation 函式）将传回 IDYES。只有这样，程式才会呼叫 DestroyWindow，否则，程式不会结束。

如果要在程式结束之前确认使用者真的要结束程式，那么您还必须处理 WM_QUERYENDSESSION 讯息。当使用者要关闭 Windows 时，Windows 开始向每个视窗讯息处理程式发送一个 WM_QUERYENDSESSION 讯息。如果有任何一个视窗讯息处理程式处理这个讯息後传回 0，那么 Windows 将不会结束。我们如下处理了 WM_QUERYENDSESSION：

```
case WM_QUERYENDSESSION :  
    if (IDYES == AskConfirmation (hwnd))  
        return 1 ;  
    else  
        return 0 ;
```

如果要在程式结束之前要求使用者的确认，必须处理 WM_CLOSE 和 WM_QUERYENDSESSION 这两个讯息，这就是为什么我们使 POPPAD2 中的「Exit」功能表选项只向视窗讯息处理程式发送一个 WM_CLOSE 讯息的原因。这样做，我们避免了在别处进行请求确认的动作。

如果要处理 WM_QUERYENDSESSION 讯息，那么您也许还会对 WM_ENDSESSION 讯息感兴趣。Windows 把这个讯息发送给先前收到 WM_QUERYENDSESSION 讯息的每个视窗讯息处理程式。如果由於另一个程式从 WM_QUERYENDSESSION 传回了 0 而不能结束 Windows 的执行，那么 WM_ENDSESSION 的 wParam 参数为 0。WM_ENDSESSION 讯息实际上回答了这个问题：我告诉过 Windows 可以把我结束掉，但是我真的被结束掉了吗？

尽管在 POPPAD2 的「File」功能表中我加上了常见的「New」、「Open」、「Save」和「Save As」选项，但是它们现在并不起作用。要处理这些命令，我们需要使用对话方块。现在是讨论对话方块的时机，也是您准备学习它们的时候了。

第十一章 对话方块

如果有很多输入超出了功能表可以处理的程度，那么我们可以使用对话方块来取得输入资讯。程式写作者可以通过在某选项後面加上省略号 (i) 来表示该功能表项将启动一个对话方块。

对话方块的一般形式是包含多种子视窗控制项的弹出式视窗，这些控制项的大小和位置在程式资源描述档的「对话方块模板」中指定。虽然程式写作者能够「手工」定义对话方块模板，但是现在通常是在 Visual C++ Developer Studio 中以交谈式操作的方式设计的，然後由 Developer Studio 建立对话方块模板。

当程式呼叫依据模板建立的对话方块时，Microsoft Windows 98 负责建立弹出式对话方块视窗和子视窗控制项，并提供处理对话方块讯息（包括所有键盘和滑鼠输入）的视窗讯息处理程式。有时候称呼完成这些功能的 Windows 内部程式码为「对话方块管理器」。

Windows 的内部对话方块视窗讯息处理程式所处理的许多讯息也传递给您自己程式中的函式，这个函式即是所谓的「对话方块程序」或者「对话程序」。对话程序与普通的视窗讯息处理程式类似，但是也存在著一些重要区别。一般来说，除了在建立对话方块时初始化子视窗控制项，处理来自子视窗控制项的讯息以及结束对话方块之外，程式写作者不需要再给对话方块程序增加其他功能。对话程序通常不处理 WM_PAINT 讯息，也不直接处理键盘和滑鼠输入。

对话方块这个主题的含义太广了，因为它还包含子视窗控制项的使用。不过，我们已经在第九章研究了子视窗控制项。当您在对话方块中使用子视窗控制项时，第九章所提到的许多工作都可以由 Windows 的对话方块管理器来完成。尤其是，在程式 COLORS1 中遇到在卷动列之间切换输入焦点的问题也不会在对话方块中出现。Windows 会处理对话方块中的控制项之间切换输入焦点所必需完成的全部工作。

不过，在程式中添加对话方块要比添加图示或者功能表更麻烦一些。我们将从一个简单的对话方块开始，让您对各部分之间的相互联系有所了解。

模态对话方块

对话方块分为两类：「模态的」和「非模态的」，其中模态对话方块最为普遍。当您的程式显示一个模态对话方块时，使用者不能在对话方块与同一个程式中的另一个视窗之间进行切换，使用者必须主动结束该对话方块，这藉由通过按一下「OK」或者「Cancel」键来完成。不过，在显示模态对话方块时，

使用者通常可以从目前的程式切换到另一个程式。而有些对话方块（称为「系统模态」）甚至连这样的切换程式操作也不允许。在 Windows 中，显示了系统模态对话方块之後，要完成其他任何工作，都必须先结束该对话方块。

建立「About」对话方块

Windows 程式即使不需要接收使用者输入，也通常具有由功能表上的「About」选项启动的对话方块，该对话方块用来显示程式的名字、图示、版权旗标和标记为「OK」的按键，也许还会有其他资讯（例如技术支援的电话号码）。我们将要看到的第一个程式除了显示一个「About」对话方块外，别无它用。这个 ABOUT1 程式如程式 11-1 所示：

程式 11-1 ABOUT1

```
ABOUT1.C
/*-----
    ABOUT1.C -- About Box Demo Program No. 1
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT      CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
BOOL         CALLBACK AboutDlgProc     (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("About1") ;
    MSG              msg ;
    HWND             hwnd ;
    WNDCLASS          wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (hInstance, szAppName) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
```

```
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow ( szAppName, TEXT ("About Box Demo Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HINSTANCE hInstance ;
    switch (message)
    {
    case WM_CREATE :
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
        return 0 ;

    case WM_COMMAND :
        switch (LOWORD (wParam))
        {
            case IDM_APP_ABOUT :
                DialogBox (hInstance, TEXT ("AboutBox"), hwnd,
AboutDlgProc) ;
                break ;
        }
        return 0 ;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

```

}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG :
            return TRUE ;

        case WM_COMMAND :
            switch (LOWORD (wParam))
            {
                case IDOK :
                case IDCANCEL :
                    EndDialog (hDlg, 0) ;
                    return TRUE ;

            }

            break ;
    }

    return FALSE ;
}

ABOUT1.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON        "OK", IDOK, 66, 80, 50, 14
    ICON
    "ABOUT1", IDC_STATIC, 7, 7, 21, 20
    CTEXT
    "About1", IDC_STATIC, 40, 12, 100, 8
    CTEXT                "About                Box                Demo
Program", IDC_STATIC, 7, 40, 166, 8
    CTEXT                "(c) Charles Petzold,
1998", IDC_STATIC, 7, 52, 166, 8
END

////////////////////////////////////
/
// Menu
ABOUT1        MENU DISCARDABLE

```

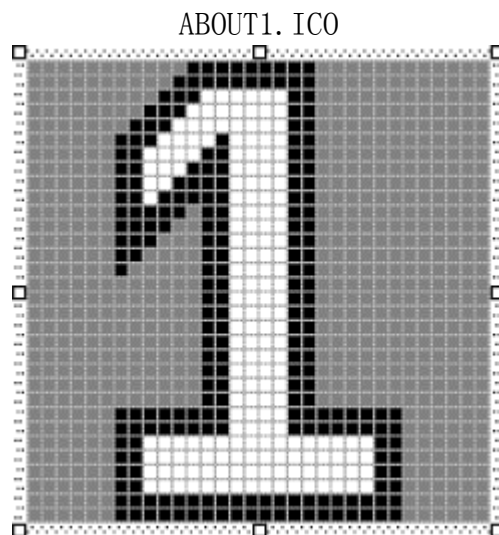
```

BEGIN
    POPUP "&Help"
    BEGIN
        MENUITEM "&About About1...",
        IDM_APP_ABOUT
    END
END

/////////////////////////////////////////////////////////////////
/
// Icon
ABOUT1        ICON        DISCARDABLE        "About1.ico"
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by About1.rc

#define IDM_APP_ABOUT        40001
#define IDC_STATIC        -1

```



藉由後面章节中介绍的方法，您还可以在程式中建立图示和功能表。图示和功能表的 ID 名均为「About1」。功能表有一个选项，它产生一条 ID 名为 IDM_APP_ABOUT 的 WM_COMMAND 讯息。这使得程式显示的图 11-1 所示的对话方块。

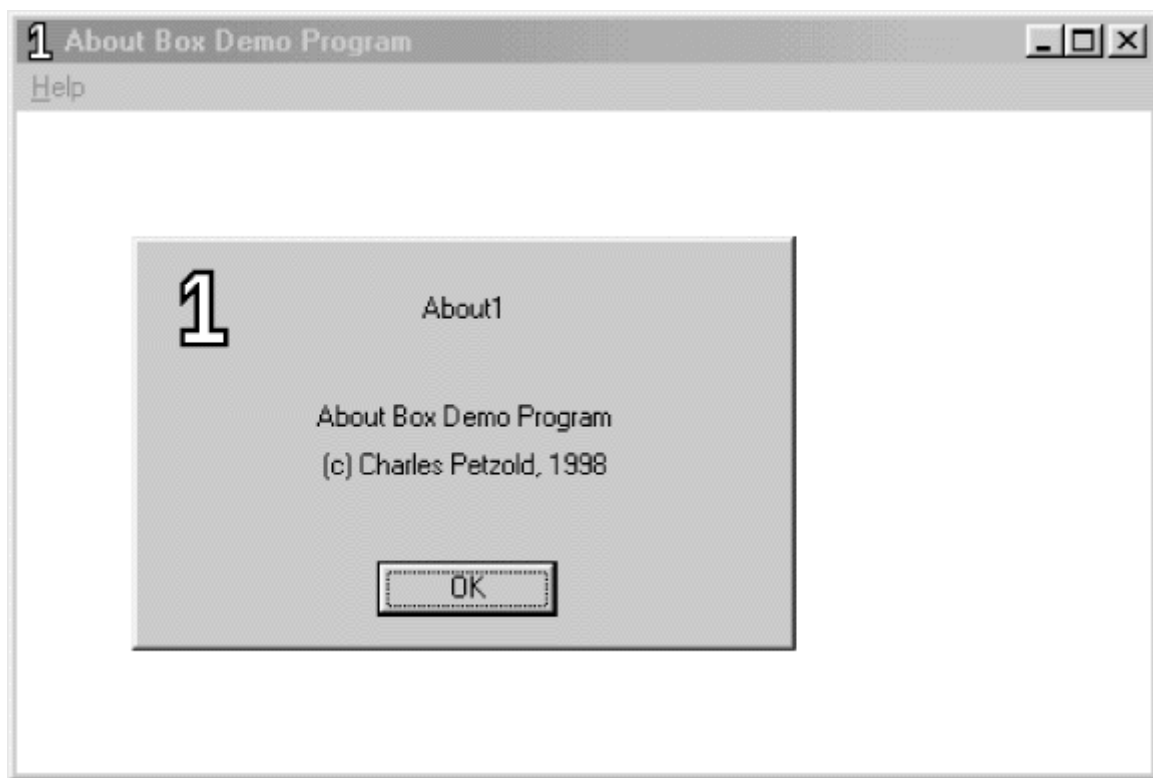


图 11-1 程式 ABOUT1 的对话方块

对话方块及其模板

要把一个对话方块添加到 Visual C++ Developer Studio 会有的应用程序上，可以先从 **Insert** 功能表中选择 **Resource**，然後选择 **Dialog Box**。现在一个对话方块出现在您的眼前，该对话方块带有标题列、标题 (Dialog) 以及 **OK** 和 **Cancel** 按钮。**Controls** 工具列允许您在对话方块中插入不同的控制项。

Developer Studio 将对话方块的 ID 设为标准的 `IDD_DIALOG1`。您可以在名称上 (或者在对话方块本身) 单击右键，然後从功能表中选择 **Properties**。在本程式中，将 ID 改为「AboutBox」（带有引号）。为了与我建立的对话方块保持一致，请将 **X Pos** 和 **Y Pos** 栏位改为 32。这表示对话方块相对於程式视窗显示区域左上角的显示位置待会会有有关於对话方块座标的详细讨论）。

现在，继续在 **Properties** 对话方块中选择 **Styles** 页面标签。因为此对话方块没有标题列，所以不要选取 **Title Bar** 核取方块。然後请单击 **Properties** 对话方块的 **关闭** 按钮。

现在可以设计对话方块了。因为不需要 **Cancel** 按钮，所以先单击该按钮，然後按下键盘上的 **Delete** 键。接著单击 **OK** 按钮，将其移动到对话方块的底部。在 Developer Studio 视窗下面的工具列上有一个小点阵图，它可使控制项在视窗内水平居中对齐，请按下此钮。

如果您要让程式的图示出现在对话方块中，可以这样做：先在浮动的 **Controls** 工具列中按下「**Pictures**」按钮。将滑鼠移动到对话方块的表面，按下左键，然後拉出一个矩形。这就是图示将出现的位置。然後在次矩形上按下滑鼠右键，从功能表中选择 **Properties**。保持 **ID** 为 **IDC_STATIC**。此识别字在 RESOURCE.H 中定义为 -1，用於程式中不使用的所有 ID。将 **Type** 改为 **Icon**。您可以在 **Image** 栏位输入程式图示的名称，或者，如果您已经建立了一个图示，那么您也可以从下拉式清单方块中选择一个名称 (About1)。

對於对话方块中的三个静态字串，可以从 **Controls** 工具列中选择 **Static Text**，然後确定文字在对话方块中的位置。右键单击控制项，然後从功能表中选择 **Properties**。在 **Properties** 框的 **Caption** 栏位中输入要显示的文字。选择 **Styles** 页面标签，从 **Align Text** 栏位选择 **Center**。

在添加这些字串的时候，若希望对话方块可以更大一些，请先选中对话方块，然後拖曳边框。您也可以选择并缩放控制项。通常用键盘上的游标移动键完成此操作会更容易些。箭头键本身移动控制项，按下 **Shift** 键後按箭头键，可以改变控制项的大小。所选控制项的座标和大小显示在 **Developer Studio** 视窗的右下角。

如果您建立了一个應用程式，那么以後在查看资源描述档 ABOUT1.RC 时，您将发现 **Developer Studio** 建立的模板。我所设计的对话方块模板如下：

```
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON    "OK",IDOK,66,80,50,14
    ICON              "ABOUT1",IDC_STATIC,7,7,21,20
    CTEXT
    "About1",IDC_STATIC,40,12,100,8
    CTEXT            "About Box Demo Program",IDC_STATIC,7,40,166,8
    CTEXT            "(c) Charles Petzold, 1998",IDC_STATIC,7,52,166,8
END
```

第一行给出了对话方块的名称 (这里为 ABOUTBOX)。如同其他资源，您也可以使用数字作为对话方块的名称。名称後面是关键字 **DIALOG** 和 **DISCARDABLE** 以及四个数字。前两个数字是对话方块左上角的 **x**、**y** 座标，该座标在程式呼叫对话方块时，是相对於父视窗显示区域的。後两个数字是对话方块的宽度和高度。

这些座标和大小的单位都不是图素。它们实际上依据一种特殊的座标系统，该系统只用於对话方块模板。数字依据对话方块使用字体的大小而定 (这里是 8 点的 **MS Sans Serif** 字体)：**x** 座标和宽度的单位是字元平均宽度的 1/4；**y** 座

标和高度的单位是字元高度的 1/8。因此，对这个对话方块来说，对话方块左上角距离主视窗显示区域的左边是 5 个字元，距离顶边是 2-1/2 个字元。对话方块本身宽 40 个字元，高 10 个字元。

这样的座标系使得程式写作者可以使用座标和大小来大致勾勒对话方块的尺寸和外观，而不管视讯显示器的解析度是多少。由於系统字体字元的高度大致为其宽度的两倍，所以，x 轴和 y 轴的量度差不多相等。

模板中的 STYLE 叙述类似於 CreateWindow 呼叫中的 style 栏位。對於模态对话方块，通常使用 WS_POPUP 和 DS_MODALFRAME，我们将在稍後介绍其他的选项。

在 BEGIN 和 END 叙述（或者是左右大括弧，手工设计对话方块模板时，您可能会使用）之间，定义出现在对话方块中的子视窗控制项。这个对话方块使用了三种型态的子视窗控制项，它们分别是 DEFPUSHBUTTON（内定按键）、ICON（图示）和 CTEXT（文字居中）。这些叙述的格式为：

```
control-type "text" id, xPos, yPos, xWidth, yHeight, iStyle
```

其中，後面的 iStyle 项是可选的，它使用 Windows 表头档案中定义的识别字来指定其他视窗样式。

DEFPUSHBUTTON、ICON 和 CTEXT 等识别字只可以在对话方块中使用，它们是某种特定视窗类别和视窗样式的缩写。例如，CTEXT 指示这个子视窗控制项类别是「静态的」，其样式为：

```
WS_CHILD | SS_CENTER | WS_VISIBLE | WS_GROUP
```

虽然前面没有出现过 WS_GROUP 识别字，但是在第九章的 COLORS1 程式中已经出现过 WS_CHILD、SS_CENTER 和 WS_VISIBLE 视窗样式，我们在建立静态子视窗文字控制项时已经用到了它们。

對於图示，文字栏位是程式的图示资源名称，它也在 ABOUT1 资源描述档中定义。對於按键，文字栏位是出现在按键里的文字，这个文字相同於在程式中建立子视窗控制项时呼叫 CreateWindow 所指定的第二个参数。

id 栏位是子视窗在向其父视窗发送讯息（通常为 WM_COMMAND 讯息）时用来标示它自身的值。这些子视窗控制项的父视窗就是对话方块本身，它将这些讯息发送给 Windows 的一个视窗讯息处理程式。不过，这个视窗讯息处理程式也将这些讯息发送给您在程式中给出的对话方块程序。ID 值相同於我们在第九章建立子视窗时，在 CreateWindow 函式中使用的子视窗 ID。由於文字和图示控制项不向父视窗回送讯息，所以这些值被设定为 IDC_STATIC，它在 RESOURCE.H 中定义为-1。按键的 ID 值为 IDOK，它在 WINUSER.H 中定义为 1。

接下来的四个数字设定子视窗的位置（相對於对话方块显示区域的左上角）和大小，它们是以系统字体平均宽度的 1/4 和平均高度的 1/8 为单位来表示的。

对于 ICON 叙述，宽度和高度将被忽略。

对话方块模板中的 DEFPUSHBUTTON 叙述，除了包含 DEFPUSHBUTTON 关键字所隐含的视窗样式，还包含视窗样式 WS_GROUP。稍后讨论该程式的第二个版本 ABOUT2 时，还会详细说明 WS_GROUP（以及相关的 WS_TABSTOP 样式）。

对话方块程序

您程式内的对话方块程序处理传送给对话方块的讯息。尽管看起来很像是视窗讯息处理程式，但是它并不是真实的视窗讯息处理程式。对话方块的视窗讯息处理程式在 Windows 内部定义，这个视窗程序呼叫您编写的对话方块程序，把它所接收到的许多讯息作为参数。下面是 ABOUT1 的对话方块程序：

```

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG :
            return TRUE ;

        case WM_COMMAND :
            switch (LOWORD (wParam))
            {
                case IDOK :
                case IDCANCEL :
                    EndDialog (hDlg, 0) ;
                    return TRUE ;

                }
            break ;
    }
    return FALSE ;
}

```

该函式的参数与常规视窗讯息处理程式的参数相同，与视窗讯息处理程式类似，对话方块程序都必须定义为一个 CALLBACK (callback) 函式。尽管我使用了 hDlg 作为对话方块视窗的代号，但是您也可以按照您自己的意思使用 hwnd。首先，让我们来看一下这个函式与视窗讯息处理程式的区别：

- 视窗讯息处理程式传回一个 LRESULT。对话方块传回一个 BOOL，它在 Windows 表头档案中定义为 int 型态。
- 如果视窗讯息处理程式不处理某个特定的讯息，那么它将呼叫 DefWindowProc。如果对话方块程序处理一个讯息，那么它传回 TRUE (非 0)，如果不处理，则传回 FALSE (0)。
- 对话方块程序不需要处理 WM_PAINT 或 WM_DESTROY 讯息。对话方块程序

不接收 WM_CREATE 讯息，而是在特殊的 WM_INITDIALOG 讯息处理期间，对话方块程序执行初始化操作。

WM_INITDIALOG 讯息是对话方块接收到的第一个讯息，这个讯息只发送给对话方块程序。如果对话方块程序传回 TRUE，那么 Windows 将输入焦点设定给对话方块中第一个具有 WS_TABSTOP 样式（我们将在 ABOUT2 的讨论中加以解释）的子视窗控制项。在这个对话方块中，第一个具有 WS_TABSTOP 样式的子视窗控制项是按键。另外，对话方块程序也可以在处理 WM_INITDIALOG 时使用 SetFocus 来将输入焦点设定为对话方块中的某个子视窗控制项，然後传回 FALSE。

此外，对话方块程序只处理 WM_COMMAND 讯息。这是当按键被滑鼠点中，或者在按钮具有输入焦点的情况下按下空白键时，按键控制项发送给其父视窗的讯息。这个控制项的 ID（我们在对话方块模板中将其设定为 IDOK）在 wParam 的低字组中。对于这个讯息，对话方块程序呼叫 EndDialog，它告诉 Windows 清除对话方块。对于所有其他讯息，对话方块程序传回 FALSE，并告诉 Windows 内部的对话方块视窗讯息处理程式：我们的对话方块程序不处理这些讯息。

模态对话方块的讯息不通过您程式的讯息伫列，所以不必担心对话方块中键盘加速键的影响。

启动对话方块

在 WndProc 中处理 WM_CREATE 讯息时，ABOUT1 取得程式的执行实体代号并将它放在静态变数中：

```
hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
```

ABOUT1 检查 WM_COMMAND 讯息，以确保讯息 wParam 的低位元字等於 IDM_APP_ABOUT。当它获得这样一个讯息时，程式呼叫 DialogBox：

```
DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
```

该函式需要执行实体代号（在处理 WM_CREATE 时储存的）、对话方块名称（在资源描述档中定义的）、对话方块的父视窗（也是程式的主视窗）和对话方块程序的位址。如果您使用一个数字而不是对话方块模板名称，那么可以用 MAKEINTRESOURCE 巨集将它转换为一个字串。

从功能表中选择「About About1」，将显示图 11-2 所示的对话方块。您可以使用滑鼠单击「OK」按钮、按空白键或者按 Enter 键来结束这个对话方块。对任何包含内定按钮的对话方块，在按下 Enter 键或空白键之後，Windows 发送一个 WM_COMMAND 讯息给对话方块，并令 wParam 的低字组等於内定按键的 ID，此时的 ID 为 IDOK。按下 Escape 键也可以关闭对话方块，这时 Windows 将发送一个 WM_COMMAND 讯息，并令 ID 等於 IDCANCEL。

直到对话方块结束之後，用来显示对话方块的 DialogBox 才将控制权传回

给 WndProc。DialogBox 的传回值是对话方块程序内部呼叫的 EndDialog 函式的第二个参数(这个值未在 ABOUT1 中使用,但会在 ABOUT2 中使用)。然後,WndProc 可以将控制权传回给 Windows。

即使在显示对话方块时,WndProc 也可以继续接收讯息。实际上,您可以从对话方块程序内部给 WndProc 发送讯息。ABOUT1 的主视窗是弹出式对话方块视窗的父视窗,所以 AboutDlgProc 中的 SendMessage 呼叫可以使用如下叙述来开始:

```
SendMessage (GetParent (hDlg), . . . ) ;
```

不同的主题

虽然 Visual C++ Developer Studio 中的对话方块编辑器和其他资源编辑器,使我们几乎不用考虑资源描述的写作问题,但是学习一些资源描述的语法还是有用的。尤其对于对话方块模板来说,知道了语法,您就可以进一步了解对话方块的范围和限制。甚至当它不能满足您的需要时,您还可以自己建立一个对话方块模板(就像本章后面的 HEXCALC 程式)。资源编译器和资源描述语法的文件位于/Platform SDK/Windows Programming Guidelines/Platform SDK Tools/Compiling/Using the Resource Compiler。

在 Developer Studio 的「Properties」对话方块中指定了对话方块的视窗样式,它翻译成对话方块模板中的 STYLE 叙述。对于 ABOUT1,我们使用模态对话方块最常用的样式;

```
STYLE WS_POPUP | DS_MODALFRAME
```

然而,您也可以尝试其他样式。有些对话方块有标题列,标题列用于指出对话方块的用途,并允许使用者通过滑鼠在显示幕上移动对话方块。此样式为 WS_CAPTION。如果您使用 WS_CAPTION,那么 DIALOG 叙述中所指定的 x 座标和 y 座标是对话方块显示区域的座标,并相对于父视窗显示区域的左上角。标题列将在 y 座标之上显示。

如果使用了标题列,那么您可以用 CAPTION 叙述将文字放入标题中。在对话方块模板中,CAPTION 叙述在 STYLE 叙述的后面:

```
CAPTION "Dialog Box Caption"
```

另外,在对话方块程序处理 WM_INITDIALOG 讯息处理期间,您还可以呼叫:

```
SetWindowText (hDlg, TEXT ("Dialog Box Caption")) ;
```

如果您使用 WS_CAPTION 样式,也可以添加一个 WS_SYSMENU 样式的系统功能表按钮。此样式允许使用者从系统功能表中选择 **Move** 或 **Close**。

从 **Properties** 对话方块的 **Border** 清单方块中选择 **Resizing** (相同于样式 WS_THICKFRAME),允许使用者缩放对话方块,尽管此操作并不常用。如果

您不介意更特殊一点的话，还可以著为此对话方块样式添加最大化方块。

您甚至可以给对话方块添加一个功能表。这时对话方块模板将包括下面的叙述：

```
MENU menu-name
```

其参数不是功能表的名称，就是资源描述中的功能表号。模态对话方块很少使用功能表。如果使用了功能表，那么您必须确保功能表和对话方块控制项中的所有 ID 都是唯一的；或者不是唯一的，却表达了相同的命令。

FONT 叙述使您可以设定非系统字体，以供对话方块文字使用。这在过去的对话方块中不常用，但现在却非常普遍。事实上，在内定情况下，Developer Studio 为您建立的每一个对话方块都选用 8 点的 MS Sans Serif 字体。一个 Windows 程式能把自己外观打点得非常与众不同，这只需为程式的对话方块及其他文字输出单独准备一种字体即可。

尽管对话方块视窗讯息处理程式通常位於 Windows 内部，但是您也可以使用自己编写的视窗讯息处理程式来处理对话方块讯息。要这样做，您必须在对话方块模板中指定一个视窗类别名：

```
CLASS "class-name"
```

这种用法很少见，但是在本章後面所示的 HEXCALC 程式中我们将用到它。

当您使用对话方块模板的名称来呼叫 DialogBox 时，Windows 通过呼叫普通的 CreateWindow 函式来完成建立弹出式视窗所需要完成的一切操作。Windows 从对话方块模板中取得视窗的座标、大小、视窗样式、标题和功能表，从 DialogBox 的参数中获得执行实体代号和父视窗代号。它所需要的唯一其他资讯是一个视窗类别（假设对话方块模板不指定视窗类别的话）。Windows 为对话方块注册一个专用的视窗类别，这个视窗类别的视窗讯息处理程式可以存取对话方块程序位址（该位址是您在 DialogBox 呼叫中指定的），所以它可以使程式获得该弹出式视窗所接收的讯息。当然，您可以通过自己建立弹出式视窗来建立和维护自己的对话方块。不过，使用 DialogBox 则更简单。

也许您希望受益於 Windows 对话方块管理器，但不希望（或者能够）在资源描述中定义对话方块模板，也可能您希望程式在执行时可以动态地建立对话方块。这时可以完成这种功能的函式是 DialogBoxIndirect，此函式用资料结构来定义模板。

在 ABOUT1.RC 的对话方块模板中，我们使用缩写 CTEXT、ICON 和 DEFPUSHBUTTON 来定义对话方块所需要的三种型态的子视窗控制项。您还可以使用其他型态，每种型态都隐含一个特定的预先定义视窗类别和一种视窗样式。下表显示了与一些控制项型态相同的视窗类别和视窗样式：

表 11-1

控制项型态	视窗类别	视窗样式
PUSHBUTTON	按钮	BS_PUSHBUTTON WS_TABSTOP
DEFPUSHBUTTON	按钮	BS_DEFPUSHBUTTON WS_TABSTOP
CHECKBOX	按钮	BS_CHECKBOX WS_TABSTOP
RADIOBUTTON	按钮	BS_RADIOBUTTON WS_TABSTOP
GROUPBOX	按钮	BS_GROUPBOX WS_TABSTOP
LTEXT	静态文字	SS_LEFT WS_GROUP
CTEXT	静态文字	SS_CENTER WS_GROUP
RTEXT	静态文字	SS_RIGHT WS_GROUP
ICON	静态图示	SS_ICON
EDITTEXT	编辑框	ES_LEFT WS_BORDER WS_TABSTOP
SCROLLBAR	滚动列	SBS_HORZ
LISTBOX	清单方块	LBS_NOTIFY WS_BORDER WS_VSCROLL
COMBOBOX	下拉式清单方块	CBS_SIMPLE WS_TABSTOP

资源编译器是唯一能够识别这些缩写的程式。除了表中所示的视窗样式外，每个控制项还具有下面的样式：

```
WS_CHILD | WS_VISIBLE
```

對於这些控制项型态，除了 EDITTEXT、SCROLLBAR、LISTBOX 和 COMBOBOX 之外，控制项叙述的格式为：

```
control-type "text", id, xPos, yPos, xWidth, yHeight, iStyle
```

對於 EDITTEXT、SCROLLBAR、LISTBOX 和 COMBOBOX，其格式为：

```
control-type id, xPos, yPos, xWidth, yHeight, iStyle
```

其中没有文字栏位。在这两种叙述中，iStyle 参数都是选择性的。

在第九章，我讨论了确定预先定义子视窗的宽度和高度的规则。您可能需要回到第九章去参考这些规则，这时请记住：对话方块模板中指定大小的单位为平均字元宽度的 1/4，及平均字元高度的 1/8。

控制项叙述的 **style** 栏位是可选的。它允许您包含其他视窗样式识别字。例如，如果您想建立在正方形框左边包含文字的核取方块，那么可以使用：

```
CHECKBOX "text", id, xPos, yPos, xWidth, yHeight, BS_LEFTTEXT
```

注意，控制项型态 EDITTEXT 会自动添加一个边框。如果您想建立一个没有边框的子视窗编辑控制项，您可以使用：

```
EDITTEXT id, xPos, yPos, xWidth, yHeight, NOT WS_BORDER
```

资源编译器也承认与下面叙述类似的专用控制项叙述：

```
CONTROL "text", id, "class", iStyle, xPos, yPos, xWidth, yHeight
```

此叙述允许您通过指定视窗类别和完整的视窗样式，来建立任意型态的子视窗控制项。例如，要取代：

```
PUSHBUTTON "OK", IDOK, 10, 20, 32, 14
```

您可以使用：

```
CONTROL "OK", IDOK, "button", WS_CHILD | WS_VISIBLE |  
BS_PUSHBUTTON | WS_TABSTOP, 10, 20, 32, 14
```

当编译资源描述档时，这两条叙述在 .RES 和 .EXE 档案中的编码是相同的。在 Developer Studio 中，您可以使用 **Controls** 工具列中的 **Custom Control** 选项来建立此叙述。在 ABOUT3 程式中，我向您展示了如何用此选项建立一个控制项，且在您的程式中已定义了该控制项的视窗类别。

当您在对话方块模板中使用 CONTROL 叙述时，不必包含 WS_CHILD 和 WS_VISIBLE 样式。在建立子视窗时，Windows 已经包含了这些视窗样式。CONTROL 叙述的格式也说明 Windows 对话方块管理器在建立对话方块时就完成了此项操作。首先，就像我前面所讨论的，它建立一个弹出式视窗，其父视窗代号在 DialogBox 函式中提供。然後，对话方块管理器为对话方块模板中的每个控制项建立一个子视窗。所有这些控制项的父视窗均是这个弹出式对话方块。上面给出的 CONTROL 叙述被转换成一个 CreateWindow 呼叫，形式如下所示：

```
hCtrl =CreateWindow (TEXT ("button"), TEXT ("OK"),  
WS_CHILD | WS_VISIBLE | WS_TABSTOP |  
BS_PUSHBUTTON,  
10 * cxChar / 4, 20 * cyChar / 8,  
32 * cxChar / 4, 14 * cyChar / 8,  
hDlg, IDOK, hInstance, NULL) ;
```

其中，cxChar 和 cyChar 是系统字体字元的宽度和高度，以图素为单位。hDlg 参数是从建立该对话方块视窗的 CreateWindow 呼叫传回的值；hInstance 参数是从 DialogBox 呼叫获得的。

更复杂的对话方块

ABOUT1 中的简单对话方块展示了设计和执行一个对话方块的要点，现在让我们来看一个稍微复杂的例子。程式 11-2 给出的 ABOUT2 程式展示了如何在对话方块程序中管理控制项（这里用单选按钮）以及如何在对话方块的显示区域中绘图。

程式 11-2 ABOUT2

```
ABOUT2.C  
/*-----  
ABOUT2.C -- About Box Demo Program No. 2  
              (c) Charles Petzold, 1998  
-----*/  
  
#include <windows.h>
```

```

#include "resource.h"

LRESULT      CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
BOOL         CALLBACK AboutDlgProc     (HWND, UINT, WPARAM, LPARAM) ;

int   iCurrentColor      = IDC_BLACK,
      iCurrentFigure     = IDC_RECT ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR          szAppName[] = TEXT ("About2") ;
    MSG                  msg ;
    HWND                 hwnd ;
    WNDCLASS              wndclass ;

    wndclass.style        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc  = WndProc ;
    wndclass.cbClsExtra   = 0 ;
    wndclass.cbWndExtra   = 0 ;
    wndclass.hInstance    = hInstance ;
    wndclass.hIcon        = LoadIcon      (hInstance,
szAppName) ;
    wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("About Box Demo Program"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {

```

```

        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void PaintWindow (HWND hwnd, int iColor, int iFigure)
{
    static COLORREF crColor[8] = { RGB ( 0, 0, 0), RGB ( 0, 0, 255),
        RGB ( 0, 255, 0), RGB ( 0, 255, 255),
        RGB (255, 0, 0), RGB (255, 0, 255),
        RGB (255, 255, 0), RGB (255, 255, 255)} ;

    HBRUSH          hBrush ;
    HDC              hdc ;
    RECT             rect ;

    hdc = GetDC (hwnd) ;
    GetClientRect (hwnd, &rect) ;
    hBrush = CreateSolidBrush (crColor[iColor - IDC_BLACK]) ;
    hBrush = (HBRUSH) SelectObject (hdc, hBrush) ;

    if (iFigure == IDC_RECT)
        Rectangle (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
    else
        Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
    DeleteObject (SelectObject (hdc, hBrush)) ;
    ReleaseDC (hwnd, hdc) ;
}

void PaintTheBlock (HWND hCtrl, int iColor, int iFigure)
{
    InvalidateRect (hCtrl, NULL, TRUE) ;
    UpdateWindow (hCtrl) ;
    PaintWindow (hCtrl, iColor, iFigure) ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HINSTANCE      hInstance ;
    PAINTSTRUCT           ps ;

    switch (message)
    {
    case WM_CREATE:
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
        return 0 ;
    }
}

```



```

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_APP_ABOUT:
                if (DialogBox (hInstance, TEXT ("AboutBox"),
hwnd, AboutDlgProc))
                    InvalidateRect (hwnd, NULL, TRUE) ;
                return 0 ;
        }
        break ;

    case WM_PAINT:
        BeginPaint (hwnd, &ps) ;
        EndPaint (hwnd, &ps) ;

        PaintWindow (hwnd, iCurrentColor, iCurrentFigure) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND      hCtrlBlock ;
    static int       iColor, iFigure ;

    switch (message)
    {
        case WM_INITDIALOG:
            iColor          = iCurrentColor ;
            iFigure         = iCurrentFigure ;

            CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, iColor) ;
            CheckRadioButton (hDlg, IDC_RECT, IDC_ELLIPSE, iFigure) ;

            hCtrlBlock = GetDlgItem (hDlg, IDC_PAINT) ;

            SetFocus (GetDlgItem (hDlg, iColor)) ;
            return FALSE ;

        case WM_COMMAND:
            switch (LOWORD (wParam))
            {
                case IDOK:

```

```

        iCurrentColor      = iColor ;
        iCurrentFigure     = iFigure ;
        EndDialog (hDlg, TRUE) ;
        return TRUE ;

    case IDCANCEL:
        EndDialog (hDlg, FALSE) ;
        return TRUE ;

    case IDC_BLACK:
    case IDC_RED:
    case IDC_GREEN:
    case IDC_YELLOW:
    case IDC_BLUE:
    case IDC_MAGENTA:
    case IDC_CYAN:
    case IDC_WHITE:
        iColor = LOWORD (wParam) ;
        CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE,
LOWORD (wParam)) ;
        PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
        return TRUE ;

    case IDC_RECT:
    case IDC_ELLIPSE:
        iFigure = LOWORD (wParam) ;
        CheckRadioButton (hDlg, IDC_RECT,
IDC_ELLIPSE, LOWORD (wParam)) ;
        PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
        return TRUE ;
    }
    break ;

    case WM_PAINT:
        PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
        break ;
    }
    return FALSE ;
}

ABOUT2.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 200, 234

```

```

STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
FONT 8, "MS Sans Serif"
BEGIN
    ICON
    "ABOUT2", IDC_STATIC, 7, 7, 20, 20
    CTEXT        "About2", IDC_STATIC, 57, 12, 86, 8
    CTEXT        "About Box Demo Program", IDC_STATIC, 7, 40, 186, 8
    LTEXT        "", IDC_PAINT, 114, 67, 74, 72
    GROUPBOX                                "&Color", IDC_STATIC, 7, 60, 84, 143
    RADIOBUTTON                                "&Black", IDC_BLACK, 16, 76, 64, 8, WS_GROUP
WS_TABSTOP
    RADIOBUTTON                                "B&blue", IDC_BLUE, 16, 92, 64, 8
    RADIOBUTTON                                "&Green", IDC_GREEN, 16, 108, 64, 8
    RADIOBUTTON                                "Cya&n", IDC_CYAN, 16, 124, 64, 8
    RADIOBUTTON                                "&Red", IDC_RED, 16, 140, 64, 8
    RADIOBUTTON                                "&Magenta", IDC_MAGENTA, 16, 156, 64, 8
    RADIOBUTTON                                "&Yellow", IDC_YELLOW, 16, 172, 64, 8
    RADIOBUTTON                                "&White", IDC_WHITE, 16, 188, 64, 8
    GROUPBOX
    "&Figure", IDC_STATIC, 109, 156, 84, 46, WS_GROUP
    RADIOBUTTON
    "Rec&tangle", IDC_RECT, 116, 172, 65, 8, WS_GROUP | WS_TABSTOP
    RADIOBUTTON                                "&Ellipse", IDC_ELLIPSE, 116, 188, 64, 8
    DEFPUSHBUTTON                                "OK", IDOK, 35, 212, 50, 14, WS_GROUP
    PUSHBUTTON
    "Cancel", IDCANCEL, 113, 212, 50, 14, WS_GROUP
END

////////////////////////////////////
/
// Icon
ABOUT2      ICON      DISCARDABLE      "About2.ico"

////////////////////////////////////
/
// Menu
ABOUT2      MENU DISCARDABLE
BEGIN
    POPUP "&Help"
    BEGIN
        MENUITEM "&About",          IDM_APP_ABOUT
    END
END
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by About2.rc

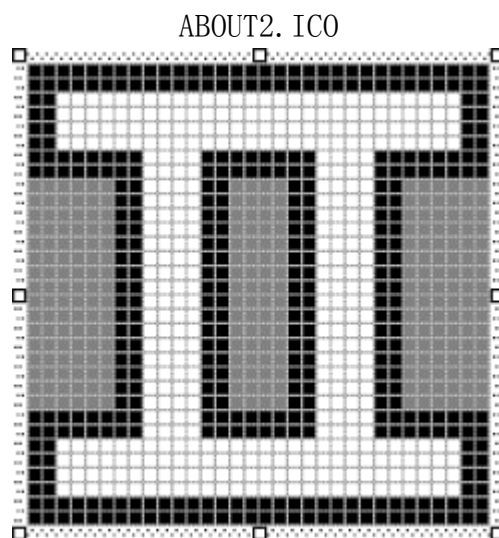
#define IDC_BLACK      1000

```

```

#define IDC_BLUE      1001
#define IDC_GREEN     1002
#define IDC_CYAN      1003
#define IDC_RED       1004
#define IDC_MAGENTA   1005
#define IDC_YELLOW    1006
#define IDC_WHITE     1007
#define IDC_RECT      1008
#define IDC_ELLIPSE   1009
#define IDC_PAINT     1010
#define IDM_APP_ABOUT 40001
#define IDC_STATIC    -1

```



ABOUT2 中的 About 框有两组单选按钮。一组用来选择颜色，另一组用来选择是矩形还是椭圆形。所选的矩形或者椭圆显示在对话方块内，其内部以目前选择的颜色著色。使用者按下「OK」按钮後，对话方块会终止，程式的视窗讯息处理程式在它自己的显示区域内绘出所选图形。如果您按下「Cancel」，则主视窗的显示区域会保持原样。对话方块如图 11-2 所示。尽管 ABOUT2 使用预先定义的识别字 IDOK 和 IDCANCEL 作为两个按键，但是每个单选按钮均有自己的识别字，它们以字首 IDC 开头（用於控制项的 ID）。这些识别字在 RESOURCE.H 中定义。

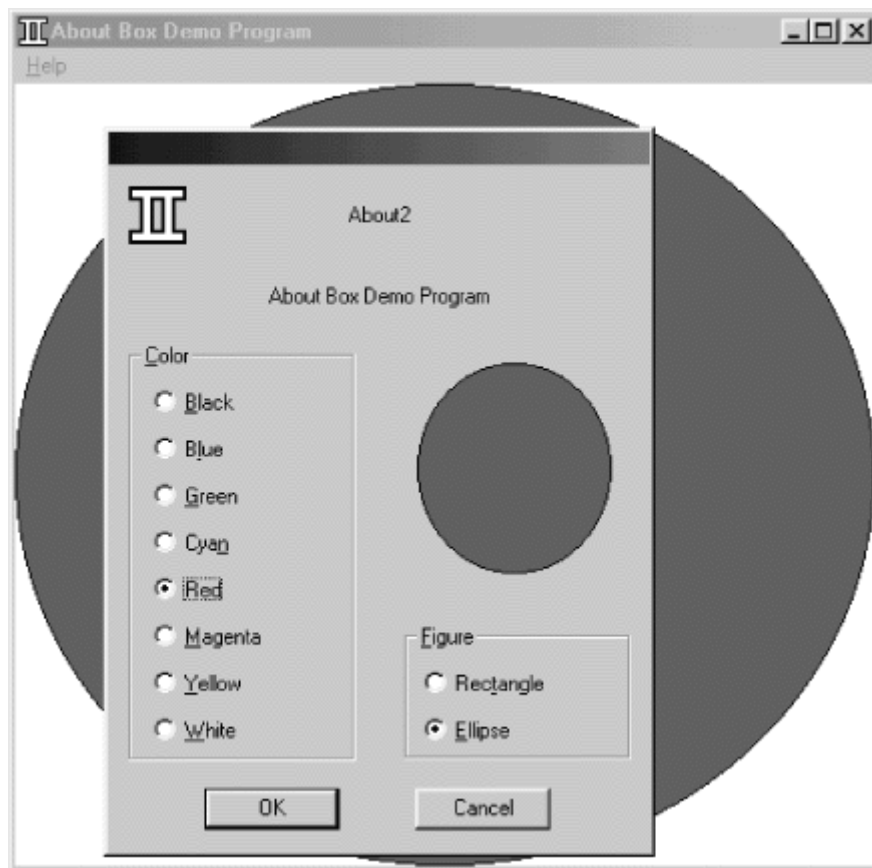


图 11-2 ABOUT2 程式的对话方块

当您在 ABOUT2 对话方块中建立单选按钮时，请按显示顺序建立。这能保证 Developer Studio 依照顺序定义识别字的值，程式将使用这些值。另外，每个单选按钮都不要选中「Auto」选项。「Auto Radio Button」需要的程式码较少，但基本上处理起来更深奥些。然後请依照 ABOUT2.RC 中的定义来设定它们的识别字。

选中「Properties」对话方块中下列物件的「Group」选项：「OK」和「Cancel」按钮、「Figure」分组方块、每个分组方块中的第一个单选按钮（「Black」和「Rectangle」）。选中这两个单选按钮的「Tab Stop」核取方块。

当您有全部控制项在对话方块中的近似位置和大小，就可以从「Layout」功能表选择「Tab Order」选项。按 ABOUT2.RC 资源描述中显示的顺序单击每一个控制项。

使用对话方块控制项

在第九章中，您会发现大多数子视窗控制项发送 WM_COMMAND 讯息给其父视窗（唯一例外的是滚动列控制项）。您还看到，经由发送讯息给子视窗控制项，父视窗可以改变子视窗控制项的状态（例如，选择或不选择单选按钮、核取方块）。您也可以类似方法在对话方块程序中改变控制项。例如，如果您设计了一系列单选按钮，就可以发送讯息给它们，以选择或者不选择这些按钮。不

过, Windows 也提供了几种使用对话方块控制项的简单办法。我们来看一看对话方块程序与子视窗控制项相互通信的方式。

ABOUT2 的对话方块模板显示在程式 11-2 的 ABOUT2.RC 资源描述档中。GROUPBOX 控制项只是一个带标题 (标题为「Color」或者「Figure」) 的分组方块, 每组单选按钮都由这样的分组方块包围。前一组的八个单选按钮是互斥的, 第二组的两个单选按钮也是如此。

当用滑鼠单击其中一个单选按钮时 (或者当单选按钮拥有输入焦点时按空白键), 子视窗向其父视窗发送一个 WM_COMMAND 讯息, 讯息的 wParam 的低字组被设为控制项的 ID, wParam 的高字组是一个通知码, lParam 值是控制项的视窗代号。对于单选按钮, 这个通知码是 BN_CLICKED 或者 0。然后 Windows 中的对话方块视窗讯息处理程式将这个 WM_COMMAND 讯息发送给 ABOUT2.C 内的对话方块程序。当对话方块程序收到一个单选按钮的 WM_COMMAND 讯息时, 它为此按钮设定选中标记, 并为组中其他按钮清除选中标记。

您可能还记得在第九章中已经提过, 选中和不选中按钮均需要向子视窗控制项发送 BM_CHECK 讯息。要设定一个按钮选中标记, 您可以使用:

```
SendMessage (hwndCtrl, BM_SETCHECK, 1, 0) ;
```

要消除选中标记, 您可以使用:

```
SendMessage (hwndCtrl, BM_SETCHECK, 0, 0) ;
```

其中 hwndCtrl 参数是子视窗按钮控制项的视窗代号。

但是在对话方块程序中使用这种方法是时有点问题的, 因为您不知道所有单选按钮的视窗代号, 只是从您获得的讯息中知道其中一个代号。幸运的是, Windows 为您提供了一个函式, 可以用对话方块代号和控制项 ID 来取得一个对话方块控制项的视窗代号:

```
hwndCtrl = GetDlgItem (hDlg, id) ;
```

(您也可以使用如下函式, 从视窗代号中取得控制项的 ID 值:

```
id = GetWindowLong (hwndCtrl, GWL_ID) ;
```

但是在大多数情况下这是不必要的。)

您会注意到, 在程式 11-2 所示的表头档案 ABOUT2.H 中, 八种颜色的 ID 值是从 IDC_BLACK 到 IDC_WHITE 连续变化的, 这种安排在处理来自单选按钮的 WM_COMMAND 讯息时将会很有用。在第一次尝试选中或者不选中单选按钮时, 您可能会在对话方块程序中编写如下的程式:

```
static int iColor ;
其他行程式
case WM_COMMAND:
    switch (LOWORD (wParam))
    {
        其他行程式
```

```

case IDC_BLACK:
case IDC_RED:
case IDC_GREEN:
case IDC_YELLOW:
case IDC_BLUE:
case IDC_MAGENTA:
case IDC_CYAN:
case IDC_WHITE:
    iColor = LOWORD (wParam) ;

    for (i = IDC_BLACK, i <= IDC_WHITE, i++)
        SendMessage (GetDlgItem (hDlg, i),
            BM_SETCHECK, i == LOWORD (wParam), 0) ;
    return TRUE ;

```

其他行程式

这种方法能让人满意地执行。您将新的颜色值储存在 iColor 中，并且还建立了一个回圈，轮流使用所有八种颜色的 ID 值。您取得每个单选按钮控制项的视窗代号，并用 SendMessage 给每个代号发送一条 BM_SETCHECK 讯息。只有对於向对话方块视窗讯息处理程式发送 WM_COMMAND 讯息的按钮，这个讯息的 wParam 值才被设定为 1。

第一种简化的方法是使用专门的对话方块程序 SendDlgItemMessage:

```
SendDlgItemMessage (hDlg, id, iMsg, wParam, lParam) ;
```

它相同於:

```
SendMessage (GetDlgItem (hDlg, id), id, wParam, lParam) ;
```

现在，回圈将变成这样:

```

for (i = IDC_BLACK, i <= IDC_WHITE, i++)
    SendDlgItemMessage (hDlg, i, BM_SETCHECK, i == LOWORD (wParam), 0) ;

```

稍微有些改进。但是真正的重大突破要等到使用了 CheckRadioButton 函式时才会出现:

```
CheckRadioButton (hDlg, idFirst, idLast, idCheck) ;
```

这个函式将 ID 在 idFirst 到 idLast 之间的所有单选按钮的选中标记都清除掉，除了 ID 为 idCheck 的单选按钮，因为它是被选中的。这里，所有 ID 必须是连续的。从此我们可以完全摆脱回圈，并使用:

```
CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, LOWORD (wParam)) ;
```

这正是 ABOUT2 对话方块程序所采用的方法。

在使用核取方块时，也提供了类似的简化函式。如果您建立了一个「CHECKBOX」对话方块视窗控制项，那么可以使用如下的函式来设定和清除选中标记:

```
CheckDlgButton (hDlg, idCheckbox, iCheck) ;
```

如果 iCheck 设定为 1，那么按钮被选中；如果设定为 0，那么按钮不被选

中。您可以使用如下的方法来取得对话方块中某个核取方块的状态：

```
iCheck = IsDlgButtonChecked (hDlg, idCheckbox) ;
```

在对话方块程序中，您既可以将选中标记的目前状态储存在一个静态变数中，又可以在收到一个 WM_COMMAND 讯息後，使用如下方法触发按钮：

```
CheckDlgButton (hDlg, idCheckbox,  
    !IsDlgButtonChecked (hDlg, idCheckbox)) ;
```

如果您定义了 BS_AUTOCHECKBOX 控制项，那么完全没有必要处理 WM_COMMAND 讯息。在终止对话方块之前，您只要使用 IsDlgButtonChecked 就可以取得按钮目前的状态。不过，如果您使用 BS_AUTORADIOBUTTON 样式，那么 IsDlgButtonChecked 就不能令人满意了，因为需要为每个单选按钮都呼叫它，直到函式传回 TRUE。实际上，您还要拦截 WM_COMMAND 讯息来追踪按下的按钮。

「OK」和「Cancel」按钮

ABOUT2 有两个按键，分别标记为「OK」和「Cancel」。在 ABOUT2.RC 的对话方块模板中，「OK」按钮的 ID 值为 IDOK（在 WINUSER.H 中被定义为 1），「Cancel」按钮的 ID 值为 IDCANCEL（定义为 2），「OK」按钮是内定的：

```
DEFPUSHBUTTON        "OK", IDOK, 35, 212, 50, 14  
PUSHBUTTON            "Cancel", IDCANCEL, 113, 212, 50, 14
```

在对话方块中，通常都这样安排「OK」和「Cancel」按钮：将「OK」按钮作为内定按钮有助於用键盘介面终止对话。一般情况下，您通过单击两个滑鼠按键之一，或者当所期望的按钮具有输入焦点时按下 Spacebar 来终止对话方块。不过，如果使用者按下 Enter，对话方块视窗讯息处理程式也将产生一个 WM_COMMAND 讯息，而不管哪个控制项具有输入焦点。wParam 的低字组被设定为对话方块中内定按键的 ID 值，除非另一个按钮拥有输入焦点。在後一种情况下，wParam 的低字组被设定为具有输入焦点之按键的 ID 值。如果对话方块中没有内定按键，那么 Windows 向对话方块程序发送一个 WM_COMMAND 讯息，讯息中 wParam 的低字组被设定为 IDOK。如果使用者按下 Esc 键或者 Ctrl-Break 键，那么 Windows 令 wParam 等於 IDCANCEL，并给对话方块程序发送一个 WM_COMMAND 讯息。所以，您不用在对话方块程序中加入单独的处理键盘操作，因为通常终止对话方块的按键会由 Windows 将这两个按键动作转换为 WM_COMMAND 讯息。

AboutDlgProc 函式通过呼叫 EndDialog 来处理这两种 WM_COMMAND 讯息：

```
switch (LWORD (wParam))  
{  
case IDOK:  
    iCurrentColor = iColor ;  
    iCurrentFigure = iFigure ;  
    EndDialog (hDlg, TRUE) ;
```



```

        return TRUE ;

case IDCANCEL :
    EndDialog (hDlg, FALSE) ;
    return TRUE ;

```

ABOUT2 的视窗讯息处理程式在程式的显示区域中绘制矩形或椭圆时，使用了整体变数 `iCurrentColor` 和 `iCurrentFigure`。AboutDlgProc 在对话方块中画图时使用了静态区域变数 `iColor` 和 `iFigure`。

注意 `EndDialog` 的第二个参数的值不同，这个值是在 `WndProc` 中作为原 `DialogBox` 函式的传回值传回的：

```

case IDM_ABOUT:
    if (DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc))
        InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

```

如果 `DialogBox` 传回 `TRUE`（非 0），则意味著按下了「OK」按钮，然後需要使用新的颜色来更新 `WndProc` 显示区域。当 `AboutDlgProc` 收到一个 `WM_COMMAND` 讯息并且讯息的 `wParam` 的低字组等於 `IDOK` 时，`AboutDlgProc` 将图形和颜色储存在整体变数 `iCurrentColor` 和 `iCurrentFigure` 中。如果 `DialogBox` 传回 `FALSE`，则主视窗继续使用 `iCurrentColor` 和 `iCurrentFigure` 的原始设定。

`TRUE` 和 `FALSE` 通常用於 `EndDialog` 呼叫中，以告知主视窗讯息处理程式使用者是用「OK」还是用「Cancel」来终止对话方块的。不过，`EndDialog` 的参数实际上是一个 `int` 值，而 `DialogBox` 也传回一个 `int` 值。所以，用这种方法能比仅用 `TRUE` 或者 `FALSE` 传回更多的资讯。

避免使用整体变数

在 ABOUT2 中使用整体变数可能会、也可能不会影响您。一些程式写作者（包括我自己）较喜欢少用整体变数。ABOUT2 中的整体变数 `iCurrentColor` 和 `iCurrentFigure` 看来使用得完全合法，因为它们必须同时在视窗讯息处理程式和对话方块程序中使用。不过，在一个有一大堆对话方块的程式中，每个对话方块都可能改变一堆变数的值，使整体变数的数量容易用得过多。

您可能更喜欢将程式中的对话方块与资料结构相联系，该资料结构含有对话方块可以改变的所有变数。您将在 `typedef` 叙述中定义这些结构。例如，在 ABOUT2 中，可以定义与「About」方块相联系的结构：

```

typedef struct
{
    int iColor, iFigure ;
}
ABOUTBOX_DATA ;

```

在 WndProc 中，您可以依据此结构来定义并初始化一个静态变数：

```
static ABOUTBOX_DATA ad = { IDC_BLACK, IDC_RECT } ;
```

在 WndProc 中也是这样，用 ad.iColor 和 ad.iFigure 替换了所有的 iCurrentColor 和 iCurrentFigure。呼叫对话方块时，使用 DialogBoxParam 而不用 DialogBox。此函式的第五个参数可以是任意的 32 位元值。一般来说，此值设定为指向一个结构的指标，在这里是 WndProc 中的 ABOUTBOX_DATA 结构。

```
case IDM_ABOUT:
    if (DialogBoxParam (hInstance, TEXT ("AboutBox"),
        hwnd, AboutDlgProc, &ad))
        InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
```

这是关键：DialogBoxParam 的最後一个参数是作为 WM_INITDIALOG 讯息中的 lParam 传递给对话方块程序的。

对话方块程序有两个 ABOUTBOX_DATA 结构型态的静态变数（一个结构和一个指向结构的指标）：

```
static ABOUTBOX_DATA ad, * pad ;
```

在 AboutDlgProc 中，此定义代替了 iColor 和 iFigure 的定义。在 WM_INITDIALOG 讯息的开始部分，对话方块程序根据 lParam 设定了这两个变数的值：

```
pad = (ABOUTBOX_DATA *) lParam ;
ad = * pad ;
```

第一道叙述中，pad 设定为 lParam 的指标。亦即，pad 实际是指向在 WndProc 定义的 ABOUTBOX_DATA 结构。第二个参数完成了从 WndProc 中的结构，到 DlgProc 中的区域结构的栏位对栏位内容复制。

现在，除了使用者按下「OK」按钮时所用的程式码以之外，所有的 AboutDlgProc 都用 ad.iColor 和 ad.iFigure 替换了 iFigure 和 iColor。这时，将区域结构的内容复制回 WndProc 中的结构：

```
case IDOK:
    * pad = ad ;
    EndDialog (hDlg, TRUE) ;
    return TRUE ;
```

Tab 停留和分组

在第九章，我们利用视窗子类别化为 COLORS1 增加功能，使我们能够按下 Tab 键从一个卷动列转移到另一个卷动列。在对话方块中，视窗子类别化是不必要的，因为 Windows 完成了将输入焦点从一个控制项移动到另一个控制项的所有工作。尽管如此，您必须在对话方块模板中使用 WS_TABSTOP 和 WS_GROUP 视窗样式达到此目的。对于所有想要使用 Tab 键存取的控制项，都要在其视窗样

式中指定 WS_TABSTOP。

如果参阅表 11-1,您就会注意到许多控制项将 WS_TABSTOP 定义为内定样式,其他一些则没有将它作为内定样式。一般而言,不包含 WS_TABSTOP 样式的控制项(特别是静态控制项)不应该取得输入焦点,因为即使有了输入焦点,它们也不能完成操作。除非在处理 WM_INITDIALOG 讯息时您将输入焦点设定给一个特定的控制项,并从讯息中传回 FALSE。否则 Windows 将输入焦点设定为对话方块内第一个具有 WS_TABSTOP 样式的控制项。

Windows 给对话方块增加的第二个键盘介面包括游标移动键,这种介面对於单选按钮有特殊的重要性。如果您使用 Tab 键移动到某一组内目前选中的单选按钮,那么,就需要使用游标移动键,将输入焦点从该单选按钮移动到组内其他单选按钮上。使用 WS_GROUP 视窗样式即可获得这个功能。对於对话方块模板中的特定控制项序列,Windows 将使用游标移动键把输入焦点从第一个具有 WS_GROUP 样式的控制权切换到下一个具有 WS_GROUP 样式的控制项中。如果有必要,Windows 将从对话方块的最後一个控制项回圈到第一个控制项,以便找到分组的结尾。

在内定设定下,控制项 LTEXT、CTEXT、RTEXT 和 ICON 包含有 WS_GROUP 样式,这种样式方便地标记了分组的结尾。您必须经常将 WS_GROUP 样式加到其他型态的控制项中。

让我们来看一看 ABOUT2.RC 中的对话方块模板。四个具有 WS_TABSTOP 样式的控制项是每个组的第一个单选按钮(明显地包含)和两个按钮(内定设定)。在第一次启动对话方块时,您可以使用 Tab 键在这四个控制项之间移动。

在每组单选按钮中,您可以使用游标移动键切换输入焦点并改变选中标记。例如, **Color** 下拉式清单方块的第一个单选按钮(**Black**)和 **Figure** 下拉式清单方块都具有 WS_GROUP 样式。这意味著您可以用游标移动键将焦点从「Black」单选按钮移动到 **Figure** 分组方块中。类似的情形, **Figure** 分组方块的第一个单选按钮(**Rectangle**)和 DEFPUSHBUTTON 都具有 WS_GROUP 样式,所以您可以使用游标移动键在组内两个单选按钮—— **Rectangle** 和 **Ellipse** 之间移动。两个按钮都有 WS_GROUP 样式,以阻止游标移动键在按钮具有输入焦点时起作用。

使用 ABOUT2 时,Windows 的对话方块管理器在两组单选按钮中完成一些相当复杂的处理。正如所预期的那样,处於单选按钮组内时,游标移动键切换输入焦点,并给对话方块程序发送 WM_COMMAND 讯息。但是,当您改变了组内选中的单选按钮时,Windows 也给新选中的单选按钮设定了 WS_TABSTOP 样式。当您下一次使用 Tab 切换到这一组後,Windows 将会把输入焦点设定为选中的单选按

钮。

文字栏位中的「&」将导致紧跟其後的字母以底线显示，这就增加了另一种键盘介面，您可以通过按底线字母来将输入焦点移动到任意单选按钮上。透过按下 C（代表 **Color** 下拉式清单方块）或者 F（代表 **Figure** 下拉式清单方块），您可以将输入焦点移动到相对应组内目前选中的单选按钮上。

尽管程式写作者通常让对话方块管理器来完成这些工作，但是 Windows 提供了两个函式，以便程式写作者找寻下一个或者前一个 Tab 键停留项或者组项。这些函式为：

```
hwndCtrl = GetNextDlgTabItem (hDlg, hwndCtrl, bPrevious) ;
```

和

```
hwndCtrl = GetNextDlgGroupItem (hDlg, hwndCtrl, bPrevious) ;
```

如果 bPrevious 为 TRUE，那么函式传回前一个 Tab 键停留项或组项；如果为 FALSE，则传回下一个 Tab 键停留项或者组项。

在对话方块上画图

ABOUT2 还完成了一些相对说来很特别的事情，亦即在对话方块上画图。让我们来看一看它是怎样做的。在 ABOUT2.RC 的对话方块模板内，使用位置和大小为我们想要画图的区域定义了一块空白文字控制项：

```
LTEXT "" IDC_PAINT, 114, 67, 72, 72
```

这个区域为 18 个字元宽和 9 个字元高。由於这个控制项没有文字，所以视窗讯息处理程式为「静态」类别所做的工作，只是在必须重绘这个子视窗控制项时清除其背景。

在目前颜色或图形选择发生改变，或者对话方块自身获得一个 WM_PAINT 讯息时，对话方块程序呼叫 PaintTheBlock，这个函式在 ABOUT2.C 中：

```
PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
```

在 AboutDlgProc 中，视窗代号 hCtrlBlock 已经在处理 WM_INITDIALOG 讯息时被设定：

```
hCtrlBlock = GetDlgItem (hDlg, IDD_PAINT) ;
```

下面是 PaintTheBlock 函式：

```
void PaintTheBlock (HWND hCtrl, int iColor, int iFigure)
{
    InvalidateRect (hCtrl, NULL, TRUE) ;
    UpdateWindow (hCtrl) ;
    PaintWindow (hCtrl, iColor, iFigure) ;
}
```

这个函式使得子视窗控制项无效，并为控制项视窗讯息处理程式产生一个 WM_PAINT 讯息，然後呼叫 ABOUT2 中的另一个函式 PaintWindow 。

PaintWindow 函式取得一个装置内容代号, 并将其放到 hCtrl 中, 画出所选图形, 根据所选颜色用一个著色画刷填入图形。子视窗控制项的大小从 GetClientRect 获得。尽管对话方块模板以字元为单位定义了控制项的大小, 但 GetClientRect 取得以图素为单位的尺寸。您也可以使用函式 MapDialogRect 将对话方块中的字元坐标转换为显示区域中的图素坐标。

我们并非真的绘制了对话方块的显示区域, 实际绘制的是子视窗控制项的显示区域。每当对话方块得到一个 WM_PAINT 讯息时, 就令子视窗控制项的显示区域失效, 并更新它, 使它确信现在其显示区域又有效了, 然後在其上画图。

将其他函式用於对话方块

大多数可以用在子视窗的函式也可以用於对话方块中的控制项。例如, 如果您想捣乱的话, 那么可以使用 MoveWindow 在对话方块内移动控制项, 强迫使用者用滑鼠来追踪它们。

有时, 您需要根据其他控制项的设定, 动态地启用或者禁用某些控制项, 这需要呼叫:

```
EnableWindow (hwndCtrl, bEnable) ;
```

当 bEnable 为 TRUE (非 0) 时, 它启用控制项; 当 bEnable 为 FALSE (0) 时, 它禁用控制项。在控制项被禁用时, 它不再接收键盘或者滑鼠输入。您不能禁用一个拥有输入焦点的控制项。

定义自己的控制项

尽管 Windows 承揽了许多维护对话方块和子视窗控制项的工作, 它同时也为您提供了各种加入程式码的方法。前面我们已经看到了在对话方块上绘图的方法。您也可以使用第九章中讨论的视窗子类别化来改变子视窗控制项的操作。

您还可以定义自己的子视窗控制项, 并将它们用到对话方块中。例如, 假定您特别不喜欢普通的矩形按键, 而倾向於建立椭圆形按键, 那么您可以通过注册一个视窗类别, 并使用自己编写的视窗讯息处理程式处理来自您所建立视窗的讯息, 从而建立椭圆形按键。在 Developer Studio 中, 您可以在与自订控制项相联系的「Properties」对话方块中指定这个视窗类别, 这将转换成对话方块模板中的 CONTROL 叙述。程式 11-3 所示的 ABOUT3 程式正是这样做的。

程式 11-3 ABOUT3

```
ABOUT3.C
/*-----
---
ABOUT3.C -- About Box Demo Program No. 3
(c) Charles Petzold, 1998
```

```

-----
*/

#include <windows.h>
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK EllipPushWndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("About3") ;
    MSG msg ;
    HWND hwnd ;
    WNDCLASS wndclass ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (hInstance,
szAppName) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox ( NULL, TEXT ("This program requires Windows
NT!"),
szAppName,
MB_ICONERROR) ;
        return 0 ;
    }

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = EllipPushWndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = NULL ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1) ;
    wndclass.lpszMenuName = NULL ;

```

```
wndclass.lpszClassName      = TEXT ("EllipPush") ;

RegisterClass (&wndclass) ;
hwnd = CreateWindow (  szAppName, TEXT ("About Box Demo Program"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static HINSTANCE hInstance ;
    switch (message)
    {
        case WM_CREATE :
            hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
            return 0 ;

        case WM_COMMAND :
            switch (LOWORD (wParam))
            {
                case IDM_APP_ABOUT :
                    DialogBox (hInstance, TEXT ("AboutBox"), hwnd,
AboutDlgProc) ;

                    return 0 ;

            }
            break ;

        case WM_DESTROY :
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
```

```

{
    switch (message)
    {
        case WM_INITDIALOG :
            return TRUE ;

        case WM_COMMAND :
            switch (LOWORD (wParam))
            {
                case IDOK :
                    EndDialog (hDlg, 0) ;
                    return TRUE ;

            }
            break ;
    }
    return FALSE ;
}

LRESULT CALLBACK EllipPushWndProc (      HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    TCHAR                szText[40] ;
    HBRUSH                hBrush ;
    HDC                   hdc ;
    PAINTSTRUCT           ps ;
    RECT                  rect ;

    switch (message)
    {
        case WM_PAINT :
            GetClientRect (hwnd, &rect) ;
            GetWindowText (hwnd, szText, sizeof (szText)) ;

            hdc = BeginPaint (hwnd, &ps) ;

            hBrush = CreateSolidBrush (GetSysColor (COLOR_WINDOW)) ;
            hBrush = (HBRUSH) SelectObject (hdc, hBrush) ;
            SetBkColor (hdc, GetSysColor (COLOR_WINDOW)) ;
            SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;

            Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
            DrawText (hdc, szText, -1, &rect,
DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

            DeleteObject (SelectObject (hdc, hBrush)) ;

            EndPaint (hwnd, &ps) ;
            return 0 ;
    }
}

```



```

        case WM_KEYUP :
            if (wParam != VK_SPACE)
                break ;// fall through
        case WM_LBUTTONDOWN :
            SendMessage (GetParent (hwnd), WM_COMMAND,
                GetWindowLong (hwnd, GWL_ID), (LPARAM) hwnd) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

ABOUT3.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL                "OK", IDOK, "EllipPush", WS_GROUP
WS_TABSTOP, 73, 79, 32, 14
    ICON                    "ABOUT3", IDC_STATIC, 7, 7, 20, 20
    CTEXT                    "About3", IDC_STATIC, 40, 12, 100, 8
    CTEXT                    "About Box Demo Program", IDC_STATIC, 7, 40, 166, 8
    CTEXT                    "(c) Charles Petzold,
1998", IDC_STATIC, 7, 52, 166, 8
END

////////////////////////////////////
/
// Menu
ABOUT3 MENU DISCARDABLE
BEGIN
    POPUP "&Help"
    BEGIN
        MENUITEM "&About About3...",
IDM_APP_ABOUT
    END
END

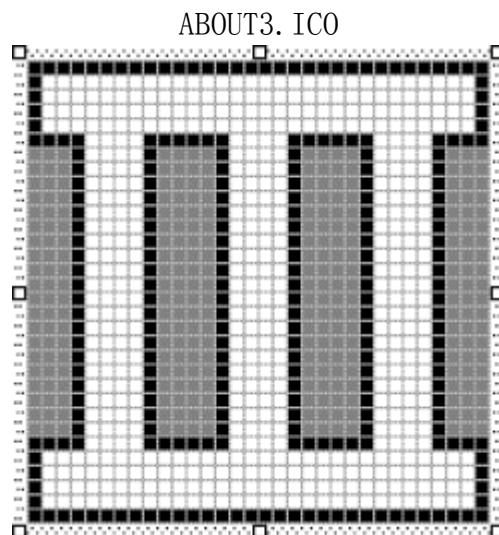
////////////////////////////////////
/
// Icon
ABOUT3 ICON DISCARDABLE "icon1.ico"

```

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
// Used by About3.rc

#define IDM_APP_ABOUT      40001
#define IDC_STATIC        -1
```



我们所注册的视窗类别叫做「EllipPush」（椭圆形按键）。在 Developer Studio 的对话方块编辑器中，删除「Cancel」和「OK」按钮。要添加依据此视窗类别的控制项，请从「[Controls](#)」工具列选择「[Custom Control](#)」。在此控制项的「[Properties](#)」对话方块的「[Class](#)」栏位输入「[EllipPush](#)」。在对话方块模板中我们没有使用 DEFPUSHBUTTON 叙述，而是用 CONTROL 叙述来指定此视窗类别：

```
CONTROL "OK" IDOK, "EllipPush", TABGRP, 64, 60, 32, 14
```

当在对话方块中建立子视窗控制项时，对话方块管理器把这个视窗类别用於 CreateWindow 呼叫中。

ABOUT3.C 程式在 WinMain 中注册了 EllipPush 视窗类别：

```
wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc    = EllipPushWndProc ;
wndclass.cbClsExtra     = 0 ;
wndclass.cbWndExtra     = 0 ;
wndclass.hInstance     = hInstance ;
wndclass.hIcon          = NULL ;
wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground  = (HBRUSH) (COLOR_WINDOW + 1) ;
wndclass.lpszMenuName   = NULL ;
wndclass.lpszClassName  = TEXT ("EllipPush") ;

RegisterClass (&wndclass) ;
```

该视窗类别指定视窗讯息处理程式为 EllipPushWndProc，在 ABOUT3.C 中正是这样。

EllipPushWndProc 视窗讯息处理程式只处理三种讯息：WM_PAINT、WM_KEYUP 和 WM_LBUTTONUP。在处理 WM_PAINT 讯息时，它从 GetClientRect 中取得视窗的大小，从 GetWindowText 中取得显示在按键上的文字，用 Windows 函式 Ellipse 和 DrawText 来输出椭圆和文字。

WM_KEYUP 和 WM_LBUTTONUP 讯息的处理非常简单：

```
case WM_KEYUP :
    if (wParam != VK_SPACE)
        break ;    // fall through
case WM_LBUTTONUP :
    SendMessage (GetParent (hwnd), WM_COMMAND,
        GetWindowLong (hwnd, GWL_ID), (LPARAM) hwnd) ;
    return 0 ;
```

视窗讯息处理程式使用 GetParent 来取得其父视窗（即对话方块）的代号，并发送一个 WM_COMMAND 讯息，讯息的 wParam 等於控制项的 ID，这个 ID 是用 GetWindowLong 取得的。然後，对话方块视窗讯息处理程式将这个讯息传给 ABOUT3 内的对话方块程序，结果得到一个使用者自订的按键，如图 11-3 所示。您可以用同样的方法来建立其他自订对话方块控制项。

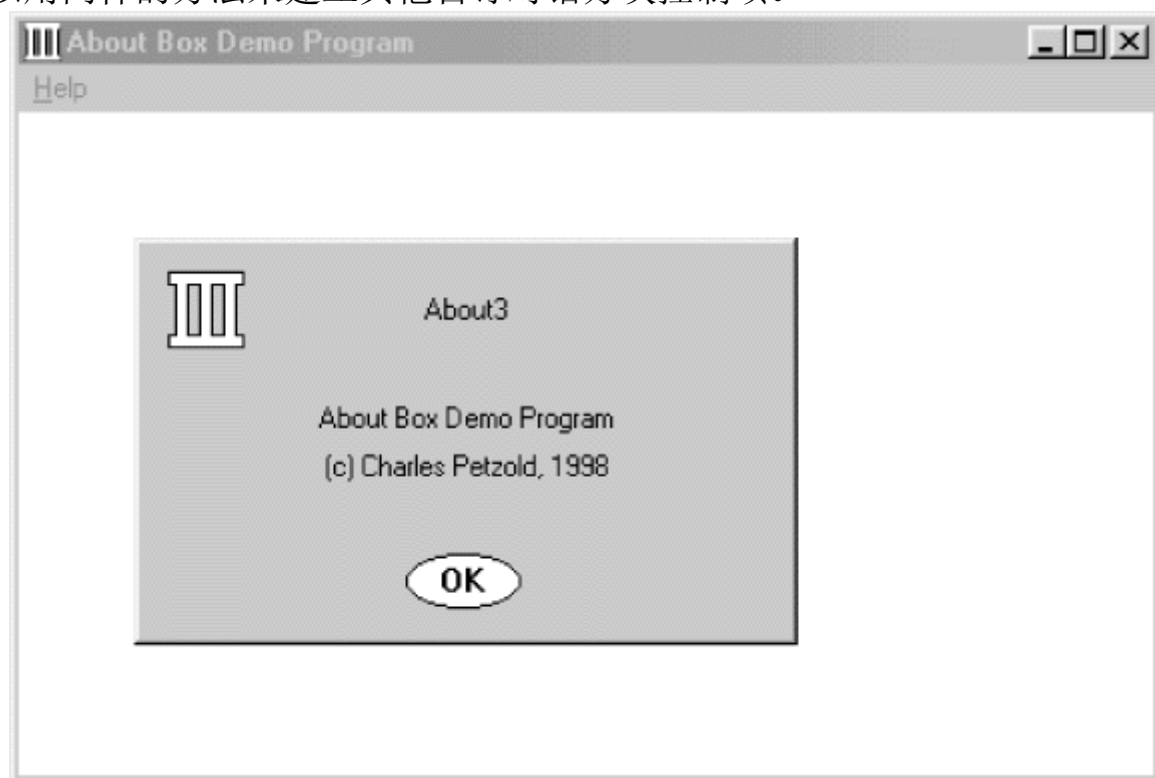


图 11-3 ABOUT3 建立的自订按键

这就是全部要做的吗？其实不然。通常，对于维护子视窗控制项所需要的处理而言，EllipPushWndProc 只是一个空架子。例如，按钮不会像普通的按键那样闪烁。要改变按钮内的颜色，视窗讯息处理程式必须处理 WM_KEYDOWN（来自空白键）和 WM_LBUTTONDOWN 讯息。视窗讯息处理程式还必须在收到

WM_LBUTTONDOWN 讯息时拦截滑鼠，并且，如果当按钮还处于按下状态，而滑鼠移到了子视窗的显示区域之外，那么得要释放滑鼠拦截（并将按钮的内部颜色回复为正常状态）。只有在滑鼠被拦截时松开该按钮，子视窗才会给其父视窗送回一个 WM_COMMAND 讯息。

EllipPushWndProc 也不处理 WM_ENABLE 讯息。如上所述，对话方块程序可以使用 EnableWindow 函式来禁用某视窗。於是，子视窗将显示灰色文字，而不再是黑色文字，以表示它已经被禁用，并且不能再接收任何讯息了。

如果子视窗控制项的视窗讯息处理程式需要为所建立的每个视窗存放各自不同的资料，那么它可以通过使用视窗类别结构中的 cbWndExtra 值来做到。这样就在内部视窗结构中保留了空间，并可以用 SetWindowLong 和 GetWindowLong 来存取该资料。

非模态对话方块

在本章的开始，我曾经说过对话方块分为「模态的」和「非模态的」两种。现在我们已经研究过这两种对话方块中最常见的一种——模态对话方块。模态对话方块（不包括系统模态对话方块）。允许使用者在对话方块与其他程式之间进行切换。但是，使用者不能切换到同一程式的另一个视窗，直到模态对话方块被清除为止。非模态对话方块允许使用者在对话方块与其他程式之间进行切换，又可以在对话方块与建立对话方块的视窗之间进行切换。因此，非模态对话方块与使用者程式常见的普通弹出式视窗可能更为相似。

当使用者觉得让对话方块保留片刻会更加方便时，使用非模态对话方块是合适的。例如，文书处理程式经常使用非模态对话方块来进行「Find」和「Change」操作。如果「Find」对话方块是模态的，那么使用者必须从功能表中选择「Find」，然後输入要寻找的字串，结束对话方块，传回到档案中，接著再重复整个程序来寻找同一字串的另一次出现。允许使用者在档案与对话方块之间进行切换则会方便得多。

您已经看到，模态对话方块是用 DialogBox 来建立的。只有在清除对话方块之後，函式才会传回值。在对话方块程序内使用 EndDialog 呼叫来终止对话方块，DialogBox 传回的是该呼叫的第二个参数的值。非模态对话方块是使用 CreateDialog 来建立的，该函式所使用的参数与 DialogBox 相同。

```
hDlgModeless = CreateDialog (    hInstance, szTemplate,  
                                hwndParent, DialogProc) ;
```

区别是 CreateDialog 函式立即传回对话方块的视窗代号，并通常将这个视窗代号存放到整体变数中。

尽管将 DialogBox 这一名字用於模态对话方块而 CreateDialog 用於非模态

对话方块是随意的，但是您可以通过非模态对话方块与普通视窗类似这一点来记住这两个函式的区别。CreateDialog 可以令人想起 CreateWindow 函式来，而後者建立的是普通视窗。

模态对话方块与非模态对话方块的区别

使用非模态对话方块与使用模态对话方块相似，但是也有一些重要的区别：

首先，非模态对话方块通常包含一个标题列和一个系统功能表按钮。当您在 Developer Studio 中建立对话方块时，这些是内定选项。用於非模态对话方块的对话方块模板中的 STYLE 叙述形如：

STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE

标题列和系统功能表允许使用者，使用滑鼠或者键盘将非模态对话方块移动到另一个显示区域。对於模态对话方块，您通常无须提供标题列和系统功能表，因为使用者不能在其下面的视窗中做任何其他的事情。

第二项重要的区别是：注意，在我们的范例 STYLE 叙述中包含有 WS_VISIBLE 样式。在 **Developer Studio** 中，从「**Dialog Properties**」对话方块的「**More Styles**」页面标签中选择此选项。如果省略了 WS_VISIBLE，那么您必须在 CreateDialog 呼叫之後呼叫 ShowWindow：

```
hDlgModeless = CreateDialog ( . . . ) ;  
ShowWindow (hDlgModeless, SW_SHOW) ;
```

如果您既没有包含 WS_VISIBLE 样式，又没有呼叫 ShowWindow，那么非模态对话方块将不会被显示。如果忽略这个事实，那么习惯於模态对话方块的程式写作者在第一次试图建立非模态对话方块时，经常会出现问题。

第三项区别：与模态对话方块和讯息方块的讯息不同，非模态对话方块的讯息要经过程序式的讯息佇列。要将这些讯息传送给对话方块视窗讯息处理程式，则必须改变讯息佇列。方法如下：当您使用 CreateDialog 建立非模态对话方块时，应该将从呼叫中传回的对话方块代号储存在一个整体变数（如 hDlgModeless）中，并将讯息回圈改变为：

```
while (GetMessage (&msg, NULL, 0, 0))  
{  
    if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))  
    {  
        TranslateMessage (&msg) ;  
        DispatchMessage (&msg) ;  
    }  
}
```

如果讯息是发送给非模态对话方块的，那么 IsDialogMessage 将它发送给对话方块中视窗讯息处理程式，并传回 TRUE（非 0）；否则，它将传回 FALSE

(0)。只有 `hDlgModeless` 为 0 或者讯息不是该对话方块的讯息时，才必须呼叫 `TranslateMessage` 和 `DispatchMessage` 函式。如果您将键盘加速键用於您的程式视窗，那么讯息回圈将如下所示：

```
while (GetMessage (&msg, NULL, 0, 0))
{
    if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))
    {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
}
```

由於整体变数被初始化为 0，所以 `hDlgModeless` 将为 0，直到建立对话方块为止，从而保证不会使用无效的视窗代号来呼叫 `IsDialogMessage`。在清除非模态对话方块时，您也必须注意这一点，正如最後一点所说明的。

`hDlgModeless` 变数也可以由程式的其他部分使用，以便对非模态对话方块是否存在加以验证。例如，程式中的其他视窗可以在 `hDlgModeless` 不等於 0 时给对话方块发送讯息。

最後一项重要的区别：使用 `DestroyWindow` 而不是 `EndDialog` 来结束非模态对话方块。当您呼叫 `DestroyWindow` 後，将 `hDlgModeless` 整体变数设定为 0。

使用者习惯於从系统功能表中选择「Close」来结束非模态对话方块。尽管启用了「Close」选项，Windows 内的对话方块视窗讯息处理程式并不处理 `WM_CLOSE` 讯息。您必须自己在对话方块程序中处理它：

```
case WM_CLOSE :
    DestroyWindow (hDlg) ;
    hDlgModeless = NULL ;
    break ;
```

注意这两个视窗代号之间的区别：`DestroyWindow` 的 `hDlg` 参数是传递给对话方块程序的参数；`hDlgModeless` 是从 `CreateDialog` 传回的整体变数，程式在讯息回圈内检验它。

您也可以允许使用者使用按键来关闭非模态对话方块，处理方式与处理 `WM_CLOSE` 讯息一样。对话方块必须传回给建立它的视窗之任何资料都可以储存在整体变数中。如果不喜欢使用整体变数，那么您也可以用 `CreateDialogParam` 来建立非模态对话方块，并按前面介绍的方法让它储存一个结构指标。

新的 COLORS 程式

第九章中所描述的 COLORS1 程式建立了九个子视窗，以便显示三个滚动列和六个文字项。那时候，这个程式还是我们所写过的程式中相当复杂的一个。如果将 COLORS1 转换为使用非模态对话方块则会使程式——特别是 WndProc 函数——变得令人难以置信的简单，修正後的 COLORS2 程式如程式 11-4 所示。

程式 11-4 COLORS2

```

COLORS2.C
/*-----
--
--      COLORS2.C -- Version using Modeless Dialog Box
--                      (c) Charles Petzold, 1998
--
*/
#include <windows.h>
LRESULT      CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
BOOL         CALLBACK ColorScrDlg      (HWND, UINT, WPARAM, LPARAM) ;

HWND hDlgModeless ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Colors2") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = CreateSolidBrush (0L) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }
}

```

```

    hwnd = CreateWindow (  szAppName, TEXT ("Color Scroll"),
                          WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    hDlgModeless = CreateDialog (      hInstance, TEXT ("ColorScrDlg"),
                                   hwnd, ColorScrDlg) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
        case WM_DESTROY :
            DeleteObject ((HGDIOBJ) SetClassLong (hwnd, GCL_HBRBACKGROUND,
            (LONG) GetStockObject (WHITE_BRUSH))) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK ColorScrDlg (  HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int          iColor[3] ;
    HWND                hwndParent, hCtrl ;
    int                  iCtrlID, iIndex ;

    switch (message)
    {
        case WM_INITDIALOG :
            for (iCtrlID = 10 ; iCtrlID < 13 ; iCtrlID++)

```



```

        {
            hCtrl = GetDlgItem (hDlg, iCtrlID) ;
            SetScrollRange (hCtrl, SB_CTL, 0, 255,
FALSE) ;

            SetScrollPos (hCtrl, SB_CTL, 0, FALSE) ;
        }
        return TRUE ;

    case WM_VSCROLL :
        hCtrl          = (HWND) lParam ;
        iCtrlID         = GetWindowLong (hCtrl, GWL_ID) ;
        iIndex          = iCtrlID - 10 ;
        hwndParent      = GetParent (hDlg) ;

        switch (LOWORD (wParam))
        {
            case SB_PAGEDOWN :
                iColor[iIndex] += 15 ;          // fall through
            case SB_LINEDOWN :
                iColor[iIndex] = min (255, iColor[iIndex] +
1) ;

                break ;
            case SB_PAGEUP :
                iColor[iIndex] -= 15 ;          // fall through
            case SB_LINEUP :
                iColor[iIndex] = max (0, iColor[iIndex] - 1) ;
                break ;
            case SB_TOP :
                iColor[iIndex] = 0 ;
                break ;
            case SB_BOTTOM :
                iColor[iIndex] = 255 ;
                break ;
            case SB_THUMBPOSITION :
            case SB_THUMBTRACK :
                iColor[iIndex] = HIWORD (wParam) ;
                break ;
            default :
                return FALSE ;
        }

        SetScrollPos (hCtrl, SB_CTL,
iColor[iIndex], TRUE) ;
        SetDlgItemInt (hDlg, iCtrlID + 3, iColor[iIndex], FALSE) ;

        DeleteObject ((HGDIOBJ) SetClassLong (hwndParent,
GCL_HBRBACKGROUND,

            (LONG) CreateSolidBrush (
                RGB (iColor[0], iColor[1], iColor[2])))) ;

```

```

        InvalidateRect (hwndParent, NULL, TRUE) ;
        return TRUE ;
    }
    return FALSE ;
}

COLORS2.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
COLORSCRDLG DIALOG DISCARDABLE 16, 16, 120, 141
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Color Scroll Scrollbars"
FONT 8, "MS Sans Serif"
BEGIN
    CTEXT                                "&Red", IDC_STATIC, 8, 8, 24, 8, NOT
WS_GROUP
    SCROLLBAR                            10, 8, 20, 24, 100, SBS_VERT | WS_TABSTOP
    CTEXT                                "0", 13, 8, 124, 24, 8, NOT WS_GROUP
    CTEXT
    "&Green", IDC_STATIC, 48, 8, 24, 8, NOT WS_GROUP
    SCROLLBAR                            11, 48, 20, 24, 100, SBS_VERT | WS_TABSTOP
    CTEXT                                "0", 14, 48, 124, 24, 8, NOT WS_GROUP
    CTEXT
    "&Blue", IDC_STATIC, 89, 8, 24, 8, NOT WS_GROUP
    SCROLLBAR                            12, 89, 20, 24, 100, SBS_VERT | WS_TABSTOP
    CTEXT                                "0", 15, 89, 124, 24, 8, NOT WS_GROUP
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by Colors2.rc

#define IDC_STATIC      -1

```

原来的 COLORS1 程式所显示的卷动列大小是依据视窗大小决定的，而新程式在非模态对话方块内以固定的尺寸来显示它们，如图 11-4 所示。

当您建立对话方块模板时，直接将三个卷动列的 ID 分别设为 10、11 和 12，将显示卷动列目前值的三个静态文字栏位的 ID 分别设为 13、14 和 15。将每个卷动列都设定为 Tab Stop 样式，而从所有的六个静态文字栏位中删除 Group 样式。

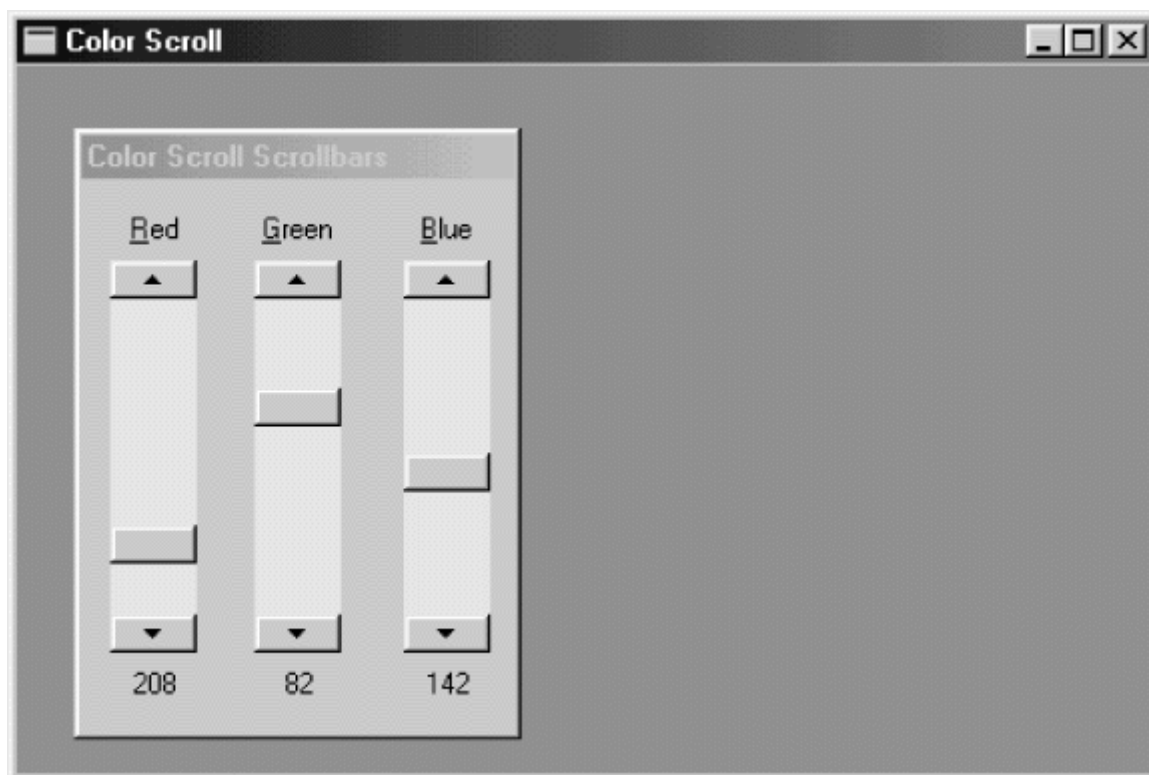


图 11-4 COLORS2 的萤幕显示

在 COLORS2 中，非模态对话方块是在 WinMain 函式里建立的，紧跟在程式主视窗的 ShowWindow 呼叫之後。注意，主视窗的视窗样式包含 WS_CLIPCHILDREN，这允许程式无须擦除对话方块就能够重画主视窗。

如上所述，从 CreateDialog 传回的对话方块视窗代号存放在整体变数 hDlgModeless 中，并在讯息回圈中被测试。不过，在这个程式中，不需要将代号存放在整体变数中，也不需要呼叫 IsDialogMessage 之前测试这个值。讯息回圈可以编写如下：

```
while (GetMessage (&msg, NULL, 0, 0))
{
    if (!IsDialogMessage (hDlgModeless, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
```

由於对话方块是在程式进入讯息回圈前建立，并且直到程式结束时才会被清除，所以 hDlgModeless 的值将总是有效的。我加入了如下的处理方式，以便您可能会往对话方块的视窗讯息处理程式中加入一段清除对话方块的程式码：

```
case WM_CLOSE :
    DestroyWindow (hDlg) ;
    hDlgModeless = NULL ;
    break ;
```

在原来的 COLORS1 程式中，SetWindowText 在使用 sprintf 将三个数值标

签转换为文字之後才设定它们的值。叙述为：

```
wsprintf (szBuffer, TEXT ("%i"), color[i]) ;
SetWindowText (hwndValue[i], szBuffer) ;
```

i 的值为目前处理的卷动列的 ID, hwndValue 是一个阵列, 它包含颜色数值的三个静态文字子视窗的视窗代号。

新版本使用 SetDlgItemInt 为每个子视窗的每个文字栏位设定一个号码：

```
SetDlgItemInt (hDlg, iCtrlID + 3, color [iCtrlID], FALSE) ;
```

尽管 SetDlgItemInt 和与其对应的 GetDlgItemInt 在编辑控制项中用得最多, 它们也可以用来设定其他控制项的文字栏位, 如静态文字控制项等。iCtrlID 变数是卷动列的 ID, 给 ID 加上 3 使之变成对应数字标签的 ID。第三个参数是颜色值。通常, 第四个参数表示第三个参数的值是解释为有正负号的 (第四个参数为 TRUE) 还是无正负号的 (第四个参数为 FALSE)。但是, 对于这个程式, 值的范围是从 0 到 256, 所以这个参数没有意义。

在将 COLORS1 转换为 COLORS2 的程式中, 我们把越来越多的工作交给了 Windows。旧版本呼叫了 CreateWindow 10 次; 而新版本只呼叫了 CreateWindow 和 CreateDialog 各一次。但是, 如果您认为我们已经把呼叫 CreateWindow 的次数降到最少, 那么您就错了, 请看下一个程式。

HEXCALC: 视窗还是对话方块?

HEXCALC 程式可能是写程式偷懒的经典之作, 如程式 11-5 所示。这个程式完全不呼叫 CreateWindow, 也不处理 WM_PAINT 讯息, 不取得装置内容, 也不处理滑鼠讯息。但是它只用了不到 150 行的原始码, 就构成了一个具有完整键盘和滑鼠介面以及 10 种运算的十六进位计算机。计算机如图 11-5 所示。

程式 11-5 HEXCALC

```
HEXCALC.C
/*-----
    HEXCALC.C -- Hexadecimal Calculator
                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR    szAppName[] = TEXT ("HexCalc") ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS         wndclass ;
```

```

    wndclass.style                = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc          = WndProc ;
    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = DLGWINDOWEXTRA ;                //
Note!
    wndclass.hInstance           = hInstance ;
    wndclass.hIcon                = LoadIcon (hInstance, szAppName) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) (COLOR_BTNFACE + 1) ;
    wndclass.lpszMenuName          = NULL ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (        NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateDialog (hInstance, szAppName, 0, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void ShowNumber (HWND hwnd, UINT iNumber)
{
    TCHAR szBuffer[20] ;
    wsprintf (szBuffer, TEXT ("%X"), iNumber) ;
    SetDlgItemText (hwnd, VK_ESCAPE, szBuffer) ;
}

DWORD CalcIt (UINT iFirstNum, int iOperation, UINT iNum)
{
    switch (iOperation)
    {
    case '=': return iNum ;
    case '+': return iFirstNum + iNum ;
    case '-': return iFirstNum - iNum ;
    case '*': return iFirstNum * iNum ;
    case '&': return iFirstNum & iNum ;
    }
}

```

```

        case '|': return iFirstNum | iNum ;
        case '^': return iFirstNum ^ iNum ;
        case '<': return iFirstNum << iNum ;
        case '>': return iFirstNum >> iNum ;
        case '/': return iNum ? iFirstNum / iNum: MAXDWORD ;
        case '%': return iNum ? iFirstNum % iNum: MAXDWORD ;
        default : return 0 ;
    }
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static BOOL        bNewNumber = TRUE ;
    static int         iOperation = '=' ;
    static UINT        iNumber, iFirstNum ;
    HWND               hButton ;

    switch (message)
    {
        case WM_KEYDOWN:                // left arrow --> backspace
            if (wParam != VK_LEFT)
                break ;
            wParam = VK_BACK ;
            // fall through
        case WM_CHAR:
            if    ((wParam = (WPARAM) CharUpper ((TCHAR *) wParam)) ==
VK_RETURN)

                wParam = '=' ;

            if    (hButton = GetDlgItem (hwnd, wParam))
            {
                SendMessage (hButton, BM_SETSTATE, 1, 0) ;
                Sleep (100) ;
                SendMessage (hButton, BM_SETSTATE, 0, 0) ;
            }
            else
            {
                MessageBeep (0) ;
                break ;
            }
            // fall through
        case WM_COMMAND:
            SetFocus (hwnd) ;

            if (LOWORD (wParam) == VK_BACK)                //backspace
                ShowNumber (hwnd, iNumber /= 16) ;
    }
}

```

```

        else if (LOWORD (wParam) == VK_ESCAPE)           // escape
            ShowNumber (hwnd, iNumber = 0) ;

        else if (isxdigit (LOWORD (wParam)))             // hex digit
        {
            if (bNewNumber)
            {
                iFirstNum = iNumber ;
                iNumber = 0 ;
            }
            bNewNumber = FALSE ;
            if (iNumber <= MAXDWORD >> 4)
                ShowNumber (hwnd, iNumber = 16 * iNumber +
wParam -
                    (isdigit (wParam) ? '0': 'A' - 10)) ;
            else
                MessageBeep (0) ;
        }
        else // operation
        {
            if (!bNewNumber)
                ShowNumber (hwnd, iNumber =
                CalcIt (iFirstNum, iOperation, iNumber)) ;
            bNewNumber = TRUE ;
            iOperation = LOWORD (wParam) ;
        }
        return 0 ;
case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

HEXCALC.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Icon

HEXCALC ICON DISCARDABLE

"HexCalc.ico"

////////////////////////////////////
/

#include "hexcalc.dlg"

HEXCALC.DLG

```

/*-----
  HEXCALC.DLG dialog script
-----*/

HexCalc DIALOG -1, -1, 102, 122
STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS "HexCalc"
CAPTION "Hex Calculator"
{
    PUSHBUTTON "D",          68,  8,  24, 14, 14
    PUSHBUTTON "A",          65,  8,  40, 14, 14
        PUSHBUTTON "7",      55,  8,  56, 14, 14
    PUSHBUTTON "4",          52,  8,  72, 14, 14
    PUSHBUTTON "1",          49,  8,  88, 14, 14
    PUSHBUTTON "0",          48,  8, 104,14, 14
    PUSHBUTTON "0",          27, 26,  4,  50, 14
    PUSHBUTTON "E",          69, 26, 24, 14, 14
    PUSHBUTTON "B",          66, 26, 40, 14, 14
    PUSHBUTTON "8",          56, 26, 56, 14, 14
    PUSHBUTTON "5",          53, 26, 72, 14, 14
    PUSHBUTTON "2",          50, 26, 88, 14, 14
    PUSHBUTTON "Back",       8,   26, 104,32, 14
    PUSHBUTTON "C",          67, 44, 40, 14, 14
    PUSHBUTTON "F",          70, 44, 24, 14, 14
    PUSHBUTTON "9",          57, 44, 56, 14, 14
    PUSHBUTTON "6",          54, 44, 72, 14, 14
    PUSHBUTTON "3",          51, 44, 88, 14, 14
    PUSHBUTTON "+",          43, 62, 24, 14, 14
    PUSHBUTTON "-",          45, 62, 40, 14, 14
    PUSHBUTTON "*",          42, 62, 56, 14, 14
    PUSHBUTTON "/",          47, 62, 72, 14, 14
    PUSHBUTTON "%",          37, 62, 88, 14, 14
    PUSHBUTTON "Equals",     61, 62, 104,32, 14
    PUSHBUTTON "&&",38, 80, 24, 14, 14
    PUSHBUTTON "|",          124, 80, 40, 14, 14
    PUSHBUTTON "^",          94, 80, 56, 14, 14
    PUSHBUTTON "<",          60, 80, 72, 14, 14
    PUSHBUTTON ">",          62, 80, 88, 14, 14
}

```

HEXCALC.ICO

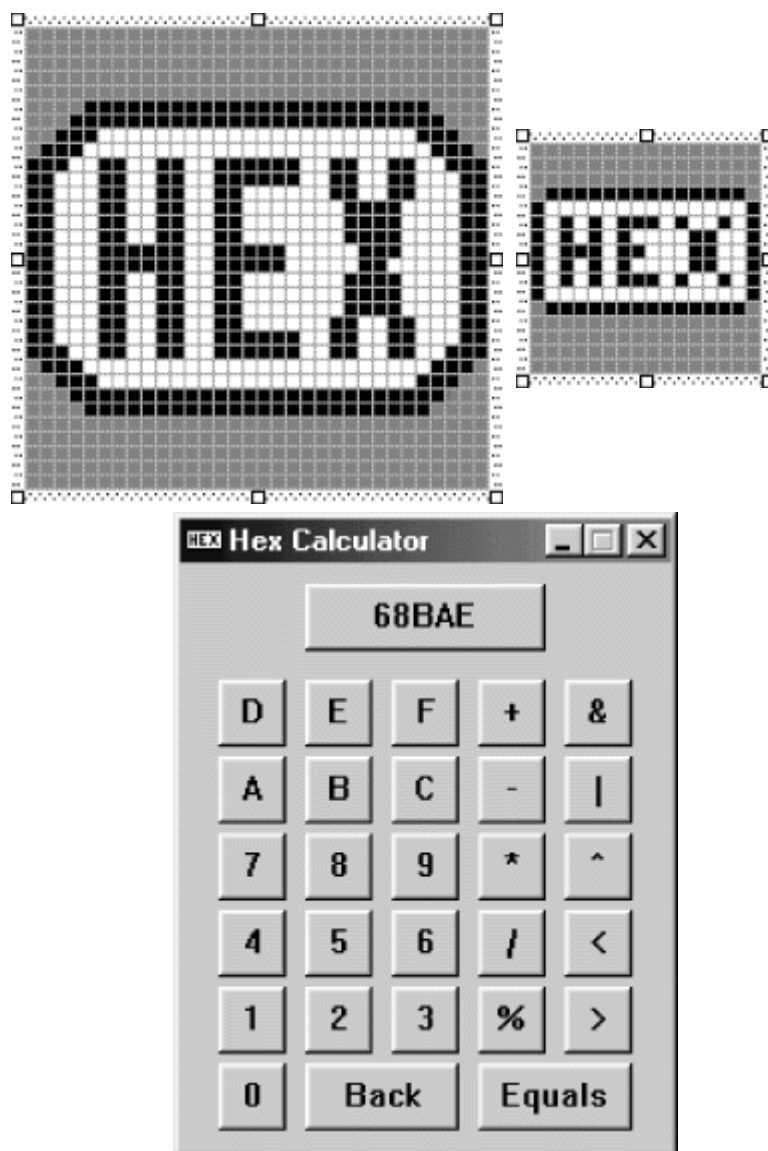


图 11-5 HEXCALC 的萤幕显示

HEXCALC 是一个普通的中序运算式计算机，使用 C 语言的符号表示方式进行计算。它对无正负号 32 位元整数作加、减、乘、除和取余数运算，位元 AND, OR, exclusive-OR 运算，还有左右位移运算。被 0 除将导致结果被设定为 FFFFFFFF。

在 HEXCALC 中既可以使用滑鼠又可以使用键盘。您从按键点入「」或者输入第一个数（最多 8 位元十六进位数位）开始，然後输入运算符，然後是第二个数。接著，您可以透过单击「Equals」按钮或者按下等号键或 Enter 键便可以显示运算结果。为了更正输入，您可以使用「Back」按钮、Backspace 或者左箭头键。单击「display」方块或者按下 Esc 键即可清除目前的输入。

HEXCALC 比较奇怪的一点是，萤幕上显示的视窗似乎是普通的重叠式视窗与非模态对话方块的混合体。一方面，HEXCALC 的所有讯息都在函式的 WndProc 中处理，这个函式与通常的视窗讯息处理程式相似，该函式传回一个长整数，它处理 WM_DESTROY 讯息，呼叫 DefWindowProc，就像普通的视窗讯息处理程式一样。另一方面，视窗是在 WinMain 中呼叫 CreateDialog 并使用 HEXCALC.DLG 中

的对话方块模板建立的。那么，HEXCALC 到底是一个普通的可重叠视窗，还是一个非模态对话方块呢？

简单的回答是，对话方块就是视窗。通常，Windows 使用它自己内部的视窗讯息处理程式处理对话方块视窗的讯息，然後，Windows 将这些讯息传送给建立对话方块的程式内的对话方块程序。在 HEXCALC 中，我们让 Windows 使用对话方块模板建立一个视窗，但是自己写程式处理这个视窗的讯息。

不幸的是，在 Developer Studio 的 Dialog Editor 中，对话方块模板需要一些我们不能添加的东西。因此，对话方块模板包含在 HEXCALC.DLG 档案中，而且需要手工输入。依照下面的方法，您可以将一个文字档案添加到任何专案中：从「File」功能表选择「New」，再选择「Files」页面标签，然後从档案型态列表中选择「Text File」。像这样的档案——包含附加资源定义——需要包含在资源描述中。从「View」功能表选择「Resource Includes」。这显示一个对话方块。在「Compile-time Directives」编辑栏输入

```
#include "hexcalc.dlg"
```

这一行将插入到 HEXCALC.RC 资源描述中，像上面所显示的一样。

仔细看一下 HEXCALC.DLG 档案中的对话方块模板，您将发现 HEXCALC 如何为对话方块使用它自己的视窗讯息处理程式。对话方块模板的上方如下：

```
HexCalc DIALOG -1, -1, 102, 122
STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS "HexCalc"
CAPTION "Hex Calculator"
```

注意诸如 WS_OVERLAPPED 和 WS_MINIMIZEBOX 等识别字，我们可以将它们用在 CreateWindow 呼叫中以建立普通的视窗。CLASS 叙述是这个对话方块与曾经建立过的对话方块之间最重要的区别（而且它也是 Developer Studio 中的 Dialog Editor 不允许我们指定的）。当对话方块模板省略了这个叙述时，Windows 为对话方块注册一个视窗类别，并使用它自己的视窗讯息处理程式处理对话方块讯息。这里，包含 CLASS 叙述就告诉 Windows 将讯息发送到其他地方——具体的说，就是发送到在 HexCalc 视窗类别中指定的视窗讯息处理程式。

HexCalc 视窗类别是在 HEXCALC 的 WinMain 函式中注册的，就像普通视窗的视窗类别一样。但是，请注意有个十分重要的区别：WNDCLASS 结构的 cbWndExtra 栏位设定为 DLGWINDOWEXTRA。对於您自己注册的对话方块程序，这是必需的。

在注册视窗类别之後，WinMain 呼叫 CreateDialog：

```
hwnd = CreateDialog (hInstance, szAppName, 0, NULL) ;
```

第二个参数（字串「HexCaEc」）是对话方块模板的名字。第三个参数通常是父视窗的视窗代号，这里设定为 0，因为视窗没有父视窗。最後一个参数，通常是对话方块程序的位址，这里不需要。因为 Windows 不会处理这些讯息，因

而也不会将讯息发送给对话方块程序。

这个 CreateDialog 呼叫与对话方块模板一起，被 Windows 有效地转换为一个 CreateWindow 呼叫。该 CreateWindow 呼叫的功能与下面的呼叫相同：

```
hwnd = CreateWindow (TEXT ("HexCalc"), TEXT ("Hex Calculator"),
                    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    102 * 4 / cxChar, 122 * 8 / cyChar,
                    NULL, NULL, hInstance, NULL) ;
```

其中，cxChar 和 cyChar 变数分别是系统字体字元的宽度和高度。

我们通过让 Windows 来进行 CreateWindow 呼叫而收获甚丰：Windows 不会在建立弹出式视窗 1 后就停止，它还会为对话方块模板中定义的其他 29 个子视窗按键控制项呼叫 CreateWindow。所有这些控制项都给父视窗的视窗讯息处理程式发送 WM_COMMAND 讯息，该程式正是 WndProc。对于建立一个包含许多子视窗的视窗来说，这是一个很好的技巧。

下面是使 HEXCALC 的程式码量下降到最少的另一种方法：或许您会注意到 HEXCALC 没有表头档案，表头档案中通常包含对话方块模板中，需要为所有子视窗控制项定义的识别字。我们之所以可以不要这个档案，是因为每个按键控制项的 ID 设定为出现在控制项上的文字的 ASCII 码。这意味著，WndProc 可以完全相同地对待 WM_COMMAND 讯息和 WM_CHAR 讯息。在每种情况下，wParam 的低字组都是按钮的 ASCII 码。

当然，对键盘讯息进行一些处理是必要的。WndProc 拦截 WM_KEYDOWN 讯息，将左箭头键转换为 Backspace 键。在处理 WM_CHAR 讯息时，WndProc 将字元代码转换为大写，Enter 键转换为等号键的 ASCII 码。

WM_CHAR 讯息的有效性是通过呼叫 GetDlgItem 来检验的。如果 GetDlgItem 函式传回 0，那么键盘字元不是对话方块模板中定义的 ID 之一。如果字元是 ID 之一，则通过给相应的按钮发送一对 BM_SETSTATE 讯息，来使之闪烁：

```
if (hButton = GetDlgItem (hwnd, wParam))
{
    SendMessage (hButton, BM_SETSTATE, 1, 0) ;
    Sleep (100) ;
    SendMessage (hButton, BM_SETSTATE, 0, 0) ;
}
```

这样做，用最小的代价，却为 HEXCALC 的键盘介面增色不少。Sleep 函式将程式暂停 100 毫秒。这会防止按钮被按得太快而让人注意不到。

当 WndProc 处理 WM_COMMAND 讯息时，它总是将输入焦点设定给父视窗：

```
case WM_COMMAND :
    SetFocus (hwnd) ;
```

否则，一旦使用滑鼠单击某按钮，输入焦点就会切换到该按钮上。

通用对话方块

Windows 的一个主要目的是推动标准的使用者介面。对许多常用的功能表项来说，这推行得很快，几乎所有软体厂商都采用 Alt-File-Open 选择来打开一个档案。然而，实际的档案开启对话方块却经常各不相同。

从 Windows 3.1 开始，对这个问题有了一个可行的解决方案，这是一种叫做「通用对话方块程式库」的增强。这个程式库由几个函式组成，这些函式启动标准对话方块来进行打开和储存档案、搜索和替换、选择颜色、选择字体（我将在本章讨论以上的这些内容）以及列印（我将在第十三章讨论）。

为了使用这些函式，您基本上都要初始化某一结构的各个栏位，并将该结构的指标传送给通用对话方块程式库的某个函式，该函式会建立并显示对话方块。当使用者关闭对话方块时，被呼叫的函式将控制权传回给程式，您可以从传送给它的结构中获得资讯。

在使用通用对话方块程式库的任何 C 原始码档案时，您都需要含入 COMMDLG.H 表头档案。通用对话方块的文件在/Platform SDK/User Interface Services/User Input/Common Dialog Box Library 中。

增强 POPPAD

当我们往第十章的 POPPAD 中增加功能表时，还有几个标准功能表项没有实作。现在我们已经准备好在 POPPAD 中加入打开档案、读入档案以及在磁片上储存编辑过档案的功能。在处理中，我们还将在 POPPAD 中加入字体选择和搜索替换功能。

实作 POPPAD3 程式的档案如程式 11-6 所示。

程式 11-6 POPPAD3

```
POPPAD.C
/*-----
POPPAD.C -- Popup Editor
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"

#define EDITID 1
#define UNTITLED TEXT ("(untitled)")

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;
```

```

        // Functions in POPFILE.C

void PopFileInitialize          (HWND) ;
BOOL PopFileOpenDlg            (HWND, PTSTR, PTSTR) ;
BOOL PopFileSaveDlg            (HWND, PTSTR, PTSTR) ;
BOOL PopFileRead                (HWND, PTSTR) ;
BOOL PopFileWrite               (HWND, PTSTR) ;

        // Functions in POPFIND.C

HWND PopFindFindDlg            (HWND) ;
HWND PopFindReplaceDlg         (HWND) ;
BOOL PopFindFindText           (HWND, int *, LPFINDREPLACE) ;
BOOL PopFindReplaceText        (HWND, int *, LPFINDREPLACE) ;
BOOL PopFindNextText           (HWND, int *) ;
BOOL PopFindValidFind          (void) ;

        // Functions in POPFONT.C

void PopFontInitialize          (HWND) ;
BOOL PopFontChooseFont         (HWND) ;
void PopFontSetFont             (HWND) ;
void PopFontDeinitialize (void) ;

        // Functions in POPPRNT.C

BOOL PopPrntPrintFile (HINSTANCE, HWND, HWND, PTSTR) ;

        // Global variables

static HWND hDlgModeless ;
static TCHAR szAppName[] = TEXT ("PopPad") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    MSG      msg ;
    HWND      hwnd ;
    HACCEL    hAccel ;
    WNDCLASS  wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon      (hInstance,
szAppName) ;

```

```

        wndclass.hCursor                = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground          = (HBRUSH)      GetStockObject
(WHITE_BRUSH) ;
        wndclass.lpszMenuName           = szAppName ;
        wndclass.lpszClassName          = szAppName ;

        if (!RegisterClass (&wndclass))
        {
                MessageBox (      NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName, MB_ICONERROR) ;

                return 0 ;
        }

        hwnd = CreateWindow (  szAppName, NULL,
                                WS_OVERLAPPEDWINDOW,
                                CW_USEDEFAULT, CW_USEDEFAULT,
                                CW_USEDEFAULT, CW_USEDEFAULT,
                                NULL, NULL, hInstance, szCmdLine) ;

        ShowWindow (hwnd, iCmdShow) ;
        UpdateWindow (hwnd) ;
        hAccel = LoadAccelerators (hInstance, szAppName) ;

        while (GetMessage (&msg, NULL, 0, 0))
        {
                if (hDlgModeless == NULL || !IsDialogMessage (hDlgModeless,
&msg))
                {
                        if (!TranslateAccelerator (hwnd, hAccel, &msg))
                        {
                                TranslateMessage (&msg) ;
                                DispatchMessage (&msg) ;
                        }
                }
        }
        return msg.wParam ;
}

void DoCaption (HWND hwnd, TCHAR * szTitleName)
{
        TCHAR szCaption[64 + MAX_PATH] ;
        wsprintf (szCaption, TEXT ("%s - %s"), szAppName,
                                szTitleName[0] ? szTitleName : UNTITLED) ;
        SetWindowText (hwnd, szCaption) ;
}

void OkMessage (HWND hwnd, TCHAR * szMessage, TCHAR * szTitleName)

```

```

{
    TCHAR szBuffer[64 + MAX_PATH] ;
    wsprintf (szBuffer, szMessage, szTitleName[0] ? szTitleName : UNTITLED) ;
    MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

short AskAboutSave (HWND hwnd, TCHAR * szTitleName)
{
    TCHAR          szBuffer[64 + MAX_PATH] ;
    int    iReturn ;

    wsprintf (szBuffer, TEXT ("Save current changes in %s?"),
              szTitleName[0] ? szTitleName : UNTITLED) ;

    iReturn = MessageBox (hwnd, szBuffer, szAppName,
                          MB_YESNOCANCEL | MB_ICONQUESTION) ;
    if (iReturn == IDYES)
        if (!SendMessage (hwnd, WM_COMMAND, IDM_FILE_SAVE, 0))
            iReturn = IDCANCEL ;

    return iReturn ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static BOOL          bNeedSave = FALSE ;
    static HINSTANCE hInst ;
    static HWND          hwndEdit ;
    static int           iOffset ;
    static TCHAR          szFileName[MAX_PATH],
szTitleName[MAX_PATH] ;
    static UINT          messageFindReplace ;
    int                  iSelBeg, iSelEnd, iEnable ;
    LPFINDREPLACE        pfr ;

    switch (message)
    {
    case WM_CREATE:
        hInst = ((LPCREATESTRUCT) lParam) -> hInstance ;
                // Create the edit control child window
        hwndEdit = CreateWindow (TEXT ("edit"), NULL,
                                WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
                                WS_BORDER | ES_LEFT | ES_MULTILINE |
                                ES_NOHIDESEL | ES_AUTOHSCROLL | ES_AUTOVSCROLL,
                                0, 0, 0, 0,
                                hwnd, (HMENU) EDITID, hInst, NULL) ;

```

```

        SendMessage (hwndEdit, EM_LIMITTEXT, 32000, 0L) ;
        // Initialize common dialog box stuff
        PopFileInitialize (hwnd) ;
        PopFontInitialize (hwndEdit) ;

        messageFindReplace = RegisterWindowMessage (FINDMSGSTRING) ;
        DoCaption (hwnd, szTitleName) ;
        return 0 ;
    case WM_SETFOCUS:
        SetFocus (hwndEdit) ;
        return 0 ;

    case WM_SIZE:
        MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam),
TRUE) ;

        return 0 ;

    case WM_INITMENUPOPUP:
        switch (lParam)
        {
            case 1: // Edit menu

                // Enable Undo if edit control can do
it
                EnableMenuItem ((HMENU) wParam,
IDM_EDIT_UNDO,
                SendMessage (hwndEdit, EM_CANUNDO, 0, 0L) ?
                MF_ENABLED : MF_GRAYED) ;

                // Enable Paste if text is in the
clipboard
                EnableMenuItem ((HMENU) wParam,
IDM_EDIT_PASTE,
                IsClipboardFormatAvailable (CF_TEXT) ?
                MF_ENABLED : MF_GRAYED) ;

                // Enable Cut, Copy, and Del if text is selected

                SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iSelBeg,
                (LPARAM) &iSelEnd) ;

                iEnable = iSelBeg != iSelEnd ? MF_ENABLED : MF_GRAYED ;

                EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT,
iEnable) ;

                EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY,

```



```

iEnable) ;
                                EnableMenuItem    ((HMENU)    wParam,    IDM_EDIT_CLEAR,
iEnable) ;
                                break ;

                                case 2:                                // Search menu

                                // Enable Find, Next, and Replace if modeless
                                // dialogs are not already active

                                iEnable = hDlgModeless == NULL ?
                                MF_ENABLED : MF_GRAYED ;
                                EnableMenuItem ((HMENU) wParam, IDM_SEARCH_FIND,
                                iEnable) ;
                                EnableMenuItem    ((HMENU)    wParam,
IDM_SEARCH_NEXT,    iEnable) ;
                                EnableMenuItem    ((HMENU)    wParam,
IDM_SEARCH_REPLACE, iEnable) ;
                                break ;
                                }
                                return 0 ;

                                case WM_COMMAND:

                                // Messages from edit control

                                if (lParam && LOWORD (wParam) == EDITID)
                                {
                                    switch (HIWORD (wParam))
                                    {
                                        case EN_UPDATE :
                                            bNeedSave = TRUE ;
                                            return 0 ;

                                        case EN_ERRSPACE :
                                            case EN_MAXTEXT :
                                                MessageBox (hwnd, TEXT ("Edit control out of space."),
                                                szAppName, MB_OK | MB_ICONSTOP) ;
                                                return 0 ;
                                    }
                                    break ;
                                }

                                switch (LOWORD (wParam))
                                {
                                    // Messages from File menu
                                    case IDM_FILE_NEW:
                                        if (bNeedSave && IDCANCEL == AskAboutSave
(hwnd, szTitleName))

                                        return 0 ;

```

```
        SetWindowText (hwndEdit, TEXT ("\0")) ;
        szFileName[0] = '\0' ;
        szTitleName[0] = '\0' ;
        DoCaption (hwnd, szTitleName) ;
        bNeedSave = FALSE ;
        return 0 ;

        case IDM_FILE_OPEN:
if (bNeedSave && IDCANCEL == AskAboutSave (hwnd, szTitleName))
    return 0 ;
if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
    {
        if (!PopFileRead (hwndEdit, szFileName))
            {
                OkMessage (hwnd, TEXT ("Could not read file %s!"),
                    szTitleName) ;
                szFileName[0] = '\0' ;
                szTitleName[0] = '\0' ;
            }
        DoCaption (hwnd, szTitleName) ;
        bNeedSave = FALSE ;
        return 0 ;

case IDM_FILE_SAVE:
    if (szFileName[0])
    {
        if (PopFileWrite (hwndEdit, szFileName))
        {
            bNeedSave = FALSE ;
            return 1 ;
        }
        else
        {
            OkMessage (hwnd, TEXT ("Could not write file %s"),
                szTitleName) ;
            return 0 ;
        }
    }
    //fall through
case IDM_FILE_SAVE_AS:
    if (PopFileSaveDlg (hwnd, szFileName, szTitleName))
    {
        DoCaption (hwnd, szTitleName) ;

        if (PopFileWrite (hwndEdit,
```

```

szFileName))

        {
                                bNeedSave    =
FALSE ;                                return 1 ;

        }
        else
        {
                OkMessage (hwnd, TEXT ("Could not write file %s"),
                        szTitleName) ;
                                return 0 ;
        }
        }
        return 0 ;

    case  IDM_FILE_PRINT:
        if  (!PopPrntPrintFile  (hInst,  hwnd,  hwndEdit,
szTitleName))
            OkMessage ( hwnd, TEXT ("Could not print file %s"),
                        szTitleName) ;
            return 0 ;

    case  IDM_APP_EXIT:
        SendMessage (hwnd, WM_CLOSE, 0, 0) ;
        return 0 ;

                                // Messages from Edit menu

    case  IDM_EDIT_UNDO:
        SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
        return 0 ;

    case  IDM_EDIT_CUT:
        SendMessage (hwndEdit, WM_CUT, 0, 0) ;
        return 0 ;

    case  IDM_EDIT_COPY:
        SendMessage (hwndEdit, WM_COPY, 0, 0) ;
        return 0 ;

    case  IDM_EDIT_PASTE:
        SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
        return 0 ;

    case  IDM_EDIT_CLEAR:
        SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
        return 0 ;

```

```
case IDM_EDIT_SELECT_ALL:
    SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
    return 0 ;

// Messages from Search menu
case IDM_SEARCH_FIND:
    SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;
    hDlgModeless = PopFindFindDlg (hwnd) ;
    return 0 ;

case IDM_SEARCH_NEXT:
    SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;

    if (PopFindValidFind ())
        PopFindNextText (hwndEdit, &iOffset) ;
    else
        hDlgModeless = PopFindFindDlg (hwnd) ;

    return 0 ;

case IDM_SEARCH_REPLACE:
    SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;
    hDlgModeless = PopFindReplaceDlg (hwnd) ;
    return 0 ;

case IDM_FORMAT_FONT:
    if (PopFontChooseFont (hwnd))
        PopFontSetFont (hwndEdit) ;

    return 0 ;

// Messages from Help menu
case IDM_HELP:
    OkMessage (hwnd, TEXT ("Help not yet implemented!"),
        TEXT ("\0")) ;
    return 0 ;

case IDM_APP_ABOUT:
    DialogBox (hInst, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
    return 0 ;
}

break ;
case WM_CLOSE:
    if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
        DestroyWindow (hwnd) ;

    return 0 ;
```

```

case WM_QUERYENDSESSION :
    if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
        return 1 ;

    return 0 ;

case WM_DESTROY:
    PopFontDeinitialize () ;
    PostQuitMessage (0) ;
    return 0 ;

default:
    // Process "Find-Replace" messages
    if (message == messageFindReplace)
    {
        pfr = (LPFINDREPLACE) lParam ;
        if (pfr->Flags & FR_DIALOGTERM)
            hDlgModeless = NULL ;

        if (pfr->Flags & FR_FINDNEXT)
            if (!PopFindFindText (hwndEdit, &iOffset, pfr))
                OkMessage (hwnd, TEXT ("Text not found!"),
                    TEXT ("\0")) ;

        if (pfr->Flags & FR_REPLACE || pfr->Flags &
FR_REPLACEALL)
            if (!PopFindReplaceText (hwndEdit, &iOffset,
pfr))
                OkMessage (hwnd, TEXT ("Text not found!"),
                    TEXT ("\0")) ;

        if (pfr->Flags & FR_REPLACEALL)
            while (PopFindReplaceText (hwndEdit,
&iOffset, pfr)) ;

        return 0 ;
    }
    break ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            return TRUE ;
    }

```

```

        case WM_COMMAND:
            switch (LOWORD (wParam))
            {
                case IDOK:
                    EndDialog (hDlg, 0) ;
                    return TRUE ;

            }
            break ;
    }
    return FALSE ;
}

POPFIL.C
/*-----
    POPFILE.C -- Popup Editor File Functions
    -----*/

#include <windows.h>
#include <commdlg.h>

static OPENFILENAME ofn ;
void PopFileInitialize (HWND hwnd)
{
    static TCHAR szFilter[] = TEXT ("Text Files (*.TXT)\0*.txt\0") \
                                TEXT ("ASCII Files (*.ASC)\0*.asc\0") \
                                TEXT ("All Files (*.*)\0*.*\0\0") ;

    ofn.lStructSize          = sizeof (OPENFILENAME) ;
    ofn.hwndOwner             = hwnd ;
    ofn.hInstance             = NULL ;
    ofn.lpstrFilter           = szFilter ;
    ofn.lpstrCustomFilter     = NULL ;
    ofn.nMaxCustFilter        = 0 ;
    ofn.nFilterIndex          = 0 ;
    ofn.lpstrFile             = NULL ;           // Set in Open and Close functions
    ofn.nMaxFile              = MAX_PATH ;
    ofn.lpstrFileName         = NULL ;           // Set in Open and Close
functions
    ofn.nMaxFileName         = MAX_PATH ;
    ofn.lpstrInitialDir       = NULL ;
    ofn.lpstrTitle            = NULL ;
    ofn.Flags                  = 0 ;               // Set in Open and
Close functions
    ofn.nFileOffset           = 0 ;
    ofn.nFileExtension        = 0 ;
    ofn.lpstrDefExt           = TEXT ("txt") ;
    ofn.lCustData              = 0L ;
    ofn.lpfnHook              = NULL ;
}

```

```

        ofn.lpTemplateName          = NULL ;
    }

BOOL PopFileOpenDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
    ofn.hwndOwner                    = hwnd ;
    ofn.lpstrFile                    = pstrFileName ;
    ofn.lpstrFileTitle               = pstrTitleName ;
    ofn.Flags                        = OFN_HIDEREADONLY | OFN_CREATEPROMPT ;

    return GetOpenFileName (&ofn) ;
}

BOOL PopFileSaveDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
    ofn.hwndOwner                    = hwnd ;
    ofn.lpstrFile                    = pstrFileName ;
    ofn.lpstrFileTitle               = pstrTitleName ;
    ofn.Flags                        = OFN_OVERWRITEPROMPT ;

    return GetSaveFileName (&ofn) ;
}

BOOL PopFileRead (HWND hwndEdit, PTSTR pstrFileName)
{
    BYTE                            bySwap ;
    DWORD                           dwBytesRead ;
    HANDLE                           hFile ;
    int                              i, iFileLength, iUniTest ;
    PBYTE                            pBuffer, pText, pConv ;

    // Open the file.
    if (INVALID_HANDLE_VALUE ==
        (hFile = CreateFile (pstrFileName,  GENERIC_READ,
FILE_SHARE_READ,
        NULL, OPEN_EXISTING, 0, NULL)))
        return FALSE ;
    // Get file size in bytes and allocate memory for read.
    // Add an extra two bytes for zero termination.

    iFileLength = GetFileSize (hFile, NULL) ;
    pBuffer = malloc (iFileLength + 2) ;

    // Read file and put terminating zeros at end.
    ReadFile (hFile, pBuffer, iFileLength, &dwBytesRead, NULL) ;
    CloseHandle (hFile) ;
    pBuffer[iFileLength] = '\0' ;
    pBuffer[iFileLength + 1] = '\0' ;

```

```

        // Test to see if the text is Unicode
iUniTest = IS_TEXT_UNICODE_SIGNATURE | IS_TEXT_UNICODE_REVERSE_SIGNATURE ;
if (IsTextUnicode (pBuffer, iFileLength, &iUniTest))
{
    pText = pBuffer + 2 ;
    iFileLength -= 2 ;

    if (iUniTest & IS_TEXT_UNICODE_REVERSE_SIGNATURE)
    {
        for (i = 0 ; i < iFileLength / 2 ; i++)
        {
            bySwap = ((BYTE *) pText) [2 * i] ;
            ((BYTE *) pText) [2 * i] = ((BYTE *) pText) [2 * i + 1] ;
            ((BYTE *) pText) [2 * i + 1] = bySwap ;
        }
    }

    // Allocate memory for possibly converted string
pConv = malloc (iFileLength + 2) ;
    // If the edit control is not Unicode, convert Unicode
text to
    // non-Unicode (i.e., in general, wide character).
#ifdef UNICODE
    WideCharToMultiByte (CP_ACP, 0, (PWSTR) pText, -1, pConv,
        iFileLength + 2, NULL, NULL) ;
    // If the edit control is Unicode, just copy the
string
#else
    lstrcpy ((PTSTR) pConv, (PTSTR) pText) ;
#endif

}
else // the file is not Unicode
{
    pText = pBuffer ;
    // Allocate memory for possibly converted string.
pConv = malloc (2 * iFileLength + 2) ;
    // If the edit control is Unicode, convert ASCII
text.
#ifdef UNICODE
    MultiByteToWideChar (CP_ACP, 0, pText, -1, (PTSTR) pConv,
        iFileLength + 1) ;
    // If not, just copy buffer
#else
    lstrcpy ((PTSTR) pConv, (PTSTR) pText) ;
#endif
}

```



```

    SetWindowText (hwndEdit, (PTSTR) pConv) ;
    free (pBuffer) ;
    free (pConv) ;

    return TRUE ;
}

BOOL PopFileWrite (HWND hwndEdit, PTSTR pstrFileName)
{
    DWORD          dwBytesWritten ;
    HANDLE          hFile ;
    int             iLength ;
    PTSTR           pstrBuffer ;
    WORD            wByteOrderMark = 0xFEFF ;
                  // Open the file, creating it if necessary

    if (INVALID_HANDLE_VALUE ==
        (hFile = CreateFile (pstrFileName, GENERIC_WRITE, 0,
        NULL, CREATE_ALWAYS, 0, NULL)))
        return FALSE ;
        // Get the number of characters in the edit control and allocate
        // memory for them.

    iLength = GetWindowTextLength (hwndEdit) ;
    pstrBuffer = (PTSTR) malloc ((iLength + 1) * sizeof (TCHAR)) ;

    if (!pstrBuffer)
    {
        CloseHandle (hFile) ;
        return FALSE ;
    }

    // If the edit control will return Unicode text, write the
    // byte order mark to the file.

#ifdef UNICODE
    WriteFile (hFile, &wByteOrderMark, 2, &dwBytesWritten, NULL) ;
#endif

    // Get the edit buffer and write that out to the file.
    GetWindowText (hwndEdit, pstrBuffer, iLength + 1) ;
    WriteFile (hFile, pstrBuffer, iLength * sizeof (TCHAR),
               &dwBytesWritten, NULL) ;
    if ((iLength * sizeof (TCHAR)) != (int) dwBytesWritten)
    {
        CloseHandle (hFile) ;
        free (pstrBuffer) ;
        return FALSE ;
    }
}

```

```

    }

    CloseHandle (hFile) ;
    free (pstrBuffer) ;

    return TRUE ;
}

POPFIND.C
/*-----
   POPFIND.C -- Popup Editor Search and Replace Functions
   -----*/

#include <windows.h>
#include <commdlg.h>
#include <tchar.h>           // for _tcsstr (strstr for Unicode &
                             // non-Unicode)

#define MAX_STRING_LEN    256

static TCHAR szFindText [MAX_STRING_LEN] ;
static TCHAR szReplText [MAX_STRING_LEN] ;

HWND PopFindFindDlg (HWND hwnd)
{
    static FINDREPLACE fr ;      // must be static for modeless dialog!!!

    fr.lStructSize      = sizeof (FINDREPLACE) ;
    fr.hwndOwner        = hwnd ;
    fr.hInstance        = NULL ;
    fr.Flags            = FR_HIDEUPDOWN | FR_HIDEMATCHCASE |
FR_HIDEWHOLEWORD ;
    fr.lpstrFindWhat    = szFindText ;
    fr.lpstrReplaceWith = NULL ;
    fr.wFindWhatLen     = MAX_STRING_LEN ;
    fr.wReplaceWithLen  = 0 ;
    fr.lCustData        = 0 ;
    fr.lpfHook          = NULL ;
    fr.lpTemplateName  = NULL ;

    return FindText (&fr) ;
}

HWND PopFindReplaceDlg (HWND hwnd)
{
    static FINDREPLACE fr ;      // must be static for modeless dialog!!!

    fr.lStructSize      = sizeof (FINDREPLACE) ;
    fr.hwndOwner        = hwnd ;

```

```

        fr.hInstance          = NULL ;
        fr.Flags              =   FR_HIDEUPDOWN    |   FR_HIDEMATCHCASE    |
FR_HIDEWHOLEWORD ;
        fr.lpstrFindWhat      = szFindText ;
        fr.lpstrReplaceWith   = szReplText ;
        fr.wFindWhatLen       = MAX_STRING_LEN ;
        fr.wReplaceWithLen    = MAX_STRING_LEN ;
        fr.lCustData          = 0 ;
        fr.lpfHook            = NULL ;
        fr.lpTemplateName    = NULL ;

        return ReplaceText (&fr) ;
}

BOOL PopFindFindText (HWND hwndEdit, int * piSearchOffset, LPFINDREPLACE pfr)
{
    int    iLength, iPos ;
    PTSTR  pstrDoc, pstrPos ;

        // Read in the edit document

    iLength = GetWindowTextLength (hwndEdit) ;

    if (NULL == (pstrDoc = (PTSTR) malloc ((iLength + 1) * sizeof (TCHAR))))
        return FALSE ;

    GetWindowText (hwndEdit, pstrDoc, iLength + 1) ;

        // Search the document for the find string

    pstrPos = _tcsstr (pstrDoc + * piSearchOffset, pfr->lpstrFindWhat) ;
    free (pstrDoc) ;

        // Return an error code if the string cannot be found

    if (pstrPos == NULL)
        return FALSE ;

        // Find the position in the document and the new start
offset

    iPos = pstrPos - pstrDoc ;
    * piSearchOffset = iPos + lstrlen (pfr->lpstrFindWhat) ;

        // Select the found text
    SendMessage (hwndEdit, EM_SETSEL, iPos, * piSearchOffset) ;
    SendMessage (hwndEdit, EM_SCROLLCARET, 0, 0) ;

```

```

        return TRUE ;
    }
}
BOOL PopFindNextText (HWND hwndEdit, int * piSearchOffset)
{
    FINDREPLACE fr ;
    fr.lpstrFindWhat = szFindText ;
    return PopFindFindText (hwndEdit, piSearchOffset, &fr) ;
}

BOOL PopFindReplaceText (HWND hwndEdit, int * piSearchOffset, LPFIND, REPLACE pfr)
{
    // Find the text
    if (!PopFindFindText (hwndEdit, piSearchOffset, pfr))
        return FALSE ;

    // Replace it
    SendMessage (      hwndEdit, EM_REPLACESEL, 0, (LPARAM) pfr->
lpstrReplaceWith) ;
    return TRUE ;
}

BOOL PopFindValidFind (void)
{
    return * szFindText != '\0' ;
}

POPFONT.C
/*-----
POPFONT.C -- Popup Editor Font Functions
-----*/

#include <windows.h>
#include <commdlg.h>

static LOGFONT logfont ;
static HFONT  hFont ;

BOOL PopFontChooseFont (HWND hwnd)
{
    CHOOSEFONT cf ;
    cf.lStructSize      = sizeof (CHOOSEFONT) ;
    cf.hwndOwner        = hwnd ;
    cf.hDC              = NULL ;
    cf.lpLogFont        = &logfont ;
    cf.iPointSize       = 0 ;
    cf.Flags            = CF_INITTLOGFONTSTRUCT | CF_SCREENFONTS |
CF_EFFECTS ;
    cf.rgbColors        = 0 ;
    cf.lCustData        = 0 ;

```

```

        cf.lpfHook          = NULL ;
        cf.lpTemplateName   = NULL ;
        cf.hInstance        = NULL ;
        cf.lpszStyle        = NULL ;
        cf.nFontType        = 0 ;           //      Returned
from ChooseFont
        cf.nSizeMin         = 0 ;
        cf.nSizeMax         = 0 ;

        return ChooseFont (&cf) ;
}

void PopFontInitialize (HWND hwndEdit)
{
    GetObject (GetStockObject (SYSTEM_FONT), sizeof (LOGFONT),
               (PTSTR) &logfont) ;
    hFont = CreateFontIndirect (&logfont) ;
    SendMessage (hwndEdit, WM_SETFONT, (WPARAM) hFont, 0) ;
}

void PopFontSetFont (HWND hwndEdit)
{
    HFONT hFontNew ;
    RECT rect ;

    hFontNew = CreateFontIndirect (&logfont) ;
    SendMessage (hwndEdit, WM_SETFONT, (WPARAM) hFontNew, 0) ;
    DeleteObject (hFont) ;
    hFont = hFontNew ;
    GetClientRect (hwndEdit, &rect) ;
    InvalidateRect (hwndEdit, &rect, TRUE) ;
}

void PopFontDeinitialize (void)
{
    DeleteObject (hFont) ;
}

POPPRNT0.C
/*-----
   POPPRNT0.C -- Popup Editor Printing Functions (dummy version)
   -----*/

#include <windows.h>
BOOL PopPrntPrintFile (      HINSTANCE hInst, HWND hwnd, HWND hwndEdit,
                             PTSTR pstrTitleName)
{
    return FALSE ;
}

```

POPPAD.RC (摘录)

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK",IDOK,66,80,50,14
    ICON
    "POPPAD",IDC_STATIC,7,7,20,20
    CTEXT
    "PopPad",IDC_STATIC,40,12,100,8
    CTEXT        "Popup Editor for Windows",IDC_STATIC,7,40,166,8
    CTEXT        "(c) Charles Petzold, 1998",IDC_STATIC,7,52,166,8
END
PRINTDLGBOX DIALOG DISCARDABLE 32, 32, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "PopPad"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON    "Cancel",IDCANCEL,67,74,50,14
    CTEXT          "Sending",IDC_STATIC,8,8,172,8
    CTEXT          "",IDC_FILENAME,8,28,172,8
    CTEXT          "to print spooler.",IDC_STATIC,8,48,172,8
END

////////////////////////////////////
/
// Menu
POPPAD MENU DISCARDABLE
BEGIN
    POPUP          "&File"
    BEGIN
        MENUITEM    "&New\tCtrl+N",    IDM_FILE_NEW
        MENUITEM    "&Open...\tCtrl+O",IDM_FILE_OPEN
        MENUITEM    "&Save\tCtrl+S",    IDM_FILE_SAVE
        MENUITEM    "Save &As...",    IDM_FILE_SAVE_AS
        MENUITEM    SEPARATOR
        MENUITEM    "&Print\tCtrl+P",    IDM_FILE_PRINT
        MENUITEM    SEPARATOR
        MENUITEM    "E&xit",            IDM_APP_EXIT
    END
    POPUP "&Edit"
```

```

BEGIN
    MENUITEM    "&Undo\tCtrl+Z",    IDM_EDIT_UNDO
    MENUITEM    SEPARATOR
    MENUITEM    "Cu&t\tCtrl+X",    IDM_EDIT_CUT
    MENUITEM    "&Copy\tCtrl+C",    IDM_EDIT_COPY
    MENUITEM    "&Paste\tCtrl+V",    IDM_EDIT_PASTE
    MENUITEM    "De&lete\tDel",    IDM_EDIT_CLEAR
    MENUITEM    SEPARATOR
    MENUITEM    "&Select All",    IDM_EDIT_SELECT_ALL
END

    POPUP      "&Search"
BEGIN
    MENUITEM    "&Find...\tCtrl+F",IDM_SEARCH_FIND
    MENUITEM    "Find &Next\tF3",  IDM_SEARCH_NEXT
    MENUITEM    "&Replace...\tCtrl+R", IDM_SEARCH_REPLACE
END

    POPUP      "F&ormat"
BEGIN
    MENUITEM    "&Font...",
END

    POPUP      "&Help"
    BEGIN
    MENUITEM    "&Help",            IDM_HELP
    MENUITEM    "&About PopPad...", IDM_APP_ABOUT
    END
END

////////////////////////////////////
/
// Accelerator
POPPAD ACCELERATORS DISCARDABLE
BEGIN
    VK_BACK,    IDM_EDIT_UNDO,    VIRTKEY,    ALT,    NOINVERT
    VK_DELETE,  IDM_EDIT_CLEAR,    VIRTKEY,    NOINVERT
    VK_DELETE,  IDM_EDIT_CUT,    VIRTKEY,    SHIFT,    NOINVERT
    VK_F1,      IDM_HELP,        VIRTKEY,    NOINVERT
    VK_F3,      IDM_SEARCH_NEXT, VIRTKEY,    NOINVERT
    VK_INSERT,  IDM_EDIT_COPY,    VIRTKEY,    CONTROL, NOINVERT
    VK_INSERT,  IDM_EDIT_PASTE,    VIRTKEY,    SHIFT,
NOINVERT
    "^C",      IDM_EDIT_COPY,      ASCII,      NOINVERT
    "^F",      IDM_SEARCH_FIND,     ASCII,      NOINVERT
    "^N",      IDM_FILE_NEW,        ASCII,      NOINVERT
    "^O",      IDM_FILE_OPEN,       ASCII,      NOINVERT
    "^P",      IDM_FILE_PRINT,      ASCII,      NOINVERT
    "^R",      IDM_SEARCH_REPLACE,  ASCII,      NOINVERT
    "^S",      IDM_FILE_SAVE,       ASCII,      NOINVERT
    "^V",      IDM_EDIT_PASTE,      ASCII,      NOINVERT
    "^X",      IDM_EDIT_CUT,        ASCII,      NOINVERT

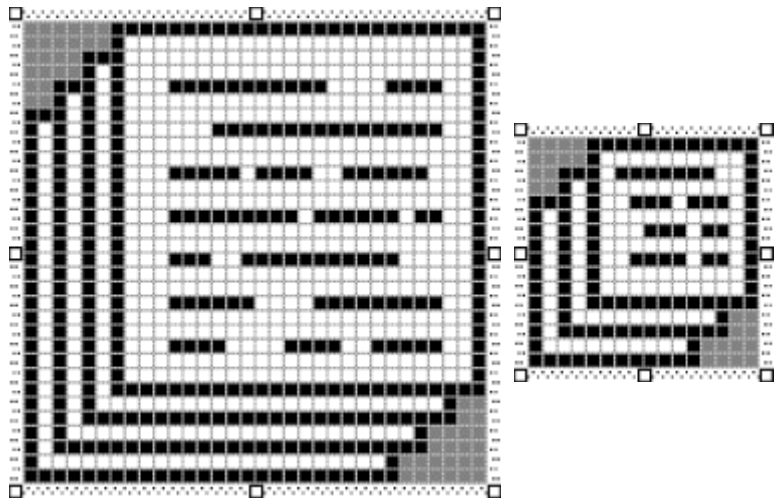
```

```
        "^Z",          IDM_EDIT_UNDO,          ASCII,          NOINVERT
END

////////////////////////////////////
/
// Icon
POPPAD          ICON          DISCARDABLE
"poppad.ico"
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by poppad.rc

#define IDC_FILENAME          1000
#define IDM_FILE_NEW          40001
#define IDM_FILE_OPEN          40002
#define IDM_FILE_SAVE          40003
#define IDM_FILE_SAVE_AS          40004
#define IDM_FILE_PRINT          40005
#define IDM_APP_EXIT          40006
#define IDM_EDIT_UNDO          40007
#define IDM_EDIT_CUT          40008
#define IDM_EDIT_COPY          40009
#define IDM_EDIT_PASTE          40010
#define IDM_EDIT_CLEAR          40011
#define IDM_EDIT_SELECT_ALL          40012
#define IDM_SEARCH_FIND          40013
#define IDM_SEARCH_NEXT          40014
#define IDM_SEARCH_REPLACE          40015
#define IDM_FORMAT_FONT          40016
#define IDM_HELP          40017
#define IDM_APP_ABOUT          40018
```

POPPAD. ICO



为了避免在第十三章中重复原始码，我在 POPPAD.RC 的功能表中加入了列印专案和一些其他的支援。

POPPAD.C 包含了程式中所有的基本原始码。POPPFILE.C 具有启动 File Open 和 File Save 对话方块的程式码, 它还包含档案 I/O 常式。POPPFIND.C 中包含了搜寻和替换文字功能。POPPFONT.C 包含了字体选择功能。POPPRNT0.C 不完成什么工作: 在第十三章中将使用 POPPRNT.C 替换 POPPRNT0.C 以建立最终的 POPPAD 程式。

让我们先来看一看 POPPAD.C。POPPAD.C 含有两个档案名字串: 第一个, 储存在 WndProc, 名称为 szFileName, 含有详细的驱动器名称、路径名称和档案名称; 第二个, 储存为 szTitleName, 是程式本身的档案名称。它用在 POPPAD3 的 DoCaption 函式中, 以便将档案名称显示在视窗的标题列上; 也用在 OKMessage 函式和 AskAboutSave 函式中, 以便向使用者显示讯息方块。

POPPFILE.C 包含了几个显示「File Open」和「File Save」对话方块以及实际执行档案 I/O 的函式。对话方块是使用函式 GetOpenFileName 和 GetSaveFileName 来显示的。这两个函式都使用一个型态为 OPENFILENAME 的结构, 这个结构在 COMMdlg.H 中定义。在 POPPFILE.C 中, 使用了一个该结构型态的整体变数, 取名为 ofn。ofn 的大多数栏位在 PopFileInitialize 函式中被初始化, POPPAD.C 在 WndProc 中处理 WM_CREATE 讯息时呼叫该函式。

将 ofn 作为静态整体结构变数会比较方便, 因为 GetOpenFileName 和 GetSaveFileName 给该结构传回的一些资讯, 并将在以后呼叫这些函式时用到。

尽管通用对话方块具有许多选项——包括设定自己的对话方块模板, 以及为对话方块程序增加「挂勾 (hook)」——POPPFILE.C 中使用的「File Open」和「File Save」对话方块是最基本的。OPENFILENAME 结构中被设定的栏位只有 lStructSize (结构的长度)、hwndOwner (对话方块拥有者)、lpstrFilter (下面将简要讨论)、lpstrFile 和 nMaxFile (指向接收完整档案名称的缓冲区指标和该缓冲区的大小)、lpstrFileName 和 nMaxFileName (档案名称缓冲区及其大小)、Flags (设定对话方块的选项) 和 lpstrDefExt (如果使用者在对话方块中输入档案名时不指定档案副档名, 那么它就是内定的档案副档名)。

当使用者在「File」功能表中选择「Open」时, POPPAD3 呼叫 POPPFILE 的 PopFileOpenDlg 函式, 将视窗代号、一个指向档案名称缓冲区的指标和一个指向档案标题缓冲区的指标传给它。PopFileOpenDlg 恰当地设定 OPENFILENAME 结构的 hwndOwner、lpstrFile 和 lpstrFileName 栏位, 将 Flags 设定为 OFN_CREATEPROMPT, 然后呼叫 GetOpenFileName, 显示如图 11-6 所示的普通对话方块。

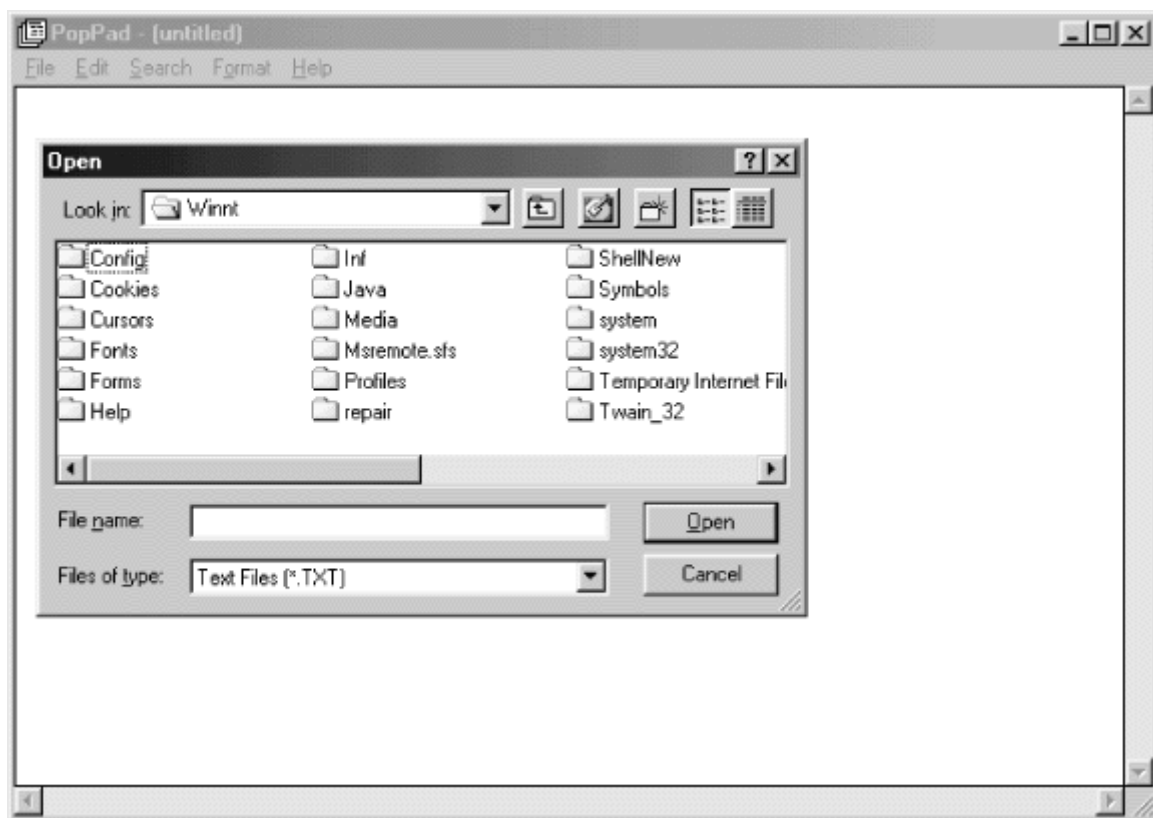


图 11-6 「File Open」对话方块

当使用者结束这个对话方块时，`GetOpenFileName` 函式传回。`OFN_CREATEPROMPT` 旗标指示 `GetOpenFileName` 显示一个讯息方块，询问使用者如果所选档案不存在，是否要建立该档案。

左下角的下拉式清单方块列出了将要显示在档案列表中的档案型态，此清单方块被称为「筛选清单」。使用者可以通过从下拉式清单方块列表中选择另一种档案型态，来改变筛选条件。在 `POPFIL.C` 的 `PopFileInitialize` 函式中，我在变数 `szFilter`（一个字串阵列）中为三种型态的档案定义了一个筛检清单：带有 `.TXT` 副档名的文字档案、带有 `.ASC` 副档名的 `ASCII` 档案和所有档案。`OPENFILENAME` 结构的 `lpstrFilter` 栏位储存指向此阵列第一个字串的指标。

如果使用者在对话方块处於活动状态时改变了筛选条件，那么 `OPENFILENAME` 的 `nFilterIndex` 栏位反映出使用者的选择。由於该结构是静态变数，下次启动对话方块时，筛选条件将被设定为选中的档案型态。

`POPFIL.C` 中的 `PopFileSaveDlg` 函式与此类似，它将 `Flags` 参数设定为 `OFN_OVERWRITEPROMPT`，并呼叫 `GetSaveFileName` 启动「File Save」对话方块。`OFN_OVERWRITEPROMPT` 旗标导致显示一个讯息方块，如果被选档案已经存在，那么将询问使用者是否覆盖该档案。

Unicode 档案 I/O

對於本书中的大多数程式，您都不必注意 Unicode 和非 Unicode 版的区别。

例如,在 POPPAD3 的 Unicode 中,编辑控制项将保留 Unicode 文字和使用 Unicode 字串的所有通用对话方块。例如,当程式需要搜索和替换时,所有的操作都会处理 Unicode 字串,而不需要转换。

不过,POPPAD3 得处理档案 I/O,也就是说,程式不能闭门造车。如果 Unicode 版的 POPPAD3 获得了编辑缓冲区的内容并将其写入磁片,档案将是使用 Unicode 存放的。如果非 Unicode 版的 POPPAD3 读取了该档案,并将其写入编辑缓冲区,其结果将是一堆垃圾。Unicode 版读取由非 Unicode 版储存的档案时也会这样。

解决的办法在於辨别和转换。首先,在 POPFILE.C 的 PopFileWrite 函式中,您将看到 Unicode 版的程式将在档案的开始位置写入 0xFEFF。这定义为位元组顺序标记,以表示文字档案含有 Unicode 文字。

其次,在 PopFileRead 函式中,程式用 IsTextUnicode 函式来决定档案是否含有位元组顺序标记。此函式甚至检测位元组顺序标记是否反向了,亦即 Unicode 文字档案在 Macintosh 或者其他使用与 Intel 处理器相反的位元组顺序的机器上建立的。这时,位元组的顺序都经过·转。如果档案是 Unicode 版,但是被非 Unicode 版的 POPPAD3 读取,这时,文字将被 WideCharToMultiChar 转换。WideCharToMultiChar 实际上是一个宽字元 ANSI 函式(除非您执行远东版的 Windows)。只有这时文字才能放入编辑缓冲区。

同样地,如果档案是非 Unicode 文字档案,而执行的是 Unicode 版的程式,那么文字必须用 MultiCharToWideChar 转换。

改变字体

我们将在第十七章详细讨论字体,但那些都不能代替通用对话方块函式来选择字体。

在 WM_CREATE 讯息处理期间,POPFONT.C 中的 POPPAD 呼叫 PopFontInitialize。这个函式取得一个依据系统字体建立的 LOGFONT 结构,由此建立一种字体,并向编辑控制项发送一个 WM_SETFONT 讯息来设定一种新的字体(内定编辑控制项字体是系统字体,而 PopFontInitialize 为编辑控制项建立一种新的字体,因为最终该字体将被删除,而删除现有系统字体是不明智的)。

当 POPPAD 收到来自程式的字体选项的 WM_COMMAND 讯息时,它呼叫 PopFontChooseFont。这个函式初始化一个 CHOOSEFONT 结构,然後呼叫 ChooseFont 显示字体选择对话方块。如果使用者按下「OK」按钮,那么 ChooseFont 将传回 TRUE。随後,POPPAD 呼叫 PopFontSetFont 来设定编辑控制项中的新字体,旧字体将被删除。

最後,在 WM_DESTROY 讯息处理期间,POPPAD 呼叫 PopFontDeinitialize 来

删除最近一次由 PopFontSetFont 建立的字体。

搜寻与替换

通用对话方块程式库也提供两个用於文字搜寻和替换函式的对话方块，这两个函式 (FindText 和 ReplaceText) 使用一个型态为 FINDREPLACE 的结构。图 10-11 中所示的 POPFIND.C 档案有两个常式 (PopFindFindDlg 和 PopFindReplaceDlg) 呼叫这些函式，还有两个函式在编辑控制项中搜寻和替换文字。

使用搜寻和替换函式有一些考虑。首先，它们启动的对话方块是非模态对话方块，这意味著必须改写讯息回圈，以便在对话方块活动时呼叫 IsDialogMessage。第二，传送给 FindText 和 ReplaceText 的 FINDREPLACE 结构必须是一个静态变数，因为对话方块是模态的，函式在对话方块显示之後传回，而不是在对话方块结束之後传回；而对话方块程序必须仍然能够存取该结构。

第三，在显示 FindText 和 ReplaceText 对话方块时，它们通过一条特殊讯息与拥有者视窗联络，讯息编号可以通过以 FINDMSGSTRING 为参数呼叫 RegisterWindowMessage 函式来获得。这是在 WndProc 中处理 WM_CREATE 讯息时完成的，讯息号存放在静态变数中。

在处理内定讯息时，WndProc 将讯息变数与 RegisterWindowMessage 传回的值相比较。lParam 讯息参数是一个指向 FINDREPLACE 结构的指标，Flags 栏位指示使用者使用对话方块是为了搜寻文字还是替换文字，以及是否要终止对话方块。POPPAD3 是呼叫 POPFIND.C 中的 PopFindFindText 和 PopFindReplaceText 函式来执行搜寻和替换功能的。

只呼叫一个函式的 Windows 程式

到现在为止，我们已经说明了两个程式，让您浏览选择颜色，这两个程式分别是第九章中的 COLORS1 和本章中的 COLORS2。现在是讲解 COLORS3 的时候了，这个程式只有一个 Windows 函式呼叫。COLORS3 的原始码如程式 11-7 所示。

COLORS3 所呼叫的唯一 Windows 函式是 ChooseColor，这也是通用对话方块程式库中的函式，它显示如图 11-7 所示的对话方块。颜色选择类似於 COLORS1 和 COLORS2，但是它与使用者交谈互动能力更强。

程式 11-7 COLORS3

```
COLORS3.C
/*-----
COLORS3.C -- Version using Common Dialog Box
```

(c) Charles Petzold, 1998

```

-----*/

#include <windows.h>
#include <commdlg.h>

int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int iCmdShow)
{
    static CHOOSECOLOR    cc ;
    static COLORREF        crCustColors[16] ;

    cc.lStructSize          = sizeof (CHOOSECOLOR) ;
    cc.hwndOwner             = NULL ;
    cc.hInstance            = NULL ;
    cc.rgbResult            = RGB (0x80, 0x80, 0x80) ;
    cc.lpCustColors         = crCustColors ;
    cc.Flags                = CC_RGBINIT | CC_FULLOPEN ;
    cc.lCustData             = 0 ;
    cc.lpfHook              = NULL ;
    cc.lpTemplateName = NULL ;

    return ChooseColor (&cc) ;
}

```

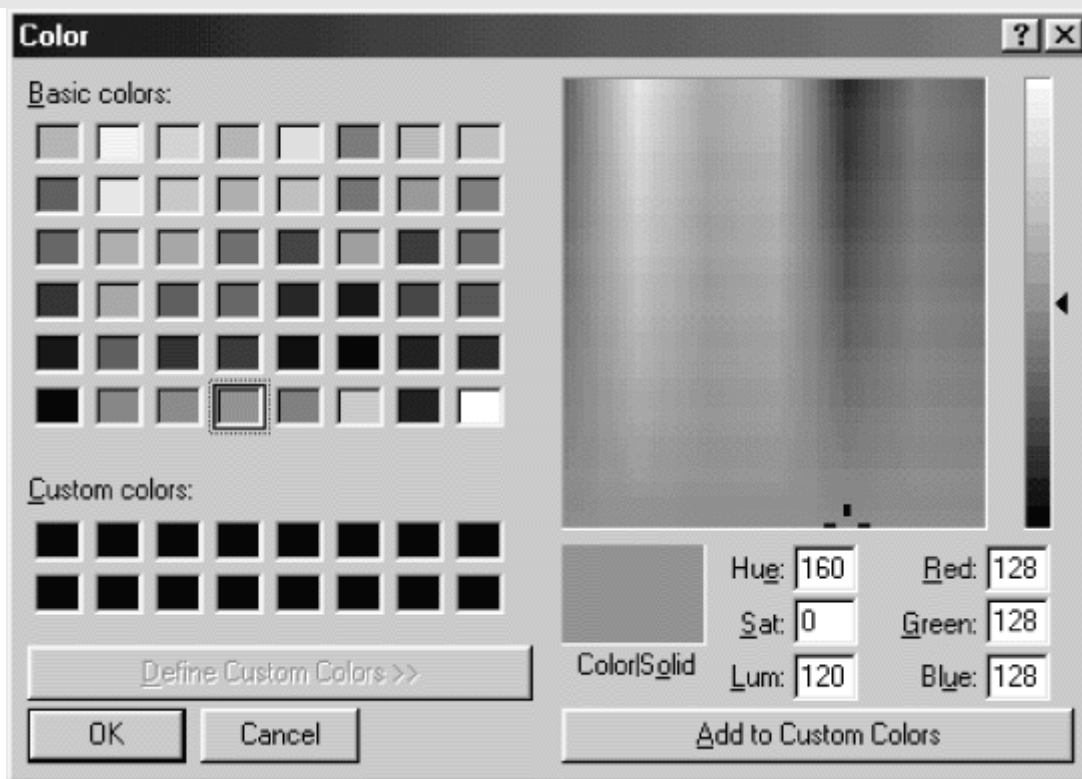


图 11-7 COLORS3 的萤幕显示

ChooseColor 函式使用一个 CHOOSECOLOR 型态的结构和含有 16 个 DWORD 的阵列来存放常用颜色，使用者将从对话方块中选择这些颜色之一。rgbResult 栏

位可以初始化为一个颜色值，如果 Flags 栏位的 CC_RGBINIT 旗标被设立，则显示该颜色。通常在使用这个函式时，rgbResult 将被设定为使用者选择的颜色。

请注意，Color 对话方块的 hWndOwner 栏位被设定为 NULL。在 ChooseColor 函式呼叫 DialogBox 以显示对话方块时，DialogBox 的第三个参数也被设定为 NULL。这是完全合法的，其含义是对话方块不为另一个视窗所拥有。对话方块的标题将显示在工作列中，而对话方块就像一个普通的视窗那样执行。

您也可以在自己程式的对话方块中使用这种技巧。使 Windows 程式只建立对话方块，其他事情都在对话方块程序中完成，这是可能的。

第十二章 剪贴簿

Microsoft Windows 剪贴簿允许把资料从一个程式传送到另一个程式中。它的原理相对而言比较简单，把资料存放到剪贴簿上的程式或从剪贴簿上取出资料的程式都无须太多的负担。Windows 98 和 Microsoft Windows NT 都提供了剪贴簿浏览程式，该程式可以显示剪贴簿的目前内容。

许多处理档案或者其他资料的程式都包含一个「Edit」功能表，其中包括「Cut」、「Copy」和「Paste」选项。当使用者选择「Cut」或者「Copy」时，程式将资料传送给剪贴簿。这个资料使用某种格式，如文字、点阵图（一种按位元排列的矩形阵列，其中的位元与平面显示的图素相对应）或者 metafile（用二进位元数值内容表示的绘图命令集）等。当使用者从功能表中选择「Paste」时，程式检查剪贴簿中包含的资料，看看使用的是否是程式可以接受的一种格式。如果是，那么资料将从剪贴簿传送到程式中。

如果使用者不发出明确的指令，程式就不能把资料送入或移出剪贴簿。例如，在某个程式中执行剪下或复制（或者按 Ctrl-X 及 Ctrl-C）操作的使用者，应该能够假定资料将储存在剪贴簿上，直到下次剪下或复制操作为止。

回忆一下第十和第十一章所示的 POPPAD 程式的修订版中，我们加上了「Edit」功能表，但是在那边这功能表的作用只是发送讯息给编辑控制项而已。多数情况下，处理剪贴簿并不方便，您必须自己呼叫剪贴簿传输函式。

本章集中讨论将文字传入和移出剪贴簿。在后面的章节里，我将向您展示如何用剪贴簿处理点阵图（第十四、十五和十六章）和 metafile（第十八章）。

剪贴簿的简单使用

我们由分析把资料传送到剪贴簿（剪下或复制）和存取剪贴簿资料（粘贴）的程式码开始。

标准剪贴簿资料格式

Windows 支援不同的预先定义剪贴簿格式，这些格式在 WINUSER.H 定义成以 CF 为字首的识别字。

首先介绍三种能够储存在剪贴簿上的文字资料型态，以及一个与剪贴簿格式相关的资料型态：

CF_TEXT 以 NULL 结尾的 ANSI 字元集字串。它在每行末尾包含一个 carriage return 和 linefeed 字元，这是最简单的剪贴簿资料格式。传送到剪

贴簿的资料存放在整体记忆体块中，并且是利用记忆体块代号进行传送的（我将简短地讨论此项概念）。这个记忆体块专供剪贴簿使用，建立它的程式不应该继续使用它。

CF_OEMTEXT 含有文字资料（与 CF_TEXT 类似）的记忆体块。但是它使用的是 OEM 字元集。通常 Windows 程式不必关心这一点；它只有与在视窗中执行 MS-DOS 程式一起使用剪贴簿时才会使用。

CF_UNICODETEXT 含有 Unicode 文字的记忆体块。与 CF_TEXT 类似，它在每一行的末尾包含一个 carriage return 和 linefeed 字元，以及一个 NULL 字元（两个 0 位元组）以表示资料结束。CF_UNICODETEXT 只支援 Windows NT。

CF_LOCALE 一个国家地区识别字的代号。表示剪贴簿文字使用的国别地区设定。

下面是两种附加的剪贴簿格式，它们在概念上与 CF_TEXT 格式相似（也就是说，它们都是文字资料），但是它们不需要以 NULL 结尾，因为格式已经定义了资料的结尾。现在已经很少使用这些格式了：

CF_SYLK 包含 Microsoft「符号连结」资料格式的整体记忆体块。这种格式用在 Microsoft 的 Multiplan、Chart 和 Excel 程式之间交换资料，它是一种 ASCII 码格式，每一行都用 carriage return 和 linefeed 结尾。

CF_DIF 包含资料交换格式 (DIF) 之资料的整体记忆体块。这种格式是由 Software Arts 公司提出的，用於把资料送到 VisiCalc 试算表程式中。这也是一种 ASCII 码格式，每一行都使用 carriage return 和 linefeed 结尾。

下面三种剪贴簿格式与点阵图有关。所谓点阵图就是资料位元的矩形阵列，其中的资料位元与输出设备的图素相对应。第十四和第十五章将详细讨论点阵图以及这些点阵图剪贴簿的格式：

CF_BITMAP 与装置相关的点阵图格式。点阵图是通过点阵图代号传送给剪贴簿的。同样，在把这个点阵图传送给剪贴簿之後，程式不应该再继续使用这个点阵图。

CF_DIB 定义一个装置无关点阵图（在第十五章中描述）的记忆体块。这种记忆体块是以点阵图资讯结构开始的，後面跟著可用的颜色表和点阵图资料位元。

CF_PALETTE 调色盘代号。它通常与 CF_DIB 配合使用，以定义与装置相关的点阵图所使用的颜色调色盘。

在剪贴簿中，还有可能以工业标准的 TIFF 格式储存的点阵图资料：

CF_TIFF 含有标号图像档案格式 (TIFF) 资料的整体记忆体块。这种格式由 Microsoft、Aldus 公司和 Hewlett-Packard 公司以及一些硬体厂商推荐使用。

这一格式可从 Hewlett-Packard 的网站上获得。

下面是两个 metafile 格式，我将在第十八章详细讨论。一个 metafile 就是一个以二进位格式储存的画图命令集：

CF_METAFILEPICT 以旧的 metafile 格式存放的「图片」。

CF_ENHMETAFILE 增强型 metafile (32 位元 Windows 支援的) 代号。

最後介绍几个混合型的剪贴簿格式：

CF_PENDATA 与 Windows 的笔式输入扩充功能联合使用。

CF_WAVE 声音 (波形) 档案。

CF_RIFF 使用资源交换档案格式 (Resource Interchange File Format) 的多媒体资料。

CF_HDROP 与拖放服务相关的档案列表。

记忆体配置

程式向剪贴簿传输一些资料的时候，必须配置一个记忆体块，并且将这块记忆体交给剪贴簿处理。在本书早期的程式中需要配置记忆体时，我们只需使用标准 C 执行时期程式库所支援的 malloc 函式。但是，由於在 Windows 中执行的应用程式之间必须要共用剪贴簿所储存的记忆体块，这时 malloc 函式就有些不适任这项任务了。

实际上，我们必须把早期 Windows 所开发的记忆体配置函式再拿出来使用，那时的作业系统在 16 位元的实际模式记忆体结构中执行。现在的 Windows 仍然支援这些函式，您还可以使用它们，但不是必须使用这些函式就是了。

要用 Windows API 来配置一个记忆体块，可以呼叫：

```
hGlobal = GlobalAlloc (uiFlags, dwSize) ;
```

此函式有两个参数：一系列可能的旗标和记忆体块的位元组大小。函式传回一个 HGLOBAL 型态的代号，称为「整体记忆体块代号」或「整体代号」。传回值为 NULL 表示不能配置足够的记忆体。

虽然 GlobalAlloc 的两个参数略有不同，但它们都是 32 位元的无正负号整数。如果将第一个参数设定为 0，那么您就可以更有效地使用旗标 GMEM_FIXED。在这种情况下，GlobalAlloc 传回的整体代号实际是指向所配置记忆体块的指标。

如果不喜欢将记忆体块中的每一位元都初始化为 0，那么您也能够使用旗标 GMEM_ZEROINIT。在 Windows 表头档案中，简洁的 GPTR 旗标定义为 GMEM_FIXED 和 GMEM_ZEROINIT 旗标的组合：

```
#define GPTR (GMEM_FIXED | GMEM_ZEROINIT)
```

下面是一个重新配置函式：

```
hGlobal = GlobalReAlloc (hGlobal, dwSize, uiFlags) ;
```

如果记忆体块扩大了，您可以用 GMEM_ZEROINIT 旗标将新的位元组设为 0。

下面是获得记忆体块大小的函式：

```
dwSize = GlobalSize (hGlobal) ;
```

释放记忆体块的函式：

```
GlobalFree (hGlobal) ;
```

在早期 16 位元的 Windows 中，因为 Windows 不能在实体记忆体中移动记忆体块，所以禁止使用 GMEM_FIXED 旗标。在 32 位元的 Windows 中，GMEM_FIXED 旗标很常见。这是因为它将传回一个虚拟位址，并且作业系统也能够通过改变记忆体页映射表在实体记忆体中移动记忆体块。因此为 16 位元的 Windows 写程式时，GlobalAlloc 推荐使用 GMEM_MOVEABLE 旗标。在 Windows 的表头档案中还定义了一个简写识别字，用此识别字可以在可移动的记忆体之外填 0：

```
#define GHND (GMEM_MOVEABLE | GMEM_ZEROINIT)
```

GMEM_MOVEABLE 旗标允许 Windows 在虚拟记忆体中移动一个记忆体块。这不是说将在实体记忆体中移动记忆体块，只是应用程式用於读写这块记忆体的位址可以被变动。

尽管 GMEM_MOVEABLE 是 16 位元 Windows 的通则，但是它的作用现在已经少得多了。如果您的应用程式频繁地配置、重新配置以及释放不同大小的记忆体块，应用程式的虚拟位址空间将会变得支离破碎。可以想像得到，最後虚拟记忆体位址空间就会被用完。如果这是个可能会发生的问题，那么您将希望记忆体是可移动的。下面就介绍如何让记忆体块成为可搬移位置的。

首先定义一个指标（例如，一个 int 型态的）和一个 GLOBALHANDLE 型态的变数：

```
int * p ;  
  
GLOBALHANDLE hGlobal ;
```

然後配置记忆体。例如：

```
hGlobal = GlobalAlloc (GHND, 1024) ;
```

与处理其他 Windows 代号一样，您不必担心数字的实际意义，只要照著作就好了。需要存取记忆体块时，可以呼叫：

```
p = (int *) GlobalLock (hGlobal) ;
```

此函式将代号转换为指标。在记忆体块被锁定期间，Windows 将固定虚拟记忆体中的位址，不再移动那块记忆体。存取结束後呼叫：

```
GlobalUnlock (hGlobal) ;
```

这将使 Windows 可以在虚拟记忆体中移动记忆体块。要真正确保此程序正常运作（体验早期 Windows 程式写作者的痛苦经历），您应该在单一个讯息处理期间锁定和解锁记忆体块。

在释放记忆体时，呼叫 GlobalFree 应使用代号而不是指标。如果您现在不

能存取代号，可以使用下面的函式：

```
hGlobal = GlobalHandle (p) ;
```

在解锁之前，您能够多次锁定一个记忆体块。Windows 保留一个锁定次数，而且在记忆体块可被自由移动之前，每次锁定都需要相对应的解锁。当 Windows 在虚拟记忆体中移动一个记忆体块时，不需要将位元组从一个位置复制到另一个，只需巧妙地处理记忆体页映射表。通常，让 32 位元 Windows 为您的程式配置可移动的记忆体块，其唯一确实的理由只是避免虚拟记忆体的空间碎裂出现。使用剪贴簿时，也应该使用可移动记忆体。

为剪贴簿配置记忆体时，您应该以 GMEM_MOVEABLE 和 GMEM_SHARE 旗标呼叫 GlobalAlloc 函式。GMEM_SHARE 旗标使得其他應用程式也可以使用那块记忆体。

将文字传送到剪贴簿

让我们想像把一个 ANSI 字串传送到剪贴簿上，并且我们已经有了指向这个字串的指标(pString)。现在希望传送这个字串的 iLength 字元，这些字元可能以 NULL 结尾，也可能不以 NULL 结尾。

首先，通过使用 GlobalAlloc 来配置一个足以储存字串的记忆体块，其中还包括一个终止字元 NULL：

```
hGlobal = GlobalAlloc (GHND | GMEM_SHARE, iLength + 1) ;
```

如果未能配置到记忆体块，hGlobal 的值将为 NULL。如果配置成功，则锁定这块记忆体，并得到指向它的一个指标：

```
pGlobal = GlobalLock (hGlobal) ;
```

将字串复制到记忆体块中：

```
for (i = 0 ; i < wLength ; i++)  
    *pGlobal++ = *pString++ ;
```

由於 GlobalAlloc 的 GHND 旗标已使整个记忆体块在配置期间被清除为零，所以不需要增加结尾的 NULL。以下叙述为记忆体块解锁：

```
GlobalUnlock (hGlobal) ;
```

现在就有了表示以 NULL 结尾的文字所在记忆体块的记忆体代号。为了把它送到剪贴簿中，打开剪贴簿并把它清空：

```
OpenClipboard (hwnd) ;  
EmptyClipboard () ;
```

利用 CF_TEXT 识别字把记忆体代号交给剪贴簿，关闭剪贴簿：

```
SetClipboardData (CF_TEXT, hGlobal) ;  
CloseClipboard () ;
```

工作告一段落。

下面是关于此过程的一些规则：

在处理同一个讯息的过程中呼叫 OpenClipboard 和 CloseClipboard。不需

要时，不要打开剪贴簿。

不要把锁定的记忆体代号交给剪贴簿。

当呼叫 `SetClipboardData` 後，请不要再继续使用该记忆体块。它不再属於使用者程式，必须把代号看成是无效的。如果需要继续存取资料，可以制作资料的副本，或从剪贴簿中读取它（如下节所述）。您也可以在 `SetClipboardData` 呼叫和 `CloseClipboard` 呼叫之间继续使用记忆体块，但是不要使用传递给 `SetClipboardData` 函式的整体代号。事实上，此函式也传回一个整体代号，必需锁定这些代码以存取记忆体。在呼叫 `CloseClipboard` 之前，应先为此代号解锁。

从剪贴簿上取得文字

从剪贴簿上取得文字只比把文字传送到剪贴簿上稍微复杂一些。您必须首先确定剪贴簿是否含有 `CF_TEXT` 格式的资料，最简单的方法是呼叫

```
bAvailable = IsClipboardFormatAvailable (CF_TEXT) ;
```

如果剪贴簿上含有 `CF_TEXT` 资料，这个函式将传回 `TRUE`（非零）。我们在第十章的 `POPPAD2` 程式中已使用了这个函式，用它来确定「Edit」功能表中「Paste」项是被启用还是被停用的。`IsClipboardFormatAvailable` 是少数几个不需先打开剪贴簿就可以使用的剪贴簿函式之一。但是，如果您之後想再打开剪贴簿以取得这个文字，就应该再做一次检查（使用同样的函式或其他方法），以便确定 `CF_TEXT` 资料是否仍然留在剪贴簿中。

为了传送出文字，首先打开剪贴簿：

```
OpenClipboard (hwnd) ;
```

会得到代表文字的记忆体块代号：

```
hGlobal = GetClipboardData (CF_TEXT) ;
```

如果剪贴簿不包含 `CF_TEXT` 格式的资料，此代号就为 `NULL`。这是确定剪贴簿是否含有文字的另一种方法。如果 `GetClipboardData` 传回 `NULL`，则关闭剪贴簿，不做其他任何工作。

从 `GetClipboardData` 得到的代号并不属於使用者程式——它属於剪贴簿。仅在 `GetClipboardData` 和 `CloseClipboard` 呼叫之间这个代号才有效。您不能释放这个代号或更改它所引用的资料。如果需要继续存取这些资料，必须制作这个记忆体块的副本。

这里有一种将资料复制到使用者程式中的方法。首先，配置一块与剪贴簿资料块大小相同的记忆体块，并配置一个指向该块的指标：

```
pText = (char *) malloc (GlobalSize (hGlobal)) ;
```

再次呼叫 `hGlobal`，而 `hGlobal` 是从 `GetClipboardData` 呼叫传回的整体代

号。现在锁定代号，获得一个指向剪贴簿块的指标：

```
pGlobal = GlobalLock (hGlobal) ;
```

现在就可以复制资料了：

```
strcpy (pText, pGlobal) ;
```

或者，您可以使用一些简单的 C 程式码：

```
while (*pText++ = *pGlobal++) ;
```

在关闭剪贴簿之前先解锁记忆体块：

```
GlobalUnlock (hGlobal) ;
```

```
CloseClipboard () ;
```

现在您有了一个叫做 pText 的指标，以後程式的使用者就可以用它来复制文字了。

打开和关闭剪贴簿

在任何时候，只有一个程式可以打开剪贴簿。呼叫 OpenClipboard 的作用是当一个程式使用剪贴簿时，防止剪贴簿的内容发生变化。OpenClipboard 传回 BOOL 值，它说明是否已经成功地打开了剪贴簿。如果另一个应用程式没有关闭剪贴簿，那么它就不能被打开。如果每个程式在回应使用者的命令时都尽快地、遵守规范地打开然後关闭剪贴簿，那么您将永远不会遇到不能打开剪贴簿的问题。

但是，在不遵守规范程式和优先权式多工环境中，总会发生一些问题。即使在您的程式将某些东西放入剪贴簿和使用者的启动一个「Paste」选项期间，您的程式并没有失去输入焦点，但是您也不能假定您放入的东西仍然在那里，一个背景程式有可能已经在这段期间存取过剪贴簿了。

而且，请留意一个与讯息方块有关的更微妙问题：如果不能配置足够的记忆体来将内容复制到剪贴簿，那么您可能希望显示一个讯息方块。但是，如果这个讯息方块不是系统模态的，那么使用者可以在显示讯息方块期间切换到另一个应用程式中。您应该使用系统模态的讯息方块，或者在您显示讯息方块之前关闭剪贴簿。

如果您在显示一个对话方块时将剪贴簿保持为打开状态，那么您可能遇到其他问题，对话方块中的编辑栏位会使用剪贴簿进行文字的剪贴。

剪贴簿和 Unicode

迄今为止，我只讨论了用剪贴簿处理 ANSI 文字(每个字元对应一个位元组)。我们用 CF_TEXT 识别字时就是这种格式。您可能对 CF_OEMTEXT 和 CF_UNICODETEXT 还不熟悉吧。

我有一些好消息：在处理您所想要的文字格式时，您只需呼叫 SetClipboardData 和 GetClipboardData，Windows 将处理剪贴簿中所有的文字转换。例如，在 Windows NT 中，如果一个程式用 SetClipboardData 来处理 CF_TEXT 剪贴簿资料型态，程式也能用 CF_OEMTEXT 呼叫 GetClipboardData。同样地，剪贴簿也能将 CF_OEMTEXT 资料转换为 CF_TEXT。

在 Windows NT 中，转换发生在 CF_UNICODETEXT、CF_TEXT 和 CF_OEMTEXT 之间。程式应该使用对程式本身而言最方便的一种文字格式来呼叫 SetClipboardData。同样地，程式应该用程式需要的文字格式来呼叫 GetClipboardData。我们已经知道，本书附上的程式在编写时可以带有或不带 UNICODE 识别字。如果您的程式也依此编写，那么在定义了 UNICODE 识别字之後，程式将执行带有 CF_UNICODETEXT 参数的 SetClipboardData 以及 GetClipboardData 呼叫，而不是 CF_TEXT。

CLIPTEXT 程式，如程式 12-1 所示，展示了一种可行的方法。

程式 12-1 CLIPTEXT

```
CLIPTEXT.C
/*-----
    CLIPTEXT.C --          The Clipboard and Text
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
#ifdef UNICODE
#define CF_TCHAR CF_UNICODETEXT
TCHAR szDefaultText[]      = TEXT ("Default Text - Unicode Version") ;
TCHAR szCaption[]          = TEXT ("Clipboard Text Transfers - Unicode
Version") ;
#else
#define CF_TCHAR CF_TEXT
TCHAR szDefaultText[] = TEXT ("Default Text - ANSI Version") ;
TCHAR szCaption[]      = TEXT ("Clipboard Text Transfers - ANSI Version") ;
#endif
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("ClipText") ;
    HACCEL            hAccel ;
    HWND              hwnd ;
    MSG                msg ;
    WNDCLASS           wndclass ;
```

```

    wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc          = WndProc ;
    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance           = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName         = szAppName ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, szCaption,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    hAccel = LoadAccelerators (hInstance, szAppName) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static PTSTR                pText ;
    BOOL                        bEnable ;
    HGLOBAL                     hGlobal ;

```

```

HDC          hdc ;
PTSTR        pGlobal ;
PAINTSTRUCT  ps ;
RECT         rect ;

switch (message)
{
case WM_CREATE:
    SendMessage (hwnd, WM_COMMAND, IDM_EDIT_RESET, 0) ;
    return 0 ;

case WM_INITMENUPOPUP:
    EnableMenuItem ((HMENU) wParam,
IDM_EDIT_PASTE,
        IsClipboardFormatAvailable (CF_TCHAR) ?
MF_ENABLED : MF_GRAYED) ;

    bEnable = pText ? MF_ENABLED : MF_GRAYED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT,
bEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY,
bEnable) ;
    EnableMenuItem ((HMENU) wParam,
IDM_EDIT_CLEAR, bEnable) ;
    break ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
case IDM_EDIT_PASTE:
        OpenClipboard (hwnd) ;

        if (hGlobal = GetClipboardData
(CF_TCHAR))
        {
            pGlobal = GlobalLock (hGlobal) ;
            if (pText)
            {
                free (pText) ;
                pText = NULL ;
            }
            pText = malloc (GlobalSize (hGlobal)) ;
            lstrcpy (pText, pGlobal) ;
            InvalidateRect (hwnd, NULL, TRUE) ;
        }
        CloseClipboard () ;
        return 0 ;

```



```

case IDM_EDIT_CUT:
case IDM_EDIT_COPY:
    if (!pText)
        return 0 ;

    hGlobal = GlobalAlloc (GHND | GMEM_SHARE,
        (lstrlen (pText) + 1) * sizeof (TCHAR)) ;
    pGlobal = GlobalLock (hGlobal) ;
    lstrcpy (pGlobal, pText) ;
    GlobalUnlock (hGlobal) ;

    OpenClipboard (hwnd) ;
    EmptyClipboard () ;
    SetClipboardData (CF_TCHAR, hGlobal) ;
    CloseClipboard () ;

    if ( LOWORD (wParam) == IDM_EDIT_COPY)
        return 0 ;
    // fall through for IDM_EDIT_CUT
case IDM_EDIT_CLEAR:
    if (pText)
    {
        free (pText) ;
        pText = NULL ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case IDM_EDIT_RESET:
    if (pText)
    {
        free (pText) ;
        pText = NULL ;
    }
    pText = malloc ((lstrlen (szDefaultText) + 1) * sizeof
(TCHAR)) ;

    lstrcpy (pText, szDefaultText) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
}
break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

```

```

        if (pText != NULL)
            DrawText (hdc, pText, -1, &rect, DT_EXPANDTABS |
DT_WORDBREAK) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        if ( pText)
            free (pText) ;

            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

CLIPTEXT.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

CLIPTEXT MENU DISCARDABLE

BEGIN

POPUP "&Edit"

BEGIN

MENUITEM "Cu&t\tCtrl+X", IDM_EDIT_CUT

MENUITEM "&Copy\tCtrl+C", IDM_EDIT_COPY

MENUITEM "&Paste\tCtrl+V", IDM_EDIT_PASTE

MENUITEM "De&lete\tDel", IDM_EDIT_CLEAR

MENUITEM SEPARATOR

MENUITEM "&Reset", IDM_EDIT_RESET

END

END

////////////////////////////////////
/

// Accelerator

CLIPTEXT ACCELERATORS DISCARDABLE

BEGIN

"C", IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT

"V", IDM_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT

VK_DELETE, IDM_EDIT_CLEAR, VIRTKEY, NOINVERT

"X", IDM_EDIT_CUT, VIRTKEY, CONTROL, NOINVERT

END

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
// Used by ClipText.rc

#define IDM_EDIT_CUT            40001
#define IDM_EDIT_COPY          40002
#define IDM_EDIT_PASTE         40003
#define IDM_EDIT_CLEAR         40004
#define IDM_EDIT_RESET         40005
```

这是在 Windows NT 下执行 Unicode 版和 ANSI 版程式的概念，而且可以看到，剪贴簿是如何在两种字元集之间转换的。注意 CLIPTEXT.C 顶部的 `#ifdef` 叙述。如果定义了 UNICODE 识别字，那么 `CF_TCHAR`（我命名的一种常用的剪贴簿格式）就等於 `CF_UNICODETEXT`；否则，它就等於 `CF_TEXT`。程式後面呼叫的 `IsClipboardFormatAvailable`、`GetClipboardData` 和 `SetClipboardData` 函式都使用 `CF_TCHAR` 来指定资料型态。

在程式的开始部分（以及您从「Edit」功能表中选择「Reset」选项时），静态变数 `pText` 包含一个指标，在 Unicode 版的程式中，指标指向 Unicode 字串「Default Text - Unicode version」；在非 Unicode 版的程式中，指标指向「Default Text - ANSI version」。您可以用「Cut」或「Copy」命令将字串传递给剪贴簿，用「Cut」或「Delete」命令从程式中删除字串。「Paste」命令将剪贴簿中的文字内容复制到 `pText`。在 `WM_PAINT` 讯息处理期间，`pText` 将字串显示在程式的显示区域。

如果您先在 Unicode 版的 CLIPTEXT 中选择了「Copy」命令，然後在非 Unicode 版中选择「Paste」命令，那么您就能看到文字已经从 Unicode 转换成了 ANSI。类似地，如果您执行相反的操作，那么文字就会从 ANSI 转换成 Unicode。

复杂的剪贴簿用法

我们已经看到，在将资料准备好之後，从剪贴簿传输资料时需要四个呼叫：

```
OpenClipboard            (hwnd) ;
EmptyClipboard           () ;
SetClipboardData         (iFormat, hGlobal) ;
CloseClipboard           () ;
存取这些资料需要三个呼叫
OpenClipboard (hwnd) ;
hGlobal = GetClipboardData (iFormat) ;
其他行程式
CloseClipboard () ;
```

在 `GetClipboardData` 和 `CloseClipboard` 呼叫之间，可以复制剪贴簿资料或以其他方式来使用它。很多应用程式都需要采用这种方法，但也可以用更复杂的方式来使用剪贴簿。

利用多个资料项目

当打开剪贴簿并把资料传送给它时，必须先呼叫 `EmptyClipboard`，通知 Windows 释放或删除剪贴簿上的内容。不能在现有的剪贴簿内容中附加其他东西。所以，从这种意义上说，剪贴簿每次只能保留一个资料项目。

但是，可以在 `EmptyClipboard` 和 `CloseClipboard` 呼叫之间多次呼叫 `SetClipboardData`，每次都使用不同的剪贴簿格式。例如，如果想在剪贴簿中储存一个很短的文字字串，可以把这个文字写入 metafile，也可以把这个文字写入点阵图。把点阵图选进记忆体装置内容中，并把这个字串写进点阵图中。利用这种方法可以使字串不仅能为从剪贴簿上读取文字的程式所使用，也可以为从剪贴簿上读取点阵图和 metafile 的程式所使用。当然，这些程式并不能知道 metafile 或点阵图实际上包含了一个字串。

如果想把一些代号写到剪贴簿上，对每个代号均可以呼叫 `SetClipboardData`：

```
OpenClipboard      (hwnd) ;
EmptyClipboard     () ;
SetClipboardData   (CF_TEXT, hGlobalText) ;
SetClipboardData   (CF_BITMAP, hBitmap) ;
SetClipboardData   (CF_METAFILEPICT, hGlobalMFP) ;
CloseClipboard     () ;
```

当这三种格式的资料同时位於剪贴簿上时，用 `CF_TEXT`、`CF_BITMAP` 或 `CF_METAFILEPICT` 参数呼叫 `IsClipboardFormatAvailable` 将传回 `TRUE`。通过下列呼叫程式可以存取这些代码：

```
hGlobalText = GetClipboardData (CF_TEXT) ;
```

或

```
hBitmap = GetClipboardData (CF_BITMAP) ;
```

或

```
hGlobalMFP = GetClipboardData (CF_METAFILEPICT) ;
```

下一次程式呼叫 `EmptyClipboard` 时，Windows 将释放或删除剪贴簿上保留的所有三个代号。

在将不同的文字格式、不同的点阵图格式或者不同的 metafile 格式添加到剪贴簿时，不要使用这种技术。只使用一种文字格式、一种点阵图格式以及一种 metafile 格式。就像我所说的那样，Windows 将在 `CF_TEXT`、`CF_OEMTEXT` 和 `CF_UNICODETEXT` 之间转换，也可以在 `CF_BITMAP` 和 `CF_DIB` 之间，以及在 `CF_METAFILEPICT` 和 `CF_ENHMETAFILE` 之间进行转换。

透过首先打开剪贴簿，然後呼叫 `EnumClipboardFormats`，程式可以确定剪贴簿储存的所有格式。开始时设定变数 `iFormat` 为 0：

```
iFormat = 0 ;
OpenClipboard (hwnd) ;
```

现在从 0 值开始逐次进行连续的 EnumClipboardFormats 呼叫。函式将为目前在剪贴簿中的每种格式传回一个正的 iFormat 值。当函式传回 0 时，表示完成：

```
while (iFormat = EnumClipboardFormats (iFormat))
{
    各个 iFormat 值的处理方式
}
CloseClipboard () ;
```

您可以通过下面的呼叫来取得目前在剪贴簿中之不同格式的个数：

```
iCount = CountClipboardFormats () ;
```

延迟提出

当把资料放入剪贴簿中时，一般来说要制作一份资料的副本，并将包含这份副本的记忆体块代号传给剪贴簿。对非常大的资料项目来说，这种方法会浪费记忆体空间。如果使用者不想把资料粘贴到另一个程式里，那么，在被其他内容取代之前，它将一直占据著记忆体空间。

通过使用一种叫做「延迟提出」的技术可以避免这个问题。实际上，直到另一个程式需要资料，程式才提供这份资料。为此，不将资料代号传给 Windows，而是在 SetClipboardData 呼叫中使用 NULL：

```
OpenClipboard          (hwnd) ;
EmptyClipboard         () ;
SetClipboardData (iFormat, NULL) ;
CloseClipboard         () ;
```

可以有多个使用不同 iFormat 值的 SetClipboardData 呼叫，对其中某些呼叫可使用 NULL 值。而对其他一些则使用实际的代号值。

前面的过程比较简单，以下的过程就要稍微复杂一些了。当另一个程式呼叫 GetClipboardData 时，Windows 将检查那种格式的代号是否为 NULL。如果是，Windows 将给「剪贴簿所有者」（您的程式）发送一个讯息，要求取得资料的实际代号，这时您的程式必须提供这个代号。

更具体地说，「剪贴簿所有者」是将资料放入剪贴簿的最後一个视窗。当一个程式呼叫 OpenClipboard 时，Windows 储存呼叫这个函式时所用的视窗代号，这个代号标示打开剪贴簿的视窗。一旦收到一个 EmptyClipboard 呼叫，Windows 就使这个视窗作为新的剪贴簿所有者。

使用延迟提出技术的程式在它的视窗讯息处理程式中必须处理三个讯息：WM_RENDERFORMAT、WM_RENDERALLFORMATS 和 WM_DESTROYCLIPBOARD。当另一个

程式呼叫 `GetClipboardData` 时，Windows 给视窗讯息处理程式发送一个 `WM_RENDERFORMAT` 讯息，`wParam` 的值是所要求的格式。在处理 `WM_RENDERFORMAT` 讯息时，不要打开或清空剪贴簿。为 `wParam` 所指定的格式建立一个整体记忆体块，把数据传给它，并用正确的格式和相应代号呼叫 `SetClipboardData`。很明显地，为了在处理 `WM_RENDERFORMAT` 时正确地构造出此资料，需要在程式中保留这些资讯。当另一个程式呼叫 `EmptyClipboard` 时，Windows 给您的程式发送一个 `WM_DESTROYCLIPBOARD` 讯息，告诉您不再需要构造剪贴簿资料的资讯。您的程式不再是剪贴簿的所有者。

如果程式在它自己仍然是剪贴簿所有者的时候就要终止执行，并且剪贴簿上仍然包含著该程式用 `SetClipboardData` 设定的 `NULL` 资料代号，它将收到 `WM_RENDERALLFORMATS` 讯息。这时，应该打开剪贴簿，清空它，把资料载入记忆体块中，并为每种格式呼叫 `SetClipboardData`，然後关闭剪贴簿。`WM_RENDERALLFORMATS` 讯息是视窗讯息处理程式最後收到的讯息之一。它後面跟有 `WM_DESTROYCLIPBOARD` 讯息（由於已经提出了所有资料），然後是正常的 `WM_DESTROY` 讯息。

如果您的程式只能向剪贴簿传输一种格式的资料（例如文字），那么您可以把 `WM_RENDERALLFORMATS` 和 `WM_RENDERFORMAT` 处理结合在一起。这些程式码应该类似下面这样：

```
case WM_RENDERALLFORMATS :
    OpenClipboard (hwnd) ;
    EmptyClipboard () ;

                                                    // fall through
case WM_RENDERFORMAT :
    // 将文字放入整体记忆体块
    SetClipboardData (CF_TEXT, hGlobal) ;
    if (message == WM_RENDERALLFORMATS)
        CloseClipboard () ;
    return 0 ;
```

如果您的程式使用好几种剪贴簿格式，那么您可能想为 `wParam` 所要求的格式处理 `WM_RENDERFORMAT`。除非程式在存放构造资料所需的资讯时遇到困难，否则不需要处理 `WM_DESTROYCLIPBOARD` 讯息。

自订资料格式

到目前为止，我们仅处理了 Windows 定义的标准剪贴簿资料格式。但是，您可能想用剪贴簿来储存「自订资料格式」。许多文书处理程式使用这种技术来储存包含著字体和格式化资讯的文字。

初看之下，这个概念似乎是没有意义的。如果剪贴簿的作用是在应用程式

之间传送资料，那么，为什么剪贴簿中要含有只有一个应用程式才能理解的资料呢？答案很简单：剪贴簿允许在同一个程式的内部（或者可能在一个程式中的不同执行实体之间）传送资料。很明显地，这些执行实体能理解它们自己的自订资料格式。

有几种使用自订资料格式的方法。最简单的方法用到一种表面上是标准剪贴簿格式（文字、点阵图或 metafile）的资料，可是该资料实际上只对您的程式有意义。这种情况下，在 SetClipboardData 和 GetClipboardData 呼叫中可使用下列 wFormat 值：CF_DSPTEXT、CF_DSPBITMAP、CF_DSPMETAFILEPICT 或 CF_DSPENHMETAFIELD（字母 DSP 代表「显示器」）。这些格式允许 Windows 按文字、点阵图或 metafile 来浏览或显示资料。但是，另一个使用常规的 CF_TEXT、CF_BITMAP、CF_DIB、CF_METAFILEPICT 或 CF_ENHMETAFIELD 格式呼叫 GetClipboardData 的程式将不能取得这个资料。

如果用其中一种格式把资料放入剪贴簿中，则必须使用同样的格式读出资料。但是，如何知道资料是来自程式的另一个执行实体，还是来自使用其中某种资料格式的另一个程式呢？这里有一种方法，可以透过下列呼叫首先获得剪贴簿所有者：

```
hwndClipOwner = GetClipboardOwner () ;
```

然後可以得到此视窗代号的视窗类别名称：

```
TCHAR szClassName [32] ;  
//其他行程式  
GetClassName (hwndClipOwner, szClassName, 32) ;
```

如果类别名称与程式名称相同，那么资料是由程式的另一个执行实体传送到剪贴簿中的。

使用自订资料格式的第二种方法涉及到 CF_OWNERDISPLAY 旗标。SetClipboardData 的整体记忆体代号是 NULL：

```
SetClipboardData (CF_OWNERDISPLAY, NULL) ;
```

这是某些文书处理程式在 Windows 的剪贴簿浏览器的显示区域中显示格式化文字时所采用的方法。很明显地，剪贴簿浏览器不知道如何显示这种格式化文字。当一个文书处理程式指定 CF_OWNERDISPLAY 格式时，它也就承担起在剪贴簿浏览器的显示区域中绘图的责任。

由於整体记忆体代号为 NULL，所以用 CF_OWNERDISPLAY 格式（剪贴簿所有者）呼叫 SetClipboardData 的程式必须处理由 Windows 发往剪贴簿所有者的延迟提出讯息、以及 5 条附加讯息。这 5 个讯息是由剪贴簿浏览器发送到剪贴簿所有者的：

WM_ASKCBFORMATNAME 剪贴簿浏览器把这个讯息发送到剪贴簿所有者，以得到资料格式名称。lParam 参数是指向缓冲区的指标，wParam 是这个缓冲区能

容纳的最大字元数目。剪贴簿所有者必须把剪贴簿资料格式的名字复制到这个缓冲区中。

WM_SIZECLIPBOARD 这个讯息通知剪贴簿所有者，剪贴簿浏览器的显示区域大小已发生了变化。wParam 参数是剪贴簿浏览器的代号，lParam 是指向包含新尺寸的 RECT 结构的指标。如果 RECT 结构中都是 0，则剪贴簿浏览器退出或最小化。尽管 Windows 的剪贴簿浏览器只允许它自己的一个执行实体执行，但其他剪贴簿浏览器也能把这个讯息发送给剪贴簿所有者。应付多个剪贴簿浏览器并非不可能（假定 wParam 标识特定的浏览器），但剪贴簿所有者处理起来也不容易。

WM_PAINTCLIPBOARD 这个讯息通知剪贴簿所有者修改剪贴簿浏览器的显示区域。同时，wParam 是剪贴簿浏览器视窗的代号，lParam 是指向 PAINTSTRUCT 结构的整体指标。剪贴簿所有者可以从此结构的 hdc 栏中得到剪贴簿浏览器装置内容的代号。

WM_HSCROLLCLIPBOARD 和 **WM_VSCROLLCLIPBOARD** 这两个讯息通知剪贴簿所有者，使用者已经卷动了剪贴簿浏览器的卷动列。wParam 参数是剪贴簿浏览器视窗的代号，lParam 的低字组是卷动请求，并且，如果低字组是 SB_THUMBPOSITION，那么 lParam 的高字组就是滑块位置。

处理这些讯息比较麻烦，看来并不值得这样做。但是，这种处理对使用者来说是有益的。当从文书处理程式把文字复制到剪贴簿时，使用者在剪贴簿浏览器的显示区域中看见文字还保持著格式时心里会舒坦些。

使用私有剪贴簿资料格式的第三种方法是注册自己的剪贴簿格式名。您向 Windows 提供格式名，Windows 给程式提供一个序号，它可以用作 SetClipboardData 和 GetClipboardData 的格式参数。一般来说，采用这种方法的程式也要以一种标准格式把资料复制到剪贴簿。这种方法允许剪贴簿浏览器在它的显示区域中显示资料（没有与 CF_OWNERDISPLAY 相关的冲突），并且允许其他程式从剪贴簿上复制资料。

例如，假定我们已经编写了一个以点阵图格式、metafile 格式和自己的已注册的剪贴簿格式把资料复制到剪贴簿中的向量绘图程式。剪贴簿浏览器将显示 metafile 或者点阵图，其他从剪贴簿上读取点阵图和 metafile 的程式将获得这几种格式。但是，当我们的向量绘图程式需要从剪贴簿上读数据时，它会按照自己已注册的格式复制资料，这是因为这种格式可能包含著比点阵图档案或者 metafile 更多的资讯。

程式透过下面的呼叫来注册一个新的剪贴簿格式：

```
iFormat = RegisterClipboardFormat (szFormatName) ;
```


iFormat 的值介於 0xC000 和 0xFFFF 之间。剪贴簿浏览器 (或一个通过呼叫 EnumClipboardFormats 取得目前所有剪贴簿资料格式的程式) 可以取得这种资料格式的 ASCII 名称, 这是通过下面呼叫实作的:

```
GetClipboardFormatName (iFormat, psBuffer, iMaxCount) ;
```

Windows 将多达 iMaxCount 个字元复制到 psBuffer 中。

使用这种方法把资料复制到剪贴簿中的程式写作者, 可能需要公开资料格式名称和实际的资料格式。如果这个程式流行起来, 那么其他程式就会以这种格式从剪贴簿中复制资料。

实作剪贴簿浏览器

监视剪贴簿内容变化的程式称为「剪贴簿浏览器」。您可以在 Windows 中得到一个剪贴簿浏览器, 但是您也可以编写自己的剪贴簿浏览器程式。剪贴簿浏览器通过传递到浏览器视窗讯息处理程式的讯息来监视剪贴簿内容的变化。

剪贴簿浏览器链

任意数量的剪贴簿浏览器應用程式都可以同时在 Windows 下执行, 它们都可以监视剪贴簿内容的变化。但是, 从 Windows 的角度来看, 只存在一个剪贴簿浏览器, 我们称之为「目前剪贴簿浏览器」。Windows 只保留一个识别目前剪贴簿浏览器的视窗代号, 并且当剪贴簿的内容发生变化时只把讯息发送到那个视窗中。

剪贴簿浏览器應用程式有必要加入「剪贴簿浏览器链」, 以便执行的所有剪贴簿浏览器都可以收到 Windows 发送给目前剪贴簿浏览器的讯息。当一个程式将自己注册为一个剪贴簿浏览器时, 它就成为目前的剪贴簿浏览器。Windows 把先前的目前浏览器视窗代号交给这个程式, 并且此程式将储存这个代号。当此程式收到一个剪贴簿浏览器讯息时, 它把这个讯息发送给剪贴簿链中下一个程式的视窗讯息处理程式。

剪贴簿浏览器的函式和讯息

程式透过呼叫 SetClipboardViewer 函式可以成为剪贴簿浏览器链的一部分。如果程式的主要作用是作为剪贴簿浏览器, 那么这个程式在 WM_CREATE 讯息处理期间可以呼叫这个函式, 该函式传回前一个目前剪贴簿浏览器的视窗代号。程式应该把这个代号储存在静态变数中:

```
static HWND hwndNextViewer ;  
//其他行程式  
case WM_CREATE :
```

```
//其他程式
```

```
hwndNextViewer = SetClipboardViewer (hwnd) ;
```

如果在 Windows 的一次执行期间，您的程式成为剪贴簿浏览器的第一个程式，那么 hwndNextViewer 将为 NULL。

不管剪贴簿中的内容怎样变化，Windows 都将把 WM_DRAWCLIPBOARD 讯息发送给目前的剪贴簿浏览器（最近注册为剪贴簿浏览器的视窗）。剪贴簿浏览器链中的每个程式都应该用 SendMessage 把这个讯息发送到下一个剪贴簿浏览器。浏览器链中的最后一个程式（第一个将自己注册为剪贴簿浏览器的视窗）所储存的 hwndNextViewer 为 NULL。如果 hwndNextViewer 为 NULL，那么程式只简单地将控制项权还给系统而已，而不向其他程式发送任何讯息（不要把 WM_DRAWCLIPBOARD 讯息和 WM_PAINTCLIPBOARD 讯息混淆了。WM_PAINTCLIPBOARD 是由剪贴簿浏览器发送给使用 CF_OWNERDISPLAY 剪贴簿资料格式的程式，而 WM_DRAWCLIPBOARD 讯息是由 Windows 发往目前剪贴簿浏览器的）。

处理 WM_DRAWCLIPBOARD 讯息的最简单方法是将讯息发送给下一个剪贴簿浏览器（除非 hwndNextViewer 为 NULL），并使视窗的显示区域无效：

```
case WM_DRAWCLIPBOARD :
    if (hwndNextViewer)
        SendMessage (hwndNextViewer, message, wParam, lParam) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
```

在处理 WM_PAINT 讯息处理期间，通过使用常规的 OpenClipboard、GetClipboardData 和 CloseClipboard 呼叫可以读取剪贴簿的内容。

当某个程式想从剪贴簿浏览器链中删除它自己时，它必须呼叫 ChangeClipboardChain。这个函式接收脱离浏览器链的程式之视窗代号，和下一个剪贴簿浏览器的视窗代号：

```
ChangeClipboardChain (hwnd, hwndNextViewer) ;
```

当程式呼叫 ChangeClipboardChain 时，Windows 发送 WM_CHANGECHAIN 讯息给目前的剪贴簿浏览器。wParam 参数是从链中移除它自己的那个浏览器视窗代号（ChangeClipboardChain 的第一个参数），lParam 是从链中移除自己後的下一个剪贴簿浏览器的视窗代号（ChangeClipboardChain 的第二个参数）。

当程式接收到 WM_CHANGECHAIN 讯息时，必须检查 wParam 是否等於已经储存的 hwndNextViewer 的值。如果是这样，程式必须设定 hwndNextViewer 为 lParam。这项工作保证将来的 WM_DRAWCLIPBOARD 讯息不会发送给从剪贴簿浏览器链中删除了自己的视窗。如果 wParam 不等於 hwndNextViewer，并且 hwndNextViewer 不为 NULL，则把讯息送到下一个剪贴簿浏览器。

```
case WM_CHANGECHAIN :
```

```

    if ((HWND) wParam == hwndNextViewer)
        hwndNextViewer = (HWND) lParam ;

    else if (hwndNextViewer)
        SendMessage (hwndNextViewer, message, wParam,
lParam) ;
    return 0 ;

```

不一定要使用 else if 叙述，它只用於保证 hwndNextViewer 为非 NULL 的值。hwndNextViewer 的值为 NULL 时，执行这段程式码的程式就是链中最後一个浏览器，而这是不可能的。

当程式快结束时，如果它仍然在剪贴簿浏览器链中，则必须从链中删除它。您可以在处理 WM_DESTROY 讯息时呼叫 ChangeClipboardChain 来完成这项工作。

```

case WM_DESTROY :
    ChangeClipboardChain (hwnd, hwndNextViewer) ;
    PostQuitMessage (0) ;
    return 0 ;

```

Windows 还有一个允许程式获得第一个剪贴簿浏览器视窗代号的函式：

```
hwndViewer = GetClipboardViewer () ;
```

一般来说不需要这个函式。如果没有目前的剪贴簿浏览器，则传回值为 NULL。

下面是一个说明剪贴簿浏览器链如何工作的例子。当 Windows 刚启动时，目前剪贴簿浏览器是 NULL：

剪贴簿浏览器： NULL

一个具有 hwnd1 视窗代号的程式呼叫 SetClipboardViewer。这个函式传回的 NULL 成为这个程式中的 hwndNextViewer 值：

目前剪贴簿浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

第二个具有 hwnd2 视窗代号的程式呼叫 SetClipboardViewer，并传回 hwnd1：

目前的剪贴簿浏览器： hwnd2

hwnd2 的下一个浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

每三个程式 (hwnd3) 和第四个程式 (hwnd4) 也呼叫 SetClipboardViewer，并且传回 hwnd2 和 hwnd3：

目前的剪贴簿浏览器： hwnd4

hwnd4 的下一个浏览器： hwnd3

hwnd3 的下一个浏览器： hwnd2

hwnd2 的下一个浏览器： hwnd1

hwnd1 的下一个浏览器: NULL

当剪贴簿的内容发生变化时, Windows 发送一个 WM_DRAWCLIPBOARD 讯息给 hwnd4, hwnd4 发送讯息给 hwnd3, hwnd3 发送讯息给 hwnd2, hwnd2 发送讯息给 hwnd1, hwnd1 传回。

现在 hwnd2 决定通过下列呼叫从链中删除自己:

ChangeClipboardChain (hwnd2, hwnd1) ;

Windows 将 wParam 等於 hwnd2、lParam 等於 hwnd1 的 WM_CHANGECHAIN 讯息发送给 hwnd4。由於 hwnd4 的下一个浏览器是 hwnd3, 所以 hwnd4 把这个讯息传给 hwnd3。现在 hwnd3 注意到 wParam 等於它的下一个浏览器(hwnd2), 所以将下一个浏览器设定为 lParam (hwnd1)并且传回。这样工作就完成了。现在剪贴簿浏览器链如下:

目前剪贴簿浏览器: hwnd4

hwnd4 的下一个浏览器: hwnd3

hwnd3 的下一个浏览器: hwnd1

hwnd1 的下一个浏览器: NULL

一个简单的剪贴簿浏览器

剪贴簿浏览器不一定要像 Windows 所提供的那样完善, 例如, 剪贴簿浏览器可以只显示一种剪贴簿资料格式。程式 12-2 中所示的 CLIPVIEW 程式是一种只能显示 CF_TEXT 格式的剪贴簿浏览器。

程式 12-2 CLIPVIEW

```
CLIPVIEW.C
/*-----
    CLIPVIEW.C --      Simple Clipboard Viewer
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain      (      HINSTANCE hInstance, HINSTANCE hPrevInstance,
                          PSTR szCmdLine, int iCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("ClipView") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
```

```

    wndclass.cbClsExtra          = 0 ;
    wndclass.cbWndExtra          = 0 ;
    wndclass.hInstance          = hInstance ;
    wndclass.hIcon               = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor             = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName        = NULL ;
    wndclass.lpszClassName       = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName,
                        TEXT ("Simple Clipboard Viewer (Text Only)"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc      (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND      hwndNextViewer ;
    HGLOBAL          hGlobal ;
    HDC              hdc ;
    PTSTR            pGlobal ;
    PAINTSTRUCT      ps ;
    RECT             rect ;

    switch (message)
    {
    case WM_CREATE:

```

```
        hwndNextViewer = SetClipboardViewer (hwnd) ;
        return 0 ;

    case WM_CHANGECHAIN:
        if ((HWND) wParam == hwndNextViewer)
            hwndNextViewer = (HWND) lParam ;

        else if (hwndNextViewer)
            SendMessage (hwndNextViewer, message,
wParam, lParam) ;

        return 0 ;
    case WM_DRAWCLIPBOARD:
        if (hwndNextViewer)
            SendMessage (hwndNextViewer, message, wParam,
lParam) ;

        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect (hwnd, &rect) ;
        OpenClipboard (hwnd) ;

#ifdef UNICODE
        hGlobal = GetClipboardData (CF_UNICODETEXT) ;
#else
        hGlobal = GetClipboardData (CF_TEXT) ;
#endif

        if (hGlobal != NULL)
        {
            pGlobal = (PTSTR) GlobalLock (hGlobal) ;
            DrawText (hdc, pGlobal, -1, &rect, DT_EXPANDTABS) ;
            GlobalUnlock (hGlobal) ;
        }

        CloseClipboard () ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        ChangeClipboardChain (hwnd, hwndNextViewer) ;
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CLIPVIEW 依上面所讨论的方法来处理 WM_CREATE、WM_CHANGECHAIN、WM_DRAWCLIPBOARD 和 WM_DESTROY 讯息。WM_PAINT 讯息处理打开剪贴簿，并用 CF_TEXT 格式呼叫 GetClipboardData。如果函式传回一个整体记忆体代号，那么 CLIPVIEW 将锁定它，并用 DrawText 在显示区域显示文字。

处理标准格式（如 Windows 提供的那个剪贴簿一样）以外的资料格式的剪贴簿浏览器还需要完成一些其他工作，比如显示剪贴簿中目前所有资料格式的名称。使用者可以通过呼叫 EnumClipboardFormats 并使用 GetClipboardFormatName 得到非标准资料格式名称来完成这项工作。使用 CF_OWNERDISPLAY 资料格式的剪贴簿浏览器必须把下面四个讯息送往剪贴簿资料的拥有者以显示该资料：

- WM_PAINTCLIPBOARD
- WM_SIZECLIPBOARD
- WM_VSCROLLCLIPBOARD
- WM_HSCROLLCLIPBOARD

如果您想编写这样的剪贴簿浏览器，那么必须使用 GetClipboardOwner 获得剪贴簿所有者的视窗代号，并当您需要修改剪贴簿的显示区域时，将这些讯息发送给该视窗。

第十三章 使用印表机

为了处理文字和图形而使用视讯显示器时，装置无关的概念看来非常完美，但对於印表机，装置无关的概念又怎样呢？

总的说来，效果也很好。在 Windows 程式中，用於视讯显示器的 GDI 函式一样可以在印表纸上列印文字和图形，在以前讨论的与装置无关的许多问题（多数都与平面显示的尺寸、解析度以及颜色数有关）都可以用相同的方法解决。当然，一台印表机不像使用阴极射线管的显示器那么简单，它们使用的是印表纸。它们之间有一些比较大的差异。例如，我们从来不必考虑视讯显示器没有与显示卡连结好，或者显示器出现「萤幕空间不够」的错误，但印表机 off line 和缺纸却是经常会遇到的问题。

我们也不必担心显示卡不能执行某些图形操作，更不用担心显示卡能否处理图形，因为，如果它不能处理图形，就根本不能使用 Windows。但有些印表机不能列印图形（尽管它们能在 Windows 环境中使用）。绘图机尽管可以列印向量图形，却存在位元图块的传输问题。

以下是其他一些需要考虑的问题：

- 印表机比视讯显示器慢。尽管我们没有机会将程式性能调整到最佳状态，却不必担心视讯显示器更新所需的时间。然而，没有人想在做其他工作前一直等待印表机完成列印任务。
- 程式可以用新的输出覆盖原有的显示输出，以重新使用视讯显示器表面。这对印表机是不可能的，印表机只能用完一整页纸，然後在新一页的纸上列印新的内容。
- 在视讯显示器上，不同的應用程式都被视窗化。而对於印表机，不同應用程式的输出必须分成不同的文件或列印作业。

为了在 GDI 的其余部分中加入印表机支援功能，Windows 提供几个只用於印表机的函式。这些限用在印表机上的函式（StartDoc、EndDoc、StartPage 和 EndPage）负责将印表机的输出组织列印到纸页上。而一个程式呼叫普通的 GDI 函式在一张纸上显示文字和图形，和在萤幕上显示的方式一样。

在第十五、十七和十八章有列印点阵图、格式化的文字以及 metafile 的其他资讯。

列印入门

当您在 Windows 下使用印表机时，实际上启动了一个包含 GDI32 动态连结

程式库模组、列印驱动程序动态连结模组（带.DRV 副档名）、Windows 幕後列印程式，以及有用到的其他相关模组。在写印表机列印程式之前，让我们先看一看这个程序是如何进行的。

列印和背景处理

当应用程式要使用印表机时，它首先使用 CreateDC 或 PrintDlg 来取得指向印表机装置内容的代号，於是使得印表机装置驱动程序动态连结程式库模组被载入到记忆体（如果还没有载入记忆体的话）并自己进行初始化。然後，程式呼叫 StartDoc 函式，通知说一个新文件开始了。StartDoc 函式是由 GDI 模组来处理的，GDI 模组呼叫印表机装置驱动程序中的 Control 函式告诉装置驱动程序准备进行列印。

列印一个文件的程序以 StartDoc 呼叫开始，以 EndDoc 呼叫结束。这两个呼叫对於在文件页面上书写文字或者绘制图形的 GDI 命令来说，其作用就像分隔页面的书挡一样。每页本身是这样来划清界限的：呼叫 StartPage 来开始一页，呼叫 EndPage 来结束该页。

例如，如果应用程式想在一页纸上画出一个椭圆，它首先呼叫 StartDoc 开始列印任务，然後再呼叫 StartPage 通知这是新的一页，接著呼叫 Ellipse，正如同在萤幕上画一个椭圆一样。GDI 模组通常将程式对印表机装置内容做出的 GDI 呼叫储存在磁片上的 metafile 中，该档案名以字串~EMF（代表「增强型 metafile」）开始，且以.TMP 为副档名。然而，我在这里应该指出，印表机驱动程序可能会跳过这一步骤。

当绘制第一页的 GDI 呼叫结束时，应用程式呼叫 EndPage。现在，真正的工作开始了。印表机驱动程序必须把存放在 metafile 中的各种绘图命令翻译成印表机输出资料。绘制一页图形所需的印表机输出资料量可能非常大，特别是当印表机没有高级页面制作语言时，更是如此。例如，一台每英寸 600 点且使用 8.5 11 英寸印表纸的雷射印表机，如果要定义一个图形页，可能需要 4 百万以上位元组的资料。

为此，印表机驱动程序经常使用一种称作「列印分带」的技术将一页分成若干称为「输出带」的矩形。GDI 模组从印表机驱动程序取得每个输出带的大小，然後设定一个与目前要处理的输出带相等的剪裁区，并为 metafile 中的每个绘图函式呼叫印表机装置驱动程式的 Output 函式，这个程序叫做「将 metafile 输出到装置驱动程序」。对装置驱动程序所定义的页面上的每个输出带，GDI 模组必须将整个 metafile「输出到」装置驱动程序。这个程序完成以後，该 metafile 就可以删除了。

对每个输出带，装置驱动程式将这些绘图函式转换为在印表机上列印这些图形所需要的输出资料。这种输出资料的格式是依照印表机的特性而异的。对点阵印表机，它将是包括图形序列在内的一系列控制命令序列的集合（印表机驱动程式也能呼叫在 GDI 模组中的各种「helper」辅助常式，用来协助这种输出的构造）。对於带有高阶页面制作语言（如 PostScript）的雷射印表机，印表机将用这种语言进行输出。

列印驱动程式将列印输出的每个输出带传送到 GDI 模组。随後，GDI 模组将该列印输出存入另一个暂存档案中，该暂存档案名以字串~SPL 开始，带有.TMP 副档名。当处理好整页之後，GDI 模组对幕後列印程式进行一个程序间呼叫，通知它一个新的列印页已经准备好了。然後，应用程式就转向处理下一页。当应用程式处理完所有要列印的输出页後，它就呼叫 EndDoc 发出一个信号，表示列印作业已经完成。图 13-1 显示了应用程式、GDI 模组和列印驱动程式的交互作用程序。

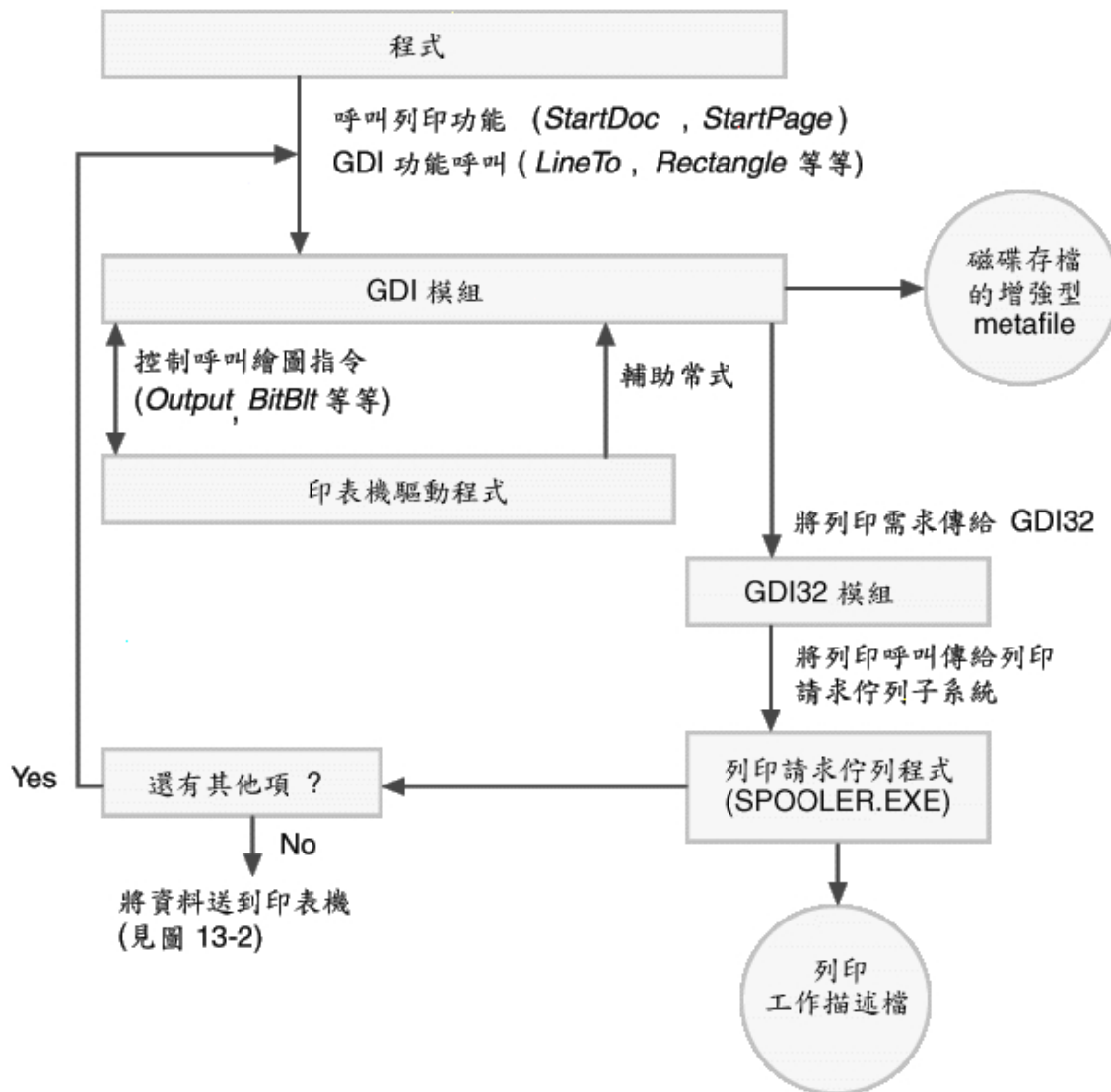


图 13-1 应用程式、GDI 模组、列印驱动程式和列印佇列程式的交互作用过程

Windows 幕後列印程式实际上是几个元件的一种组合（见表 13-1）。

表 13-1

列印伫列程式元件	说明
列印请求伫列程式	将资料流程传递给列印功能提供者
本地列印功能提供者	为本地印表机建立背景档案
网路列印功能提供者	为网路印表机建立背景档案
列印处理程式	将列印伫列中与装置无关的资料转换为针对目的印表机的格式
列印埠监视程式	控制项连结印表机的埠
列印语言监视程式	控制项可以双向通讯的印表机，设定装置设定并检测印表机状态

列印伫列程式可以减轻应用程式的列印负担。 Windows 在启动时就载入列印伫列程式，因此，当应用程式开始列印时，它已经是活动的了。当程式列印一个档案时，GDI 模组会建立包含列印输出资料的档案。幕後列印程式的任务是 将这些档案发往印表机。GDI 模组发出一个讯息来通知它一个新的列印作业开始，然後它开始读档案并将档案直接传送到印表机。为了传送这些档案，列印 伫列程式依照印表机所连结的并列埠或串列埠使用各种不同的通信函式。在列 印伫列程式向印表机发送档案的操作完成後，它就将包含输出资料的暂存档案 删除。这个交互作用过程如图 13-2 所示。

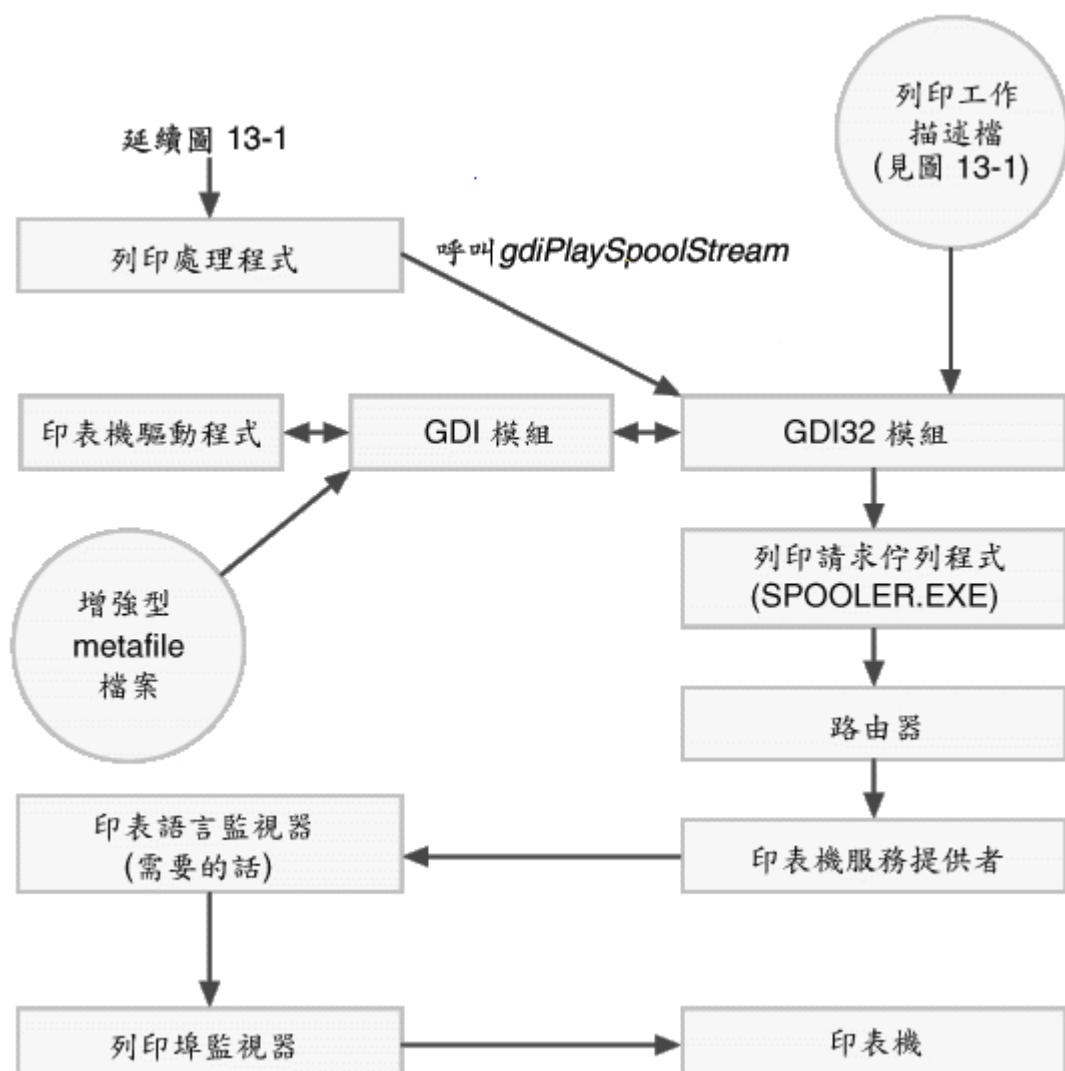


圖 13-2 幕後列印程式的操作程序

这个程序的大部分对应用程序来说是透明的。从应用程序的角度来看，「列印」只发生在 GDI 模組将所有列印输出资料储存在磁片档案中的时候，在这之後（如果列印是由第二个执行绪来操作的，甚至可以在这之前）应用程序可以自由地进行其他操作。真正的档案列印操作成了幕後列印程式的任务，而不是应用程序的任务。通过印表机档案夹，使用者可以暂停列印作业、改变作业的优先顺序或取消列印作业。这种管理方式使应用程序能更快地将列印资料以即时方式列印，况且这样必须等到列印完一页後才能处理下一页。

我们已经描述了一般的列印原理，但还有一些例外情况。其中之一是 Windows 程式要使用印表机时，并非一定需要幕後列印程式。使用者可以在印表机属性表格的详细资料属性页中关闭印表机的背景操作。

为什么使用者希望不使用背景操作呢？因为使用者可能使用了比 Windows 列印佇列程式更快的硬体或软体幕後列印程式，也可能是印表机在一个自身带有列印佇列器的网路上使用。一般的规则是，使用一个列印佇列程式比使用两个列印佇列程式更快。去掉 Windows 幕後列印程式可以加快列印速度，因为列

印输出资料不必储存在硬碟上，而可以直接输出到印表机，并被外部的硬体列印伫列器或软体的幕後列印程式所接收。

如果没有启用 Windows 列印伫列程式，GDI 模组就不把来自装置驱动程式的列印输出资料存入档案中，而是将这些输出资料直接输出到列印输出埠。与列印伫列程式进行的列印不同，GDI 进行的列印一定会让应用程式暂停执行一段时间（特别是进行列印中的程式）直到列印完成。

还有另一个例外。通常，GDI 模组将定义一页所需的所有函式存入一个增强型 metafile 中，然後替驱动程式定义的每个列印输出带输出一遍该 metafile 到列印驱动程式中。然而，如果列印驱动程式不需要列印分带的话，就不会建立这个 metafile；GDI 只需简单地将绘图函式直接送往驱动程式。进一步的变化是，应用程式也可能得承担起对列印输出资料进行列印分带的责任，这就使得应用程式中的列印程式码更加复杂了，但却免去了 GDI 模组建立 metafile 的麻烦。这样，GDI 只需简单地为每个输出带将函式传到列印驱动程式。

或许您现在已经发现了从一个 Windows 应用程式进行列印操作要比使用视讯显示器的负担更大，这样可能出现一些问题——特别是，如果 GDI 模组在建立 metafile 或列印输出档案时耗尽了磁碟空间。您可以更关切这些问题，并尝试著处理这些问题并告知使用者，或者您当然也可以置之不理。

對於一个应用程式，列印文件的第一步就是如何取得印表机装置的内容。

印表机装置内容

正如在视讯显示器上绘图前需要得到装置内容代号一样，在列印之前，使用者必须取得一个印表机装置内容代号。一旦有了这个代号（并为建立一个新文件呼叫了 StartDoc 以及呼叫 StartPage 开始一页），就可以用与使用视讯显示装置内容代号相同的方法来使用印表机装置内容代号，该代号即为各种 GDI 呼叫的第一个参数。

大多数应用程式经由呼叫 PrintDlg 函式打开一个标准的列印对话方块（本章後面会展示该函式的用法）。这个函式还为使用者提供了一个在列印之前改变印表机或者指定其他特性的机会。然後，它将印表机装置内容代号交给应用程式。该函式能够省下应用程式的一些工作。然而，某些应用程式（例如 Notepad）仅需要取得印表机装置内容，而不需要那个对话方块。要做到这一点，需要呼叫 CreateDC 函式。

在第五章中，您已知道如何通过如下的呼叫来为整个视讯显示器取得指向装置内容的代号：

```
hdc = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
```

您也可以使用该函式来取得印表机装置内容代号。然而，对印表机装置内容，CreateDC 的一般语法为：

```
hdc = CreateDC (NULL, szDeviceName, NULL, pInitializationData) ;
```

pInitializationData 参数一般被设为 NULL。szDeviceName 参数指向一个字串，以告诉 Windows 印表机设备的名称。在设定设备名称之前，您必须知道有哪些印表机可用。

一个系统可能有不只一台连结著的印表机，甚至可以有其他程式，如传真软体，将自己伪装成印表机。不论连结的印表机有多少台，都只能有一台被认为是「目前的印表机」或者「内定印表机」，这是使用者最近一次选择的印表机。许多小型的 Windows 程式只使用内定印表机来进行列印。

取得内定印表机装置内容的方式不断在改变。目前，标准的方法是使用 EnumPrinters 函式来获得。该函式填入一个包含每个连结著的印表机资讯的阵列结构。根据所需的细节层次，您还可以选择几种结构之一作为该函式的参数。这些结构的名称为 PRINTER_INFO_x，x 是一个数字。

不幸的是，所使用的函式还取决於您的程式是在 Windows 98 上执行还是在 Windows NT 上执行。程式 13-1 展示了 GetPrinterDC 函式在两种作业系统上工作的用法。

程式 13-1 GETPRNDC

```
GETPRNDC.C
/*-----
   GETPRNDC.C -- GetPrinterDC function
-----*/

#include <windows.h>
HDC GetPrinterDC (void)
{
    DWORD                                dwNeeded, dwReturned ;
    HDC                                  hdc ;
    PRINTER_INFO_4 *                    pinfo4 ;
    PRINTER_INFO_5 *                    pinfo5 ;

    if (GetVersion () & 0x80000000)      // Windows 98
    {
        EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, NULL,
            0, &dwNeeded, &dwReturned) ;
        pinfo5 = malloc (dwNeeded) ;
        EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, (PBYTE)
pinfo5,
            dwNeeded, &dwNeeded, &dwReturned) ;
        hdc = CreateDC (NULL, pinfo5->pPrinterName, NULL, NULL) ;
        free (pinfo5) ;
    }
}
```

```

    }
    else
//Windows NT
    {
        EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
0, &dwNeeded, &dwReturned) ;
        pinfo4 = malloc (dwNeeded) ;
        EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, (PBYTE)
pinfo4,
        dwNeeded, &dwNeeded, &dwReturned) ;
        hdc = CreateDC (NULL, pinfo4->pPrinterName, NULL, NULL) ;
        free (pinfo4) ;
    }
    return hdc ;
}

```

这些函式使用 `GetVersion` 函式来确定程式是执行在 Windows 98 上还是 Windows NT 上。不管是什么作业系统，函式呼叫 `EnumPrinters` 两次：一次取得它所需结构的大小，一次填入结构。在 Windows 98 上，函式使用 `PRINTER_INFO_5` 结构；在 Windows NT 上，函式使用 `PRINTER_INFO_4` 结构。这些结构在 `EnumPrinters` 文件（`/Platform SDK/Graphics and Multimedia Services/GDI/Printing and Print Spooler/Printing and Print Spooler Reference/Printing and Print Spooler Functions/EnumPrinters`，范例小节的前面）中有说明，它们是「容易而快速」的。

修改後的 DEVCAPS 程式

第五章的 `DEVCAPS1` 程式只显示了从 `GetDeviceCaps` 函式获得的关于视讯显示的基本资讯。程式 13-2 所示的新版本显示了关于视讯显示和连结到系统之所有印表机的更多资讯。

程式 13-2 DEVCAPS2

```

DEVCAPS2.C
/*-----
    DEVCAPS2.C --          Displays Device Capability Information (Version 2)
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
void DoBasicInfo                   (HDC, HDC, int, int) ;
void DoOtherInfo                   (HDC, HDC, int, int) ;

```

```

void DoBitCodedCaps (HDC, HDC, int, int, int) ;

typedef struct
{
    int iMask ;
    TCHAR * szDesc ;
}
BITS ;
#define IDM_DEVMODE 1000
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int
iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("DevCaps2") ;
    HWND hwnd ;
    MSG msg ;
    WNDCLASS wndclass ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox ( NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, NULL,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))

```



```

    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static TCHAR                szDevice[32], szWindowText[64] ;
    static int                  cxChar, cyChar,  nCurrentDevice      =
IDM_SCREEN,
        nCurrentInfo           = IDM_BASIC ;
    static DWORD                dwNeeded, dwReturned ;
    static PRINTER_INFO_4 * pinfo4 ;
    static PRINTER_INFO_5 * pinfo5 ;
    DWORD                       i ;
    HDC                         hdc, hdcInfo ;
    HMENU                       hMenu ;
    HANDLE                      hPrint ;
    PAINTSTRUCT                  ps ;
    TEXTMETRIC                   tm ;

    switch (message)
    {
    case WM_CREATE :
        hdc = GetDC (hwnd) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;
        ReleaseDC (hwnd, hdc) ;

// fall through
    case WM_SETTINGCHANGE:
        hMenu = GetSubMenu (GetMenu (hwnd), 0) ;

        while (GetMenuItemCount (hMenu) > 1)
            DeleteMenu (hMenu, 1, MF_BYPOSITION) ;

        // Get a list of all local and remote printers
        //
        // First, find out how large an array we need; this
        // call will fail, leaving the required size in
dwNeeded

        //
        // Next, allocate space for the info array and fill
it

```

```

//
// Put the printer names on the menu

if (GetVersion () & 0x80000000) //
Windows 98
{
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, NULL,
0, &dwNeeded, &dwReturned) ;

    pinfo5 = malloc (dwNeeded) ;

    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, (PBYTE)
pinfo5,
dwNeeded, &dwNeeded, &dwReturned) ;

    for (i = 0 ; i < dwReturned ; i++)
    {
        AppendMenu (hMenu, (i+1) % 16 ? 0 :
MF_MENUBARBREAK, i + 1,
pinfo5[i].pPrinterName) ;
    }
    free (pinfo5) ;
}
else
// Windows NT
{
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
0, &dwNeeded, &dwReturned) ;
    pinfo4 = malloc (dwNeeded) ;
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, (PBYTE)
pinfo4,
dwNeeded, &dwNeeded, &dwReturned) ;
    for (i = 0 ; i < dwReturned ; i++)
    {
        AppendMenu (hMenu, (i+1) % 16 ? 0 : MF_MENUBARBREAK,
i + 1,
pinfo4[i].pPrinterName) ;
    }
    free (pinfo4) ;
}

AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenu, 0, IDM_DEVMODE, TEXT ("Properties")) ;

wParam = IDM_SCREEN ;
// fall through
case WM_COMMAND :

```

```

        hMenu = GetMenu (hwnd) ;

        if ( LOWORD (wParam) == IDM_SCREEN || // IDM_SCREEN &
Printers
                LOWORD (wParam) < IDM_DEVMODE)
        {
            CheckMenuItem (hMenu, nCurrentDevice, MF_UNCHECKED) ;
            nCurrentDevice = LOWORD (wParam) ;
            CheckMenuItem (hMenu, nCurrentDevice, MF_CHECKED) ;
        }
        else if (LOWORD (wParam) == IDM_DEVMODE) //
Properties selection
        {
            GetMenuString (hMenu, nCurrentDevice,
szDevice,
                sizeof (szDevice) / sizeof (TCHAR), MF_BYCOMMAND);

            if (OpenPrinter (szDevice, &hPrint,
NULL))
            {
                PrinterProperties (hwnd, hPrint) ;
                ClosePrinter (hPrint) ;
            }
        }
        else
// info menu items
        {
            CheckMenuItem (hMenu, nCurrentInfo,
MF_UNCHECKED) ;

            nCurrentInfo = LOWORD (wParam) ;
            CheckMenuItem (hMenu, nCurrentInfo,
MF_CHECKED) ;
        }
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

        case WM_INITMENUPOPUP :
            if (lParam == 0)
                EnableMenuItem (GetMenu (hwnd),
IDM_DEVMODE,
                                nCurrentDevice ==
IDM_SCREENMF_GRAYED : MF_ENABLED) ;
            return 0 ;

        case WM_PAINT :
            lstrcpy (szWindowText, TEXT ("Device Capabilities: ")) ;

            if (nCurrentDevice == IDM_SCREEN)

```

```

        {
            lstrcpy (szDevice, TEXT ("DISPLAY")) ;
            hdcInfo = CreateIC (szDevice, NULL, NULL,
NULL) ;

        }
        else
        {
            hMenu = GetMenu (hwnd) ;
            GetMenuString (hMenu, nCurrentDevice,
szDevice,
            sizeof (szDevice), MF_BYCOMMAND) ;
            hdcInfo = CreateIC (NULL, szDevice, NULL,
NULL) ;

        }

        lstrcat (szWindowText, szDevice) ;
        SetWindowText (hwnd, szWindowText) ;

        hdc = BeginPaint (hwnd, &ps) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

        if (hdcInfo)
        {
            switch (nCurrentInfo)
            {
                case IDM_BASIC :
                DoBasicInfo (hdc, hdcInfo, cxChar, cyChar) ;
                    break ;

                case IDM_OTHER :
                DoOtherInfo (hdc, hdcInfo, cxChar, cyChar) ;
                    break ;

                case IDM_CURVE :
                case IDM_LINE :
                case IDM_POLY :
                case IDM_TEXT :
                DoBitCodedCaps (hdc, hdcInfo, cxChar, cyChar,
nCurrentInfo - IDM_CURVE) ;
                    break ;
            }
            DeleteDC (hdcInfo) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY :

```

```

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

void DoBasicInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
    static struct
    {
        int      nIndex ;
        TCHAR * szDesc ;
    }
    info[] =
    {
        HORZSIZE,          TEXT ("HORZSIZE          Width in
millimeters:"),
        VERTSIZE,          TEXT ("VERTSIZE          Height
in millimeters:"),
        HORZRES,          TEXT ("HORZRES          Width in
pixels:"),
        VERTRES,          TEXT ("VERTRES          Height
in raster lines:"),
        BITSPIXEL,        TEXT ("BITSPIXEL          Color
bits per pixel:"),
        PLANES,           TEXT ("PLANES
Number of color planes:"),
        NUMBRUSHES,       TEXT ("NUMBRUSHES          Number
of device brushes:"),
        NUMPENS,          TEXT ("NUMPENS
Number of device pens:"),
        NUMMARKERS,       TEXT ("NUMMARKERS          Number
of device markers:"),
        NUMFONTS,         TEXT ("NUMFONTS
Number of device fonts:"),
        NUMCOLORS,        TEXT ("NUMCOLORS
Number of device colors:"),
        PDEVICESIZE,      TEXT("PDEVICESIZE      Size      of      device
structure:"),
        ASPECTX,          TEXT("ASPECTX Relative width of pixel:"),
        ASPECTY,          TEXT("ASPECTY Relative height of pixel:"),
        ASPECTXY,         TEXT("ASPECTXY Relative diagonal of pixel:"),
        LOGPIXELSX,       TEXT("LOGPIXELSX Horizontal dots per inch:"),
        LOGPIXELSY,       TEXT("LOGPIXELSY Vertical dots per inch:"),
        SIZEPALETTE,      TEXT("SIZEPALETTE Number of palette entries:"),
        NUMRESERVED,      TEXT("NUMRESERVED Reserved palette entries:"),
        COLORRES,         TEXT("COLORRES Actual color resolution:"),
        PHYSICALWIDTH,    TEXT("PHYSICALWIDTH Printer page pixel width:"),
    }
}

```

```

PHYSICALHEIGHT,TEXT("PHYSICALHEIGHT Printer page pixel height:"),
PHYSICALOFFSETX,TEXT("PHYSICALOFFSETX Printer page x offset:"),
PHYSICALOFFSETY,TEXT("PHYSICALOFFSETY Printer page y offset:")
    } ;
    int    i ;
    TCHAR szBuffer[80] ;

    for (i = 0 ; i < sizeof (info) / sizeof (info[0]) ; i++)
        TextOut (hdc, cxChar, (i + 1) * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("%45s%8d"), info[i].szDesc,
                GetDeviceCaps (hdcInfo, info[i].nIndex))) ;
}

void DoOtherInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
    static BITS clip[] =
    {
        CP_RECTANGLE, TEXT ("CP_RECTANGLE Can Clip To Rectangle:")
    } ;

    static BITS raster[] =
    {
        RC_BITBLT,    TEXT ("RC_BITBLT Capable of simple BitBlt:"),
        RC_BANDING,   TEXT ("RC_BANDING Requires banding support:"),
        RC_SCALING,    TEXT ("RC_SCALING Requires scaling support:"),
        RC_BITMAP64,   TEXT ("RC_BITMAP64 Supports bitmaps >64K:"),
        RC_GDI20_OUTPUT, TEXT ("RC_GDI20_OUTPUT Has 2.0 output calls:"),
        RC_DI_BITMAP,  TEXT ("RC_DI_BITMAP Supports DIB to memory:"),
        RC_PALETTE,    TEXT ("RC_PALETTE Supports a palette:"),
        RC_DIBTODEV,   TEXT ("RC_DIBTODEV Supports bitmap conversion:"),
        RC_BIGFONT,    TEXT ("RC_BIGFONT Supports fonts >64K:"),
        RC_STRETCHBLT, TEXT ("RC_STRETCHBLT Supports StretchBlt:"),
        RC_FLOODFILL,  TEXT ("RC_FLOODFILL Supports FloodFill:"),
        RC_STRETCHDIB, TEXT ("RC_STRETCHDIB Supports StretchDIBits:")
    } ;

    static TCHAR * szTech[] = { TEXT ("DT_PLOTTER (Vector plotter)"),
        TEXT ("DT_RASDISPLAY (Raster display)"),
        TEXT ("DT_RASPRINTER (Raster printer)"),
        TEXT ("DT_RASCAMERA (Raster camera)"),
        TEXT ("DT_CHARSTREAM (Character stream)"),
        TEXT ("DT_METAFILE (Metafile)"),
        TEXT ("DT_DISPFILE (Display file)") } ;

    int    i ;
    TCHAR  szBuffer[80] ;

    TextOut (hdc, cxChar, cyChar, szBuffer,
        wsprintf (szBuffer, TEXT ("%24s%04XH"), TEXT ("DRIVERVERSION:"),

```

```

        GetDeviceCaps (hdcInfo, DRIVERVERSION))) ;
    TextOut (hdc, cxChar, 2 * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("%24s%-40s"), TEXT
("TECHNOLOGY:"),
            szTech[GetDeviceCaps (hdcInfo,
TECHNOLOGY)])) ;
    TextOut (hdc, cxChar, 4 * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("CLIPCAPS (Clipping
capabilities)")) ;
    for (i = 0 ; i < sizeof (clip) / sizeof (clip[0]) ; i++)
        TextOut (hdc, 9 * cxChar, (i + 6) * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("%45s%3s"),
clip[i].szDesc,
            GetDeviceCaps (hdcInfo, CLIPCAPS)
& clip[i].iMask ?
            TEXT ("Yes") : TEXT ("No")))) ;
    TextOut (hdc, cxChar, 8 * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("RASTERCAPS (Raster
capabilities)")) ;
    for (i = 0 ; i < sizeof (raster) / sizeof (raster[0]) ; i++)
        TextOut (hdc, 9 * cxChar, (i + 10) * cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("%45s%3s"),
raster[i].szDesc,
            GetDeviceCaps (hdcInfo,
RASTERCAPS) & raster[i].iMask ?
            TEXT ("Yes") : TEXT ("No")))) ;
}

void DoBitCodedCaps ( HDC hdc, HDC hdcInfo, int cxChar, int cyChar, int iType)
{
    static BITS curves[] =
    {
        CC_CIRCLES,      TEXT ("CC_CIRCLES      Can do circles:"),
        CC_PIE,          TEXT ("CC_PIE      Can do pie wedges:"),
        CC_CHORD,        TEXT ("CC_CHORD      Can do chord arcs:"),
        CC_ELLIPSES,     TEXT ("CC_ELLIPSES   Can do ellipses:"),
        CC_WIDE,         TEXT ("CC_WIDE      Can do wide
borders:"),
        CC_STYLED,       TEXT ("CC_STYLED      Can do styled
borders:"),
        CC_WIDESTYLED,   TEXT ("CC_WIDESTYLED Can do wide and styled
borders:"),
        CC_INTERIORS,   TEXT ("CC_INTERIORS   Can do interiors:")
    } ;

    static BITS lines[] =
    {
        LC_POLYLINE,    TEXT ("LC_POLYLINE Can do polyline:"),

```

```

        LC_MARKER,      TEXT ("LC_MARKER Can do markers:"),
        LC_POLYMARKER,  TEXT ("LC_POLYMARKER Can do polymarkers"),
        LC_WIDE,        TEXT ("LC_WIDE Can do wide lines:"),
        LC_STYLED,       TEXT ("LC_STYLED Can do styled lines:"),
        LC_WIDESTYLED,   TEXT ("LC_WIDESTYLED Can do wide and styled
lines:"),
        LC_INTERIORS,   TEXT ("LC_INTERIORS Can do interiors:")
    } ;

    static BITS poly[] =
    {
        PC_POLYGON,
            TEXT ("PC_POLYGON Can do alternate fill
polygon:"),
        PC_RECTANGLE,   TEXT ("PC_RECTANGLE Can do rectangle:"),
        PC_WINDPOLYGON,
            TEXT ("PC_WINDPOLYGON Can do winding number fill
polygon:"),
        PC_SCANLINE,    TEXT ("PC_SCANLINE Can do scanlines:"),
        PC_WIDE,         TEXT ("PC_WIDE Can do wide borders:"),
        PC_STYLED,       TEXT ("PC_STYLED Can do styled
borders:"),
        PC_WIDESTYLED,
            TEXT ("PC_WIDESTYLED Can do wide and styled
borders:"),
        PC_INTERIORS,   TEXT ("PC_INTERIORS Can do interiors:")
    } ;

    static BITS text[] =
    {
        TC_OP_CHARACTER, TEXT ("TC_OP_CHARACTER Can do character
output precision:"),
        TC_OP_STROKE,    TEXT ("TC_OP_STROKE Can do stroke output
precision:"),
        TC_CP_STROKE,    TEXT ("TC_CP_STROKE Can do stroke clip
precision:"),
        TC_CR_90,        TEXT ("TC_CP_90 Can do 90 degree character
rotation:"),
        TC_CR_ANY,       TEXT ("TC_CR_ANY Can do any character
rotation:"),
        TC_SF_X_YINDEP,  TEXT ("TC_SF_X_YINDEP Can do scaling
independent of X and Y:"),
        TC_SA_DOUBLE,    TEXT ("TC_SA_DOUBLE Can do doubled character
for scaling:"),
        TC_SA_INTEGER,   TEXT ("TC_SA_INTEGER Can do integer
multiples for scaling:"),
        TC_SA_CONTIN,    TEXT ("TC_SA_CONTIN Can do any multiples
for exact scaling:"),

```



```

        TC_EA_DOUBLE,      TEXT ("TC_EA_DOUBLE    Can do double weight
characters:"),
        TC_IA_ABLE,       TEXT ("TC_IA_ABLE      Can do italicizing:"),
        TC_UA_ABLE,       TEXT ("TC_UA_ABLE      Can do underlining:"),
        TC_SO_ABLE,       TEXT ("TC_SO_ABLE      Can do strikeouts:"),
        TC_RA_ABLE,       TEXT ("TC_RA_ABLE      Can do raster fonts:"),
        TC_VA_ABLE,       TEXT ("TC_VA_ABLE      Can do vector fonts:")
    } ;

    static struct
    {
        int                iIndex ;
        TCHAR *            szTitle ;
        BITS                (*pbits)[] ;
        int                iSize ;
    }
    bitinfo[] =
    {
        CURVECAPS,         TEXT ("CURVCAPS (Curve Capabilities)",
                                (BITS (*)[]) curves, sizeof (curves) / sizeof
                                (curves[0])),
        LINECAPS,          TEXT ("LINECAPS (Line Capabilities)",
                                (BITS (*)[]) lines, sizeof (lines) / sizeof
                                (lines[0])),
        POLYGONALCAPS,     TEXT ("POLYGONALCAPS (Polygonal
                                Capabilities)",
                                (BITS (*)[]) poly, sizeof (poly) / sizeof (poly[0]),
        TEXTCAPS,          TEXT ("TEXTCAPS (Text Capabilities)",
                                (BITS (*)[]) text, sizeof (text) / sizeof (text[0])
    } ;

    static TCHAR szBuffer[80] ;
    BITS                (*pbits)[] = bitinfo[iType].pbits ;
    int                i,    iDevCaps    =    GetDeviceCaps    (hdcInfo,
bitinfo[iType].iIndex) ;

    TextOut (hdc, cxChar, cyChar, bitinfo[iType].szTitle,
                                lstrlen (bitinfo[iType].szTitle)) ;
    for (i = 0 ; i < bitinfo[iType].iSize ; i++)
        extOut (hdc, cxChar, (i + 3) * cyChar, szBuffer,
        wsprintf (szBuffer, TEXT ("%55s %3s"), (*pbits)[i].szDesc,
        iDevCaps & (*pbits)[i].iMask ? TEXT ("Yes") : TEXT ("No")));
}

```

[DEVCAPS2.RC \(摘录\)](#)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

```

```

////////////////////////////////////
/
// Menu
DEVCAPS2 MENU DISCARDABLE
BEGIN
    POPUP "&Device"
    BEGIN
        MENUITEM "&Screen",IDM_SCREEN, CHECKED
    END
    POPUP "&Capabilities"
    BEGIN
        MENUITEM "&Basic Information",IDM_BASIC
        MENUITEM "&Other Information",IDM_OTHER
        MENUITEM "&Curve Capabilities",IDM_CURVE
        MENUITEM "&Line Capabilities",IDM_LINE
        MENUITEM "&Polygonal Capabilities",IDM_POLY
        MENUITEM "&Text Capabilities",IDM_TEXT
    END
END

```

[RESOURCE.H](#) (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by DevCaps2.rc

#define IDM_SCREEN      40001
#define IDM_BASIC       40002
#define IDM_OTHER       40003
#define IDM_CURVE       40004
#define IDM_LINE        40005
#define IDM_POLY        40006
#define IDM_TEXT        40007

```

因为 DEVCAPS2 只取得印表机的资讯内容，使用者仍然可以从 DEVCAPS2 的功能表中选择所需印表机。如果使用者想比较不同印表机的功能，可以先用印表机档案夹增加各种列印驱动程式。

PrinterProperties 呼叫

DEVcaps2 的「Device」功能表中上还有一个称为「Properties」的选项。要使用这个选项，首先得从 **Device** 功能表中选择一个印表机，然後再选择 **Properties**，这时弹出一个对话方块。对话方块从何而来呢？它由印表机驱动程式呼叫，而且至少还让使用者选择纸的尺寸。大多数印表机驱动也可以让使用者在「直印 (portrait)」或「横印 (landscape)」模式中进行选择。在直印模式（一般为内定模式）下，纸的短边是顶部。在横印模式下，纸的长边是顶部。如果改变该模式，则所作的改变将在 DEVCAPS2 程式从 GetDeviceCaps 函式取得的资讯中反应出来：水平尺寸和解析度将与垂直尺寸和解析度交换。

彩色绘图机的「Properties」对话方块内容十分广泛，它们要求使用者输入安装在绘图机上之画笔的颜色和使用之绘图纸（或透明胶片）的型号。

所有印表机驱动程式都包含一个称为 ExtDeviceMode 的输出函式，它呼叫对话方块并储存使用者输入的资讯。有些印表机驱动程式也将这些资讯储存在系统登录的自己拥有的部分中，有些则不然。那些储存资讯的印表机驱动程式在下次执行 Windows 时将存取该资讯。

允许使用者选择印表机的 Windows 程式通常只呼叫 PrintDlg（本章後面我会展示用法）。这个有用的函式在准备列印时负责和使用者之间所有的通讯工作，并负责处理使用者要求的所有改变。当使用者单击「Properties」按钮时，PrintDlg 还会启动属性表格对话方块。

程式还可以通过直接呼叫印表机驱动程式的 ExtDeviceMode 或 ExtDeveModePropSheet 函式，来显示印表机的属性对话方块，然而，我不鼓励您这样做。像 DEVCAPS2 那样，透过呼叫 PrinterProperties 来启动对话方块会好得多。

PrinterProperties 要求印表机物件的代号，您可以通过 OpenPrinter 函式来得到。当使用者取消属性表格对话方块时，PrinterProperties 传回，然後使用者通过呼叫 ClosePrinter，释放印表机代号。DEVCAPS2 就是这样做到这一点的。

程式首先取得刚刚在 Device 功能表中选择的印表机名称，并将其存入一个名为 szDevice 的字元阵列中。

```
GetMenuString ( hMenu, nCurrentDevice, szDevice,
                sizeof (szDevice) / sizeof (TCHAR),
MF_BYCOMMAND) ;
```

然後，使用 OpenPrinter 获得该设备的代号。如果呼叫成功，那么程式接著呼叫 PrinterProperties 启动对话方块，然後呼叫 ClosePrinter 释放设备代号：

```
if (OpenPrinter (szDevice, &hPrint, NULL))
{
    PrinterProperties (hwnd, hPrint) ;
    ClosePrinter (hPrint) ;
}
```

检查 BitBlt 支援

您可以用 GetDeviceCaps 函式来取得页中可列印区的尺寸和解析度（通常，该区域不会与整张纸的大小相同）。如果使用者想自己进行缩放操作，也可以获得相对的图素宽度和高度。

印表机能力的大多数资讯是用于 GDI 而不是应用程式的。通常，在印表机不能做某件事时，GDI 会模拟出那项功能。然而，这是应用程式应该事先检查的。

以 RASTERCAPS (「位元映射支援」) 参数呼叫 GetDeviceCaps，它传回的 RC_BITBLT 位元包含了另一个重要的印表机特性，该位元标示设备是否能进行位元块传送。大多数点阵印表机、雷射印表机和喷墨印表机都能进行位元块传送，而大多数绘图机却不能。不能处理位元块传送的设备不支援下列 GDI 函式：CreateCompatibleDC、CreateCompatibleBitmap、PatBlt、BitBlt、StretchBlt、GrayString、DrawIcon、SetPixel、GetPixel、FloodFill、ExtFloodFill、FillRgn、FrameRgn、InvertRgn、PaintRgn、FillRect、FrameRect 和 InvertRect。这是在视讯显示器上使用 GDI 函式与在印表机上使用它们的唯一重要区别。

最简单的列印程式

现在可以开始列印了，我们尽可能简单地开始。事实上，我们的第一个程式只是让印表机送纸而已。程式 13-3 的 FORMFEED 程式，展示了列印所需的最小需求。

程式 13-3 FORMFEED

```
FORMFEED.C
/*-----
    FORMFEED.C --      Advances printer to next page
                                (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
HDC GetPrinterDC (void) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int iCmdShow)
{
    static DOCINFO di = { sizeof (DOCINFO), TEXT ("FormFeed") } ;
    HDC                hdcPrint = GetPrinterDC () ;

    if (hdcPrint != NULL)
    {
        if (StartDoc (hdcPrint, &di) > 0)
            if (StartPage (hdcPrint) > 0 && EndPage
(hdcPrint) > 0)
                EndDoc (hdcPrint) ;

        DeleteDC (hdcPrint) ;
    }
    return 0 ;
}
```

这个程式也需要前面程式 13-1 中的 GETPRNDC.C 档案。

除了取得印表机装置内容（然後再删除它）外，程式只呼叫了我们在本章前面讨论过的四个列印函式。FORMFEED 首先呼叫 StartDoc 开始一个新的档案，它测试从 StartDoc 传回的值，只有传回值是正数时，才继续下去：

```
if (StartDoc (hdcPrint, &di) > 0)
```

StartDoc 的第二个参数是指向 DOCINFO 结构的指标。该结构在第一个栏位包含了结构的大小，在第二个栏位包含了字串「FormFeed」。当档案正在被列印或者在等待列印时，这个字串将出现在印表机任务伫列中的「Document Name」列中。通常，该字串包含进行列印的应用程式名称和被列印的档案名称。

如果 StartDoc 成功（由一个正的传回值表示），那么 FORMFEED 呼叫 StartPage，紧接著立即呼叫 EndPage。这一程序将印表机推进到新的一页，再次对传回值进行测试：

```
if (StartPage (hdcPrint) > 0 && EndPage (hdcPrint) > 0)
```

最後，如果不出错，文件就结束：

```
EndDoc (hdcPrint) ;
```

要注意的是，只有当没出错时，才呼叫 EndDoc 函式。如果其他列印函式中的某一个传回错误代码，那么 GDI 实际上已经中断了文件的列印。如果印表机目前未列印，这种错误代码通常会使印表机重新设定。测试列印函式的传回值是检测错误的最简单方法。如果您想向使用者报告错误，就必须呼叫 GetLastError 来确定错误。

如果您写过 MS-DOS 下的简单利用印表机送纸的程式，就应该知道，对於大多数印表机，ASCII 码 12 启动送纸。为什么不简单地使用 C 的程式库函式 open，然後用 write 输出 ASCII 码 12 呢？当然，您完全可以这么做，但是必须确定印表机连结的是串列埠还是并列埠。然後您还要确定另外的程式（例如，列印伫列程式）是不是正在使用印表机。您并不希望在文件列印到一半时被别的程式把正在列印的那张纸送出印表机，对不对？最後，您还必须确定 ASCII 码 12 是不是所连结印表机的送纸字元，因为并非所有印表机的送纸字元都是 12。事实上，在 PostScript 中的送纸命令便不是 12，而是单字 showpage。

简单地说，不要试图直接绕过 Windows；而应该坚持在列印中使用 Windows 函式。

列印图形和文字

在一个 Windows 程式中，列印所需的额外负担通常比 FORMFEED 程式高得多，而且还要用 GDI 函式来实际列印一些东西。我们来写个列印一页文字和图形的程式，采用 FORMFEED 程式中的方法，并加入一些新的东西。该程式将有三个版本 PRINT1、PRINT2 和 PRINT3。为避免程式码重复，每个程式都用前面所示的

GETPRNDC.C 档案和 PRINT.C 档案中的函式，如程式 13-4 所示。

程式 13-4 PRINT

```

PRINT.C
/*-----
    PRINT.C -- Common routines for Print1, Print2, and Print3
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL PrintMyPage (HWND) ;

extern HINSTANCE hInst ;
extern TCHAR      szAppName[] ;
extern TCHAR      szCaption[] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hInst = hInstance ;
    hwnd = CreateWindow (szAppName, szCaption,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

```

```

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void PageGDIcalls (HDC hdcPrn, int cxPage, int cyPage)
{
    static TCHAR szTextStr[] = TEXT ("Hello, Printer!") ;
    Rectangle (hdcPrn, 0, 0, cxPage, cyPage) ;
    MoveToEx (hdcPrn, 0, 0, NULL) ;
    LineTo   (hdcPrn, cxPage, cyPage) ;
    MoveToEx (hdcPrn, cxPage, 0, NULL) ;
    LineTo   (hdcPrn, 0, cyPage) ;

    SaveDC (hdcPrn) ;

    SetMapMode          (hdcPrn, MM_ISOTROPIC) ;
    SetWindowExtEx      (hdcPrn, 1000, 1000, NULL) ;
    SetViewportExtEx    (hdcPrn, cxPage / 2, -cyPage / 2, NULL) ;
    SetViewportOrgEx    (hdcPrn, cxPage / 2, cyPage / 2, NULL) ;

    Ellipse (hdcPrn, -500, 500, 500, -500) ;
    SetTextAlign (hdcPrn, TA_BASELINE | TA_CENTER) ;
    TextOut (hdcPrn, 0, 0, szTextStr, lstrlen (szTextStr)) ;

    RestoreDC (hdcPrn, -1) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int          cxClient, cyClient ;
    HDC                hdc ;
    HMENU              hMenu ;
    PAINTSTRUCT        ps ;

    switch (message)
    {
    case WM_CREATE:
        hMenu = GetSystemMenu (hwnd, FALSE) ;
        AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;

```

```

        AppendMenu (hMenu, 0, 1, TEXT("&Print")) ;
        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_SYSCOMMAND:
        if (wParam == 1)
        {
            if (!PrintMyPage (hwnd))
                MessageBox (hwnd, TEXT ("Couldnotprint
page!"),
                            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            return 0 ;
        }
        break ;

    case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;

        PageGDI Calls (hdc, cxClient, cyClient) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

PRINT.C 包括函式 WinMain、WndProc 以及一个称为 PageGDI Calls 的函式。PageGDI Calls 函式接收印表机装置内容代号和两个包含列印页面宽度及高度的变数。这个函式还负责画一个包围整个页面的矩形，有两条对角线，页中间有一个椭圆（其直径是印表机高度和宽度中较小的那个的一半），文字「Hello, Printer!」位於椭圆的中间。

处理 WM_CREATE 讯息时，WndProc 将一个「Print」选项加到系统功能表上。选择该选项将呼叫 PrintMyPage，此函式的功能在程式的三个版本中将不断增强。当列印成功时，PrintMyPage 传回 TRUE 值，如果遇到错误时则传回 FALSE。如果 PrintMyPage 传回 FALSE，WndProc 就会显示一个讯息方块以告知使用者发生了错误。

列印的基本程序

列印程式的第一个版本是 PRINT1, 见程式 13-5。经编译后即可执行此程式, 然後从系统功能表中选择「Print」。接著, GDI 将必要的印表机输出储存在一个暂存档案中, 然後列印伫列程式将它发送给印表机。

程式 13-5 PRINT1

```
PRINT1.C
/*-----
    PRINT1.C -- Bare Bones Printing
                                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
HDC  GetPrinterDC (void) ;                // in GETPRNDC.C
void PageGDI Calls (HDC, int, int) ;      // in PRINT.C

HINSTANCE hInst ;
TCHAR      szAppName[] = TEXT ("Print1") ;
TCHAR      szCaption[] = TEXT ("Print Program 1") ;

BOOL PrintMyPage (HWND hwnd)
{
    static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print1: Printing") } ;
    BOOL          bSuccess = TRUE ;
    HDC           hdcPrn ;
    int           xPage, yPage ;

    if (NULL == (hdcPrn = GetPrinterDC ()))
        return FALSE ;
    xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
    yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

    if (StartDoc (hdcPrn, &di) > 0)
    {
        if (StartPage (hdcPrn) > 0)
        {
            PageGDI Calls (hdcPrn, xPage, yPage) ;

            if (EndPage (hdcPrn) > 0)
                EndDoc (hdcPrn) ;
            else
                bSuccess = FALSE ;
        }
    }
    else
        bSuccess = FALSE ;
}
```

```

DeleteDC (hdcPrn) ;
return bSuccess ;
}

```

我们来看看 PRINT1.C 中的程式码。如果 PrintMyPage 不能取得印表机的装置内容代号，它就传回 FALSE，并且 WndProc 显示讯息方块指出错误。如果函式成功取得了装置内容代号，它通过呼叫 GetDeviceCaps 来确定页面的水平和垂直大小（以图素为单位）。

```

xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

```

这不是纸的全部大小，只是纸的可列印区域。呼叫後，除了 PRINT1 在 StartPage 和 EndPage 呼叫之间呼叫 PageGDI Calls，PRINT1 的 PrintMyPage 函式中的程式码在结构上与 FORMFEED 中的程式码相同。仅当呼叫 StartDoc、StartPage 和 EndPage 都成功时，PRINT1 才呼叫 EndDoc 列印函式。

使用放弃程序来取消列印

对于大型文件，程式应该提供使用者在应用程式列印期间取消列印任务的便利性。也许使用者只要列印文件中的一页，而不是列印全部的 537 页。应该要能在印完全部的 537 页之前纠正这个错误。

在一个程式内取消一个列印任务需要一种被称为「放弃程序」的技术。放弃程序在程式中只是个较小的输出函式，使用者可以使用 SetAbortProc 函式将该函式的位址传给 Windows。然後 GDI 在列印时，重复呼叫该程序，不断地问：「我是否应该继续列印？」

我们看看将放弃程序加到列印处理程式中去需要些什么，然後检查一些旁枝末节。放弃程序一般命名为 AbortProc，其形式为：

```

BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
    //其他行程式
}

```

列印前，您必须通过呼叫 SetAbortProc 来登记放弃程序：

```

SetAbortProc (hdcPrn, AbortProc) ;

```

在呼叫 StartDoc 前呼叫上面的函式，列印完成後不必清除放弃程序。

在处理 EndPage 呼叫时（亦即，在将 metafile 放入装置驱动程序并建立临时列印档案时），GDI 常常呼叫放弃程序。参数 hdcPrn 是印表机装置内容代号。如果一切正常，iCode 参数是 0，如果 GDI 模组在生成暂存档案时耗尽了磁碟空间，iCode 就是 SP_OUTOFDISK。

如果列印作业继续，那么 AbortProc 必须传回 TRUE（非零）；如果列印作

业异常结束，就传回 FALSE（零）。放弃程序可以被简化为如下所示的形式：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
    MSG    msg ;

    while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return TRUE ;
}
```

这个函式看起来有点特殊，其实它看起来像是讯息回圈。使用者会注意到，这个「讯息回圈」呼叫 PeekMessage 而不是 GetMessage。我在第五章的 RANDRECT 程式中讨论过 PeekMessage。应该还记得，PeekMessage 将会控制权返回给程式，而不管程式的讯息伫列中是否有讯息存在。

只要 PeekMessage 传回 TRUE，那么 AbortProc 函式中的讯息回圈就重复呼叫 PeekMessage。TRUE 值表示 PeekMessage 已经找到一个讯息，该讯息可以通过 TranslateMessage 和 DispatchMessage 发送到程式的视窗讯息处理程式。若程式的讯息伫列中没有讯息，则 PeekMessage 的传回值为 FALSE，因此 AbortProc 将控制权返回给 Windows。

Windows 如何使用 AbortProc

当程式进行列印时，大部分工作发生在要呼叫 EndPage 时。呼叫 EndPage 前，程式每呼叫一次 GDI 绘图函式，GDI 模组只是简单地将另一个记录加到磁片上的 metafile 中。当 GDI 得到 EndPage 後，对列印页中由装置驱动程式定义每个输出带，GDI 都将该 metafile 送入装置驱动程式中。然後，GDI 将印表机驱动程式建立的列印输出储存到一个档案中。如果没有启用幕後列印，那么 GDI 模组必须自动将该列印输出写入印表机。

在 EndPage 呼叫期间，GDI 模组呼叫您设定的放弃程序。通常 iCode 参数为 0，但如果由於存在未列印的其他暂存档案，而造成 GDI 执行时磁碟空间不够，iCode 参数就为 SP_OUTOFDISK（通常您不会检查这个值，但是如果愿意，您可以进行检查）。放弃程序隨後进入 PeekMessage 回圈从自己的讯息伫列中找寻讯息。

如果在程式的讯息伫列中没有讯息，PeekMessage 会传回 FALSE，然後放弃程序跳出它的讯息回圈并给 GDI 模组传回一个 TRUE 值，指示列印应该继续进行。然後 GDI 模组继续处理 EndPage 呼叫。

如果有错误发生，那么 GDI 将中止列印程序，这样，放弃程序的主要目的是允许使用者取消列印。为此，我们还需要一个显示「Cancel」按钮的对话方块，让我们采用两个独立的步骤。首先，我们在建立 PRINT2 程式时增加一个放弃程序，然後在 PRINT3 中增加一个带有「Cancel」按钮的对话方块，使放弃程序可用。

实作放弃程序

现在快速复习一下放弃程序的机制。可以定义一个如下所示的放弃程序：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
    MSG msg ;
    while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return TRUE ;
}
```

当您想列印什么时，使用下面的呼叫将指向放弃程序的指标传给 Windows：

```
SetAbortProc (hdcPrn, AbortProc) ;
```

在呼叫 StartDoc 之前进行这个呼叫就行了。

不过，事情没有这么简单。我们忽视了 AbortProc 程序中 PeekMessage 回圈这个问题，它是个很大的问题。只有在程式处於列印程序时，AbortProc 程序才会被呼叫。如果在 AbortProc 中找到一个讯息并把它传送给视窗讯息处理程式，就会发生一些非常令人讨厌的事情：使用者可以从功能表中再次选择「Print」，但程式已经处於列印常式之中。程式在列印前一个档案的同时，使用者也可以把一个新档案载入到程式里。使用者甚至可以退出程式！如果这种情况发生了，所有使用者程式的视窗都将被清除。当列印常式执行结束时，除了退到不再有效的视窗常式之外，您无处可去。

这种东西会把人搞得晕头转向，而我们的程式对此并未做任何准备。正是由於这个原因，当设定放弃程序时，首先应禁止程式的视窗接受输入，使它不能接受键盘和滑鼠输入。可以用以下的函式完成这项工作：

```
EnableWindow (hwnd, FALSE) ;
```

它可以禁止键盘和滑鼠的输入进入讯息佇列。因此在列印程序中，使用者不能对程式做任何工作。当列印完成时，应重新允许视窗接受输入：

```
EnableWindow (hwnd, TRUE) ;
```

您可能要问，既然没有键盘或滑鼠讯息进入讯息佇列，为什么我们还要进行 AbortProc 中的 TranslateMessage 和 DispatchMessage 呼叫呢？实际上并不

一定非得需要 TranslateMessage，但是，我们必须使用 DispatchMessage，处理 WM_PAINT 讯息进入讯息伫列中的情况。如果 WM_PAINT 讯息没有得到视窗讯息处理程式中的 BeginPaint 和 EndPaint 的适当处理，由於 PeekMessage 不再传回 FALSE，该讯息就会滞留在伫列中并且妨碍工作。

当列印期间阻止视窗处理输入讯息时，您的程式不会进行显示输出。但使用者可以切换到其他程式，并在那里进行其他工作，而幕後列印程式则能继续将输出档案送到印表机。

程式 13-6 所示的 PRINT2 程式在 PRINT1 中增加了一个放弃程序和必要的支援——呼叫 AbortProc 函式并呼叫 EnableWindow 两次（第一次阻止视窗接受输入讯息，第二次启用视窗）。

程式 13-6 PRINT2

```
PRINT2.C
/*-----
    PRINT2.C -- Printing with Abort Procedure
                                     (c) Charles Petzold, 1998
-----*/

#include <windows.h>
HDC  GetPrinterDC (void) ;           // in GETPRNDC.C
void PageGDI Calls (HDC, int, int) ; // in PRINT.C

HINSTANCE hInst ;
TCHAR      szAppName[] = TEXT ("Print2") ;
TCHAR      szCaption[] = TEXT ("Print Program 2 (Abort Procedure)") ;

BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
    MSG msg ;
    while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return TRUE ;
}

BOOL PrintMyPage (HWND hwnd)
{
    static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print2: Printing") } ;
    BOOL          bSuccess = TRUE ;
    HDC           hdcPrn ;
    short         xPage, yPage ;

    if (NULL == (hdcPrn = GetPrinterDC ()))
```

```

        return FALSE ;
xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

EnableWindow (hwnd, FALSE) ;
SetAbortProc (hdcPrn, AbortProc) ;
if (StartDoc (hdcPrn, &di) > 0)
{
    if (StartPage (hdcPrn) > 0)
    {
        PageGDI Calls (hdcPrn, xPage, yPage) ;
        if (EndPage (hdcPrn) > 0)
            EndDoc (hdcPrn) ;
        else
            bSuccess = FALSE ;
    }
}
else
    bSuccess = FALSE ;
EnableWindow (hwnd, TRUE) ;
DeleteDC (hdcPrn) ;
return bSuccess ;
}

```

增加列印对话方块

PRINT2 还不能令人十分满意。首先，这个程式没有直接指示出何时开始列印和何时结束列印。只有将滑鼠指向程式并且发现它没有反应时，才能断定它仍然在处理 PrintMyPage 常式。PRINT2 在进行背景处理时也没有给使用者提供取消列印作业的机会。

您可能注意到，大多数 Windows 程式都为使用者提供了一个取消目前正在进行列印操作的机会。一个小的对话方块出现在萤幕上，它包括一些文字和「Cancel」按键。在 GDI 将列印输出储存到磁片档案或（如果停用列印伫列程式）印表机正在列印的整个期间，程式都显示这个对话方块。它是一个非系统模态对话方块，您必须提供对话程序。

通常称这个对话方块为「放弃对话方块」，称这种对话程序为「放弃对话程序」。为了更清楚地把它和「放弃程序」区别开来，我们称这种对话程序为「列印对话程序」。放弃程序（名为 AbortProc）和列印对话程序（将命名为 PrintDlgProc）是两个不同的输出函式。如果想以一种专业的 Windows 式列印方式进行列印工作，就必须拥有这两个函式。

这两个函式的交互作用方式如下：AbortProc 中的 PeekMessage 回圈得被修改，以便将非系统模态对话方块的讯息发送给对话方块视窗讯息处理程式。

PrintDlgProc 必须处理 WM_COMMAND 讯息，以检查「Cancel」按钮的状态。如果「Cancel」按钮被按下，就将一个叫做 bUserAbort 的整体变数设为 TRUE。AbortProc 传回的值正好和 bUserAbort 相反。您可能还记得，如果 AbortProc 传回 TRUE 会继续列印，传回 FALSE 则放弃列印。在 PRINT2 中，我们总是传回 TRUE。现在，使用者在列印对话方块中按下「Cancel」按钮时将传回 FALSE。程式 13-7 所示的 PRINT3 程式实作了这个处理方式。

程式 13-7 PRINT3

```
PRINT3.C
/*-----
    PRINT3.C -- Printing with Dialog Box
                                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
HDC  GetPrinterDC (void) ;                // in GETPRNDC.C
void PageGDI Calls (HDC, int, int) ;      // in PRINT.C

HINSTANCE hInst ;
TCHAR      szAppName[] = TEXT ("Print3") ;
TCHAR      szCaption[] = TEXT ("Print Program 3 (Dialog Box)") ;

BOOL bUserAbort ;
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc (HWND hDlg, UINT message,
                           WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            SetWindowText (hDlg, szAppName) ;
            EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE,
MF_GRAYED) ;

            return TRUE ;

        case WM_COMMAND:
            bUserAbort = TRUE ;
            EnableWindow (GetParent (hDlg), TRUE) ;
            DestroyWindow (hDlg) ;
            hDlgPrint = NULL ;
            return TRUE ;

    }
    return FALSE ;
}
```

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
    MSG msg ;
    while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return !bUserAbort ;
}

BOOL PrintMyPage (HWND hwnd)
{
    static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print3: Printing") } ;
    BOOL bSuccess = TRUE ;
    HDC hdcPrn ;
    int xPage, yPage ;

    if (NULL == (hdcPrn = GetPrinterDC ()))
        return FALSE ;
    xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
    yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

    EnableWindow (hwnd, FALSE) ;
    bUserAbort = FALSE ;
    hDlgPrint = CreateDialog (hInst, TEXT ("PrintDlgBox"),
                                hwnd,
PrintDlgProc) ;
    SetAbortProc (hdcPrn, AbortProc) ;
    if (StartDoc (hdcPrn, &di) > 0)
    {
        if (StartPage (hdcPrn) > 0)
        {
            PageGDI Calls (hdcPrn, xPage, yPage) ;
            if (EndPage (hdcPrn) > 0)
                EndDoc (hdcPrn) ;
            else
                bSuccess = FALSE ;
        }
    }
    else
        bSuccess = FALSE ;
    if (!bUserAbort)
    {
        EnableWindow (hwnd, TRUE) ;
    }
}
```



```

        DestroyWindow (hDlgPrint) ;
    }

    DeleteDC (hdcPrn) ;
    return bSuccess && !bUserAbort ;
}

PRINT.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
PRINTDLGBOX DIALOG DISCARDABLE 20, 20, 186, 63
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON                "Cancel", IDCANCEL, 67, 42, 50, 14
    CTEXT                      "Cancel
Printing", IDC_STATIC, 7, 21, 172, 8
END

```

如果您使用 PRINT3, 那么最好临时暂停使用幕後列印; 否则, 只有在列印
 伫列程式从 PRINT3 中接收资料时才可见到的「Cancel」按钮可能会很快消失,
 让您根本没有机会去按它。如果您按「Cancel」按钮时列印并不立即终止 (特
 别是在一个慢速印表机上), 不要惊讶。印表机有一个内部缓冲区, 在印表机
 停止之前其中的资料必须全部送出, 按「Cancel」只是告诉 GDI 不要向印表机
 的缓冲区发送更多的资料而已。

PRINT3 增加了两个整体变数: 一个是叫做 bUserAbort 的布林变数, 另一个
 是叫做 hDlgPrint 的对话方块视窗代号。PrintMyPage 函式将 bUserAbort 初始
 化为 FALSE。与 PRINT2 一样, 程式的主视窗是不接收输入讯息的。指向 AbortProc
 的指标用於 SetAbortProc 呼叫中, 而指向 PrintDlgProc 的指标用於
 CreateDialog 呼叫中。CreateDialog 传回的视窗代号储存在 hDlgPrint 中。

现在, AbortProc 中的讯息回圈如下:

```

while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
{
    if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return !bUserAbort ;

```

只有在 `bUserAbort` 为 `FALSE`，也就是使用者还没有终止列印工作时，这段程式码才会呼叫 `PeekMessage`。`IsDialogMessage` 函式用来将讯息发送给非系统模态对话方块。和普通的非系统模态对话方块一样，对话方块视窗的代号在这个呼叫之前受到检查。`AbortProc` 的传回值正好与 `bUserAbort` 相反。开始时，`bUserAbort` 为 `FALSE`，因此 `AbortProc` 传回 `TRUE`，表示继续进行列印；但是 `bUserAbort` 可能在列印对话程序中被设定为 `TRUE`。

`PrintDlgProc` 函式是相当简单的。处理 `WM_INITDIALOG` 时，该函式将视窗标题设定为程式名称，并且停用系统功能表上的「Close」选项。如果使用者按下了「Cancel」钮，`PrintDlgProc` 将收到 `WM_COMMAND` 讯息：

```
case WM_COMMAND :
    bUserAbort = TRUE ;
    EnableWindow (GetParent (hDlg), TRUE) ;
    DestroyWindow (hDlg) ;
    hDlgPrint = NULL ;
    return TRUE ;
```

将 `bUserAbort` 设定为 `TRUE`，则说明使用者已经决定取消列印操作，主视窗被启动，而对话方块被清除（按顺序完成这两项活动是很重要的，否则，在 Windows 中执行其他程式之一将变成活动程式，而您的程式将消失到背景中）。与通常的情况一样，将 `hDlgPrint` 设定为 `NULL`，防止在讯息回圈中呼叫 `IsDialogMessage`。

只有在 `AbortProc` 用 `PeekMessage` 找到讯息，并用 `IsDialogMessage` 将它们传送给对话方块视窗讯息处理程式时，这个对话方块才接收讯息。只有在 GDI 模组处理 `EndPage` 函式时，才呼叫 `AbortProc`。如果 GDI 发现 `AbortProc` 的传回值是 `FALSE`，它将控制权从 `EndPage` 传回到 `PrintMyPage`。它不传回错误码。至此，`PrintMyPage` 认为列印页已经发完了，并呼叫 `EndDoc` 函式。但是，由於 GDI 模组还没有完成对 `EndPage` 呼叫的处理，所以不会列印出什么东西来。

有些清除工作尚待完成。如果使用者没在对话方块中取消列印作业，那么对话方块仍然会显示著。`PrintMyPage` 重新启用它的主视窗并清除对话方块：

```
if (!bUserAbort)
{
    EnableWindow (hwnd, TRUE) ;
    DestroyWindow (hDlgPrint) ;
}
```

两个变数会通知您发生了什么事：`bUserAbort` 可以告诉您使用者是否终止了列印作业，`bSuccess` 会告诉您是否出了故障，您可以用这些变数来完成想做的工作。`PrintMyPage` 只简单地对它们进行逻辑上的 AND 运算，然後把值传回给 `WndProc`：

```
return bSuccess && !bUserAbort ;
```

为 POPPAD 增加列印功能

现在准备在 POPPAD 程式中增加列印功能，并且宣布 POPPAD 已告完毕。这需要第十一章中的各个 POPPAD 档案，此外，还需要程式 13-8 中的 POPPRNT.C 档案。

程式 13-8 POPPRNT

```
POPPRNT.C
/*-----
    POPPRNT.C -- Popup Editor Printing Functions
-----*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"

BOOL bUserAbort ;
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc ( HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
        case WM_INITDIALOG :
            EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE,
MF_GRAYED) ;

            return TRUE ;

        case WM_COMMAND :
            bUserAbort = TRUE ;
            EnableWindow (GetParent (hDlg), TRUE) ;
            DestroyWindow (hDlg) ;
            hDlgPrint = NULL ;
            return TRUE ;

    }
    return FALSE ;
}

BOOL CALLBACK AbortProc (HDC hPrinterDC, int iCode)
{
    MSG msg ;
    while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
    {
        if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
        {

```

```

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

    }

}

return !bUserAbort ;
}

BOOL PopPrntPrintFile (HINSTANCE hInst, HWND hwnd, HWND hwndEdit,
                        PTSTR
szTitleName)
{
    static DOCINFO    di = { sizeof (DOCINFO) } ;
    static PRINTDLG    pd ;
    BOOL              bSuccess ;
    int                yChar, iCharsPerLine, iLinesPerPage, iTotalLines,
                        iTotalPages, iPage, iLine, iLineNum ;
    PTSTR              pstrBuffer ;
    TCHAR              szJobName [64 + MAX_PATH] ;
    TEXTMETRIC         tm ;
    WORD               iColCopy, iNoiColCopy ;

    // Invoke Print common dialog box

    pd.lStructSize      =    sizeof (PRINTDLG) ;
    pd.hwndOwner         =    hwnd ;
    pd.hDevMode          =    NULL ;
    pd.hDevNames         =    NULL ;
    pd.hDC              =    NULL ;
    pd.Flags             =    PD_ALLPAGES | PD_COLLATE |
        PD_RETURNDC | PD_NOSELECTION ;
    pd.nFromPage         =    0 ;
    pd.nToPage          =    0 ;
    pd.nMinPage         =    0 ;
    pd.nMaxPage         =    0 ;
    pd.nCopies           =    1 ;
    pd.hInstance        =    NULL ;
    pd.lCustData         =    0L ;
    pd.lpfPrintHook      =    NULL ;
    pd.lpfSetupHook     =    NULL ;
    pd.lpPrintTemplateName =    NULL ;
    pd.lpSetupTemplateName =    NULL ;
    pd.hPrintTemplate    =    NULL ;
    pd.hSetupTemplate    =    NULL ;

    if    (!PrintDlg (&pd))
        return TRUE ;

    if    (0 == (iTotalLines = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0)))

```

```

        return TRUE ;

        // Calculate necessary metrics for file

    GetTextMetrics (pd.hDC, &tm) ;
    yChar = tm.tmHeight + tm.tmExternalLeading ;

    iCharsPerLine = GetDeviceCaps (pd.hDC, HORZRES) / tm.tmAveCharWidth ;
    iLinesPerPage = GetDeviceCaps (pd.hDC, VERTRES) / yChar ;
    iTotalPages    = (iTotalLines + iLinesPerPage - 1) / iLinesPerPage ;

        // Allocate a buffer for each line of text

    pstrBuffer = malloc (sizeof (TCHAR) * (iCharsPerLine + 1)) ;

        // Display the printing dialog box

    EnableWindow (hwnd, FALSE) ;
    bSuccess      = TRUE ;
    bUserAbort    = FALSE ;

    hDlgPrint     = CreateDialog (hInst, TEXT ("PrintDlgBox"),
                                hwnd,
PrintDlgProc) ;

    SetDlgItemText (hDlgPrint, IDC_FILENAME, szTitleName) ;
    SetAbortProc (pd.hDC, AbortProc) ;

        // Start the document

    GetWindowText (hwnd, szJobName, sizeof (szJobName)) ;
    di.lpszDocName = szJobName ;
    if (StartDoc (pd.hDC, &di) > 0)
    {
        // Collation requires this loop and
iNoiColCopy
        for (iColCopy = 0 ;
            iColCopy < ((WORD) pd.Flags & PD_COLLATE ?
pd.nCopies : 1) ;
            iColCopy++)
        {
            for (iPage = 0 ; iPage < iTotalPages ; iPage++)
            {
                for (iNoiColCopy = 0 ;
                    iNoiColCopy < (pd.Flags & PD_COLLATE ? 1 : pd.nCopies);
                    iNoiColCopy++)
                {
                    // Start the page
                    if (StartPage (pd.hDC) < 0)

```

```

        {
            bSuccess =
FALSE ;
            break ;
        }

    // For each page, print the lines
    for (iLine = 0 ; iLine < iLinesPerPage ; iLine++)
    {
        iLineNum = iLinesPerPage * iPage + iLine ;
        if (iLineNum > iTotalLines)
            break ;

        *(int *) pstrBuffer = iCharsPerLine ;
        TextOut (pd.hDC, 0, yChar * iLine, pstrBuffer,
        (int) SendMessage (hwndEdit, EM_GETLINE,
            (LPARAM) iLineNum, (LPARAM) pstrBuffer));
    }

    if (EndPage (pd.hDC) < 0)
    {
        bSuccess = FALSE ;
        break ;
    }

    if (bUserAbort)
        break ;
    }

    if (!bSuccess || bUserAbort)
        break ;
    }

    if (!bSuccess || bUserAbort)
        break ;
    }
}
else
    bSuccess = FALSE ;
if (bSuccess)
    EndDoc (pd.hDC) ;

if (!bUserAbort)
{
    EnableWindow (hwnd, TRUE) ;
    DestroyWindow (hDlgPrint) ;
}

free (pstrBuffer) ;

```

```
DeleteDC (pd.hDC) ;  
  
return bSuccess && !bUserAbort ;  
}
```

与 POPPAD 尽量利用 Windows 高阶功能来简化程式的方针一致, POPPRNT.C 档案展示了使用 PrintDlg 函式的方法。这个函式包含在通用对话方块程式库 (common dialog box library) 中, 使用一个 PRINTDLG 型态的结构。

通常, 程式的「File」功能表中有个「Print」选项。当使用者选中「Print」选项时, 程式可以初始化 PRINTDLG 结构的栏位, 并呼叫 PrintDlg。

PrintDlg 显示一个对话方块, 它允许使用者选择列印页的范围。因此, 这个对话方块特别适用於像 POPPAD 这样能列印多页文件的程式。这种对话方块同时也给出了一个确定副本份数的编辑区和名为「Collate (逐份列印)」的核取方块。「逐份列印」影响著多个副本页的顺序。例如, 如果文件是 3 页, 使用者要求列印三份副本, 则这个程式能以两种顺序之一列印它们。选择逐份列印後的副本的页码顺序为 1、2、3、1、2、3、1、2、3, 未选择逐份列印的副本的页码顺序是 1、1、1、2、2、2、3、3、3。程式在这里应负起的责任就是以正确的顺序列印副本。

这个对话方块也允许使用者选择非内定印表机, 它包括一个标记为「Properties」的按钮, 可以启动设备模式对话方块。这样, 至少允许使用者选择直印或横印。

从 PrintDlg 函式传回後, PRINTDLG 结构的栏位指明列印页的范围和是否对多个副本进行逐份列印。这个结构同时也给出了准备使用的印表机装置内容代号。

在 POPPRNT.C 中, PopPrntPrintFile 函式 (当使用者在「File」功能表里选中「Print」选项时, 它由 POPPAD 呼叫) 呼叫 PrintDlg, 然後开始列印档案。PopPrntPrintFile 完成某些计算, 以确定一行能容纳多少字元和一页能容纳多少行。这个程序涉及到呼叫 GetDeviceCaps 来确定页的解析度, 呼叫 GetTextMetrics 来确定字元的大小。

这个程式通过发送一条 EM_GETLINECOUNT 讯息给编辑控制项来取得文件中的总行数 (在变数 iTotallines 中)。储存各行内容的缓冲区配置在局部记忆体中。对每一行, 缓冲区的第一个字被设定为该行中字元的数目。把 EM_GETLINE 讯息发送给编辑控制项可以把一行复制到缓冲区中, 然後用 TextOut 把这一行送到印表机装置内容中 (POPPRNT.C 还没有聪明到对超出列印宽度的文字换到下一行去处理。在第十七章我们会讨论这种文字绕行的技术)。

为了确定副本份数, 应注意列印文字的处理方式包括两个 for 回圈。第一

个 for 回圈使用了一个叫作 iColCopy 的变数,当使用者指定将副本逐份列印时,它将会起作用。第二个 for 回圈使用了一个叫作 iNonColCopy 的变数,当不对副本进行逐份列印时,它将起作用。

如果 StartPage 或 EndPage 传回一个错误,或者如果 bUserAbort 为 TRUE,那么这个程式退出增加页号的那个 for 回圈。如果放弃程序的传回值是 FALSE,则 EndPage 不传回错误。正是由於这个原因,在下一页开始之前,要直接测试 bUserAbort。如果没有报告错误,则进行 EndDoc 呼叫:

```
if (!bError)
    EndDoc (hdcPrn) ;
```

您可能想通过列印多页档案来测试 POPPAD。您可以从列印任务视窗中监视列印进展情况。在 GDI 处理完第一个 EndPage 呼叫之後,首先列印的档案将显示在列印任务视窗中。此时,幕後列印程式开始把档案发送到印表机。然後,如果在 POPPAD 中取消列印作业,那么幕後列印程式将终止列印,这也就是放弃程序传回 FALSE 的结果。当档案出现在列印任务视窗中,您也可以透过从「Document」功能表中选择「Cancel Printing」来取消列印作业,在这种情况下,POPPAD 中的 EndPage 呼叫会传回一个错误。

Windows 的程式设计的新手经常会抱住 AbortDoc 函式不放,但实际上这个函式几乎不在列印中使用。像在 POPPAD 中看到的那样,使用者几乎随时可以取消列印作业,或者通过 POPPAD 的列印对话方块及通过列印任务视窗。这两种方法都不需要程式使用 AbortDoc 函式。POPPAD 中允许 AbortDoc 的唯一时刻是在对 StartDoc 的呼叫和对 EndPage 的第一个呼叫之间,但是程式很快就会执行过去,以至不再需要 AbortDoc。

图 13-3 显示出正确列印多页文件之列印函式的呼叫顺序。检查 bUserAbort 的值是否为 TRUE 的最佳位置是在每个 EndPage 函式之後。只有当对先前的列印函式的呼叫没有产生错误时,才使用 EndDoc 函式。实际上,如果任何一个列印函式的呼叫出现错误,那么表演就结束了,同时您也可以回家了。

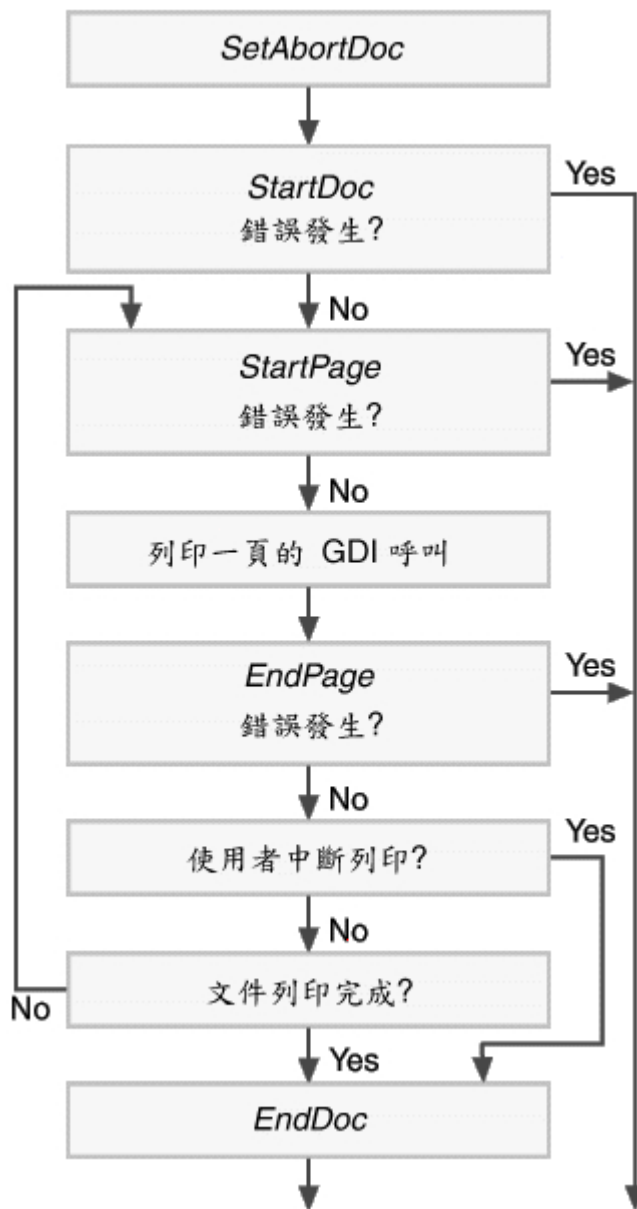


图 13-3 列印一个文件时的函式呼叫顺序

第十四章 点阵图和 Bitblt

点阵图是一个二维的位元阵列，它与图像的图素一一对应。当现实世界的图像被扫描成点阵图以后，图像被分割成网格，并以图素作为取样单位。在点阵图中的每个图素值指明了一个单位网格内图像的平均颜色。单色点阵图每个图素只需要一位元，灰色或彩色点阵图中每个图素需要多个位元。

点阵图代表了 Windows 程式内储存图像资讯的两种方法之一。储存图像资讯的另一种形式是 metafile，我将在第十八章讨论。Metafile 储存的就是对图像如何生成的描述，而不是将图像以数位化的图示代表。

以后我将更详细地讨论，Microsoft Windows 3.0 定义了一种称为装置无关点阵图 (DIB: device-independent bitmap)。我将在下一章讨论 DIB。本章主要讨论 GDI 点阵图物件，这是一种在 Windows 中比 DIB 更早支援的点阵图形资料。如同本章大量的范例程式所说明的，这种比 DIB 点阵图更早被 Windows 支援的图形格式仍然有其利用价值。

点阵图入门

点阵图和 metafile 在电脑图形处理世界中都占有一席之地。点阵图经常用来表示来自真实世界的复杂图像，例如数位化的照片或者视讯图像。Metafile 更适合於描述由人或者机器产生的图像，比如建筑蓝图。点阵图和 metafile 都能存於记忆体或作为档案存於磁片上，并且都能通过剪贴簿在 Windows 应用程式之间传输。

点阵图和 metafile 的区别在於位元映射图像和向量图像之间的差别。位元映射图像用离散的图素来处理输出设备；而向量图像用笛卡尔座标系统来处理输出设备，其线条和填充物件能被个别拖移。现在大多数的图像输出设备是位元映射设备，这包括视讯显示、点阵印表机、雷射印表机和喷墨印表机。而笔式绘图机则是向量输出设备。

点阵图有两个主要的缺点。第一个问题是容易受装置依赖性的影响。最明显的就是对颜色的依赖性，在单色设备上显示彩色点阵图的效果总是不能令人满意的。另一个问题是点阵图经常暗示了特定的显示解析度和图像纵横比。尽管点阵图能被拉伸和缩小，但是这样的处理通常包括复制或删除图素的某些行和列，这样会破坏图像的大小。而 metafile 在放大缩小后仍然能保持图形样貌不受破坏。

点阵图的第二个缺点是需要很大的储存空间。例如，描述完整的 640×480

图素，16 色的视频图形阵列 (VGA: Video Graphics Array) 萤幕的一幅点阵图需要大於 150 KB 的空间；一幅 1024×768，并且每个图素为 24 位元颜色的图像则需要大於 2 MB 的空间。Metafile 需要通常比点阵图来得少的空间。点阵图的储存空间由图像的大小及其包含的颜色决定，而 metafile 的储存空间则由图像的复杂程度和它所包含的 GDI 指令数决定。

然而，点阵图优於 metafile 之处在于速度。将点阵图复制给视讯显示器通常比复制基本图形档案的速度要快。最近几年，压缩技术允许压缩点阵图的档案大小，以使它能有效地通过电话线传输并广泛地用於 Internet 的网页上。

点阵图的来源

点阵图可以手工建立，例如，使用 Windows 98 附带的「小画家」程式。一些人宁愿使用位元映射绘图软体也不使用向量绘图软体。他们假定：图形最後一定会复杂到不能用线条跟填充区域来表达。

点阵图图像也能由电脑程式计算生成。尽管大多数计算生成的图像能按向量图形 metafile 储存，但是高清晰度的画面或碎形图样通常还是需要点阵图。

现在，点阵图通常用於描述真实世界的图像，并且有许多硬体设备能让您把现实世界的图像输入到电脑。这类硬体通常使用 **电荷耦合装置** (CCD: charge-coupled device)，这种装置接触到光就释放电荷。有时这些 CCD 单元能排列成一组，一个图素对应一个 CCD；为节约开支，只用一行 CCD 扫描图像。

在这些电脑 CCD 设备中，**扫描器** 是最古老的。它用一行 CCD 沿著纸上图像（例如照片）的表面扫描。CCD 根据光的强度产生电荷。类比数位转换器 (ADC: Analog-to-digital converters) 把电荷转换为数位讯号，然後排列成点阵图。

携带型摄像机也利用 CCD 单元组来捕捉影像。通常，这些影像是记录到录影带上。不过，这些视讯输出也能直接进入 **影像捕捉器** (frame grabber)，该装置能把类比视讯信号转换为一组图素值。这些影像捕捉器与任何相容的视讯信号来源都能同时使用，例如 VCR、光碟、DVD 播放机或有线电视解码器。

最近，数位照相机的价位对于家庭使用者来说开始变得负担得起了。它看起来很像普通照相机。但是数位照相机不使用底片，而用一组 CCD 来拦截图像，并且在 ADC 内部把数位图像直接储存在照相机内的记忆体中。通常，数位照相机与电脑的介面要通过序列埠。

点阵图尺寸

点阵图呈矩形，并有空间尺寸，图像的高度和宽度都以图素为单位。例如，此网格可描述一个很小的点阵图：宽度为 9 图素，高度为 6 图素，或者更简单

地计为 9×6 :

	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4									
5									

习惯上，点阵图的速记尺寸是先给出宽度。点阵图总数为 9×6 或者 54 图素。我将经常使用符号 cx 和 cy 来表示点阵图的宽度和高度。 c 表示计数，因此 cx 和 cy 是沿著 x 轴（水平）和 y 轴（垂直）的图素数。

我们能根据 x 和 y 座标来描述点阵图上具体的图素。一般（并不都是这样），在网格内计算图素时，点阵图开始於图像的左上角。这样，在此点阵图右下角的图素座标就是 $(8, 5)$ 。因为从 0 开始计数，所以此值比图像的宽度和高度小 1。

点阵图的空间尺寸通常也指定了解析度，但这是一个有争议的词。我们说我们的视讯显示有 640×480 的解析度，但是雷射印表机的解析度只有每英寸 300 点。我喜欢用後一种情况中解析度的意思作为每单位图素的数量。点阵图在这种意义上的解析度指的是点阵图在特定测量单位中的图素数。不管怎样，当我使用解析度这个词语时，其定义的内容应该是明确的。

点阵图是矩形的，但是电脑记忆体空间是线性的。通常（但并不都是这样）点阵图按列储存在记忆体中，且从顶列图素开始到底列结束。（DIB 是此规则的一个主要例外）。每一列，图素都从最左边的图素开始依次向右储存。这就好像储存几列文字中的各个字元。

颜色和点阵图

除空间尺寸以外，点阵图还有颜色尺寸。这里指的是每个图素所需要的位元数，有时也称为点阵图的 **颜色深度**（color depth）、**位元数**（bit-count）或 **位元/图素**（bpp: bits per pixel）数。点阵图中的每个图素都有相同数量的颜色位元。

每图素 1 位元的点阵图称为 **二阶**（bilevel）、**二色**（bicolor）或者 **单色**（monochrome）点阵图。每图素可以是 0 或 1，0 表示黑色，1 可以表示白色，但并不总是这样。对于其他颜色，一个图素就需要有多个位元。可能的颜色值等於 2 位元数值。用 2 位元可以得到 4 种颜色，用 4 位元可以得到 16 种颜色，8 位元可得到 256 种颜色，16 位元可得到 65,536 种颜色，而 24 位元可得到 16,777,216 种颜色。

如何将颜色位元的组合与人们所熟悉的颜色相对应是目前处理点阵图时经常碰到（而且常常是灾难）的问题。

实际的设备

点阵图可按其颜色位元数来分类；在 Windows 的发展过程中，不同的点阵图颜色格式取决於常用视讯显示卡的功能。实际上，我们可把视讯显示记忆体看作是一幅巨大的点阵图——我们从显示器上就可以看见。

Windows 1.0 多数采用的显示卡是 IBM 的彩色图像适配器（CGA: Color Graphics Adapter）和单色图形卡（HGC: Hercules Graphics Card）。HGC 是单色设备，而 CGA 也只能在 Windows 以单色图形模式使用。单色点阵图现在还很常用（例如，滑鼠的游标一般为单色），而且单色点阵图除显示图像以外还有其他用途。

随著增强型图形显示卡（EGA: Enhanced Graphics Adapter）的出现，Windows 使用者开始接触 16 色的图形。每个图素需要 4 个颜色位元。（实际上，EGA 比这里所讲的更复杂，它还包括一个 64 种颜色的调色盘，应用程式可以从中选择任意的 16 种颜色，但 Windows 只按较简单的方法使用 EGA）。在 EGA 中使用的 16 种颜色是黑、白、两种灰色、高低亮度的红色、绿和蓝（三原色）、青色（蓝和绿组合的颜色）。现在认为这 16 种颜色是 Windows 的最低颜色标准。同样，其他 16 色点阵图也可以在 Windows 中显示。大多数的图示都是 16 色的点阵图。通常，简单的卡通图像也可以用这 16 种颜色制作。

在 16 色点阵图中的颜色编码有时称为 IRGB（高亮红绿蓝：Intensity-Red-Green-Blue），并且实际上是源自 IBM CGA 文字模式下最初使用的十六种颜色。每个图素所用的 4 个 IRGB 颜色位元都映射为表 14-1 所示的 Windows 十六进位 RGB 颜色。

表 14-1

IRGB	RGB 颜色	颜色名称
0000	00-00-00	黑
0001	00-00-80	暗蓝
0010	00-80-00	暗绿
0011	00-80-80	暗青
0100	80-00-00	暗红
0101	80-00-80	暗洋红
0110	80-80-00	暗黄
0111	C0-C0-C0	亮灰

1000	80-80-80	暗灰
1001	00-00-FF	蓝
1010	00-FF-00	绿
1011	00-FF-FF	青
1100	FF-00-00	红
1101	FF-00-FF	洋红
1110	FF-FF-00	黄
1111	FF-FF-FF	白

EGA 的记忆体组成了四个「颜色面」，也就是说，定义每个图素颜色的四位元在记忆体中是不连续的。然而，这样组织显示记忆体便於使所有的亮度位元都排列在一起、所有的红色位元都排在一起，等等。这样听起来就好像一种设备依赖特性，即 Windows 程式写作者不需要了解所有细节，但这时应或多或少地知道一些。不过，这些颜色面会出现在一些 API 呼叫中，例如 GetDeviceCaps 和 CreateBitmap。

Windows 98 和 Microsoft Windows NT 需要 VGA 或解析度更高的图形卡。这是目前公认的显示卡的最低标准。

1987 年，IBM 最早发表视讯图像阵列 (Video Graphics Array: VGA) 以及 PS/2 系列的个人电脑。它提供了许多不同的显示模式，但最好的图像模式 (Windows 也使用其中之一) 是水平显示 640 个图素，垂直显示 480 个图素，带有 16 种颜色。要显示 256 种颜色，最初的 VGA 必须切换到 320×240 的图形模式，这种图素数不适合 Windows 的正常工作。

一般人们已经忘记了最初 VGA 卡的颜色限制，因为其他硬体制造商很快就开发了「Super-VGA」(SVGA) 显示卡，它包括更多的视讯记忆体，可显示 256 种颜色并有多於 640×480 的模式。这是现在的标准，而且也是一件好事，因为对於现实世界中的图像来说，16 种颜色过於简单，有些不适合。

显示 256 种颜色的显示卡模式采用每图素 8 位元。不过，这些 8 位元值都不必与实际的颜色相符。事实上，显示卡提供了「调色盘对照表 (palette lookup table)」，该表允许软体指定这 8 位元的颜色值，以便与实际颜色相符合。在 Windows 中，应用程式不能直接存取调色盘对照表。实际上，Windows 储存了 256 种颜色中的 20 种，而应用程式可以通过「Windows 调色盘管理器」来自订其余的 236 种颜色。关於这些内容，我将在第十六章详细介绍。调色盘管理器允许应用程式在 256 色显示器上显示实际点阵图。Windows 所储存的 20 种颜色如表 14-2 所示。

表 14-2

IRGB	RGB 颜色	颜色名称
00000000	00-00-00	黑
00000001	80-00-00	暗红
00000010	00-80-00	暗绿
00000011	80-80-00	暗黄
00000100	00-00-80	暗蓝
00000101	80-00-80	暗洋红
00000110	00-80-80	暗青
00000111	C0-C0-C0	亮灰
00001000	C0-DC-C0	美元绿
00001001	A6-CA-F0	天蓝
11110110	FF-FB-F0	乳白
11110111	A0-A0-A4	中性灰
11111000	80-80-80	暗灰
11111001	FF-00-00	红
11111010	00-FF-00	绿
11111011	FF-FF-00	黄
11111100	00-00-FF	蓝
11111101	FF-00-FF	洋红
11111110	00-FF-FF	青
11111111	FF-FF-FF	白

最近几年，True-Color 显示卡很普遍，它们在每图素使用 16 位元或 24 位元。有时每图素虽然用了 16 位元，其中有 1 位元不用，而其他 15 位元主要近似於红、绿和蓝。这样红、绿和蓝每种都有 32 色阶，组合起来就可以达到 32,768 种颜色。更普遍的是，6 位元用於绿色（人类对此颜色最敏感），这样就可得到 65,536 种颜色。对于非技术性的 PC 使用者来说，他们并不喜欢看到诸如 32,768 或 65,536 之类的数字，因此通常将这种视讯显示卡称为 Hi-Color 显示卡，它能提供数以千计的颜色。

到了每个图素 24 位元时，我们总共有了 16,777,216 种颜色（或者 True Color、数百万的颜色），每个图素使用 3 位元组。这与今後的标准很相似，因为它大致代表了人类感官的极限而且也很方便。

在呼叫 GetDeviceCaps 时（参见第五章的 DEVCAPS 程式），您能利用 BITSPIXEL 和 PLANES 常数来获得显示卡的颜色单位，这些值显示如表 14-3 所示

表 14-3

BITSPIXEL	PLANES	颜色数
1	1	2
1	4	16
8	1	256
15 或 16	1	32, 768 或 65 536
24 或 32	1	16 777 216

最近，您应该不会再碰到单色显示器了，但即便碰到了，您的應用程式也应该不会发生问题。

GDI 支援的点阵图

Windows 图形装置介面 (GDI: Graphics Device Interface) 从 1.0 版开始支援点阵图。不过，一直到 Windows 3.0 以前，Windows 下唯一支援 GDI 物件的只有点阵图，以点阵图代号来使用。这些 GDI 点阵图物件是单色的，或者与实际的图像输出设备（例如视讯显示器）有相同的颜色单位。例如，与 16 色 VGA 相容的点阵图有四个颜色面。问题是这些颜色点阵图不能储存，也不能用於颜色单位不同的图像输出设备（如每图素占 8 位元就可以产生 256 种颜色的设备）上。

从 Windows 3.0 开始，定义了一种新的点阵图格式，我们称之为装置无关点阵图 (device-independent bitmap)，或者 DIB。DIB 包括了自己的调色盘，其中显示了与 RGB 颜色相对应的图素位元。DIB 能显示在任何位元映射输出设备上。这里唯一的问题是 DIB 的颜色通常一定会转换成设备实际表现出来的颜色。

与 DIB 同时，Windows 3.0 还介绍了「Windows 调色盘管理器」，它让程式能够从显示的 256 种颜色中自订颜色。就像我们在第十六章所看到的那样，应用程式通常在显示 DIB 时使用「调色盘管理器」。

Microsoft 在 Windows 95 (和 Windows NT 4.0) 中扩展了 DIB 的定义，并且在 Windows 98 (和 Windows NT 5.0) 中再次扩展。这些扩展增加了所谓的「图像颜色管理器 (ICM: Image Color Management)」，并允许 DIB 更精确地指定图像所需要的颜色。我将在第十五章简要讨论 ICM。

不论 DIB 多么重要，在处理点阵图时，早期的 GDI 点阵图物件依然扮演了重要的角色。掌握点阵图使用方式的最好方法是按各种用法在演进发展的时间顺序来学习，先从 GDI 点阵图物件和位元块传输的概念开始。

位元块传输

我前面提到过，您可以把整个视讯显示器看作是一幅大点阵图。您在萤幕上见到的图素由储存在视讯显示卡上记忆体中的位元来描述。任何视讯显示的矩形区域也都是一个点阵图，其大小是它所包含的行列数。

让我们从将图像从视讯显示的一个区域复制到另一个区域，开始我们在点阵图世界的旅行吧！这个是强大的 BitBlt 函式的工作。

Bitblt (读作「bit blit」) 代表「位元块传输 (bit-block transfer)」。BLT 起源於一条组合语言指令，该指令在 DEC PDP-10 上用来传输记忆体块。术语「bitblt」第一次用在图像上与 Xerox Palo Alto Research Center (PARC) 设计的 SmallTalk 系统有关。在 SmallTalk 中，所有的图形输出操作都使用 bitblt。程式写作者有时将 blt 用作动词，例如：「Then I wrote some code to blt the happy face to the screen and play a wave file.」

BitBlt 函式移动的是图素，或者（更明确地）是一个位元映射图块。您将看到，术语「传输 (transfer)」与 BitBlt 函式不尽相同。此函式实际上对图素执行了一次位元操作，而且可以产生一些有趣的结果。

简单的 BitBlt

程式 14-1 所示的 BITBLT 程式用 BitBlt 函式将程式系统的功能表图示（位於程式 Windows 的左上角）复制到它的显示区域。

程式 14-1 BITBLT

```

BITBLT.C
/*-----
    BITBLT.C --      BitBlt Demonstration
                                     (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName [] = TEXT ("BitBlt") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;

```

```

    wndclass.cbWndExtra          = 0 ;
    wndclass.hInstance          = hInstance ;
    wndclass.hIcon              = LoadIcon (NULL, IDI_INFORMATION) ;
    wndclass.hCursor            = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName        = NULL ;
    wndclass.lpszClassName      = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName,
MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("BitBlt Demo"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int          cxClient, cyClient, cxSource, cySource ;
    HDC                hdcClient, hdcWindow ;
    int                x, y ;
    PAINTSTRUCT        ps ;

    switch (message)
    {
    case WM_CREATE:
        cxSource = GetSystemMetrics (SM_CXSIZEFRAME) +
            GetSystemMetrics (SM_CXSIZE) ;
        cySource = GetSystemMetrics (SM_CYSIZEFRAME) +
            GetSystemMetrics (SM_CYCAPTION) ;
        return 0 ;

```

```
case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdcClient = BeginPaint (hwnd, &ps) ;
    hdcWindow = GetWindowDC (hwnd) ;

    for (y = 0 ; y < cyClient ; y += cySource)
    for (x = 0 ; x < cxClient ; x += cxSource)
    {
        BitBlt (hdcClient, x, y, cxSource, cySource,
                hdcWindow, 0, 0, SRCCOPY) ;
    }

    ReleaseDC (hwnd, hdcWindow) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

但为什么只用了一个 BitBlt 呢？实际上，那个 BITBLT 用系统功能表图示的多个副本来填满显示区域（在此情况下是资讯方块中普遍使用的 IDI_INFORMATION 图示），如图 14-1 所示。

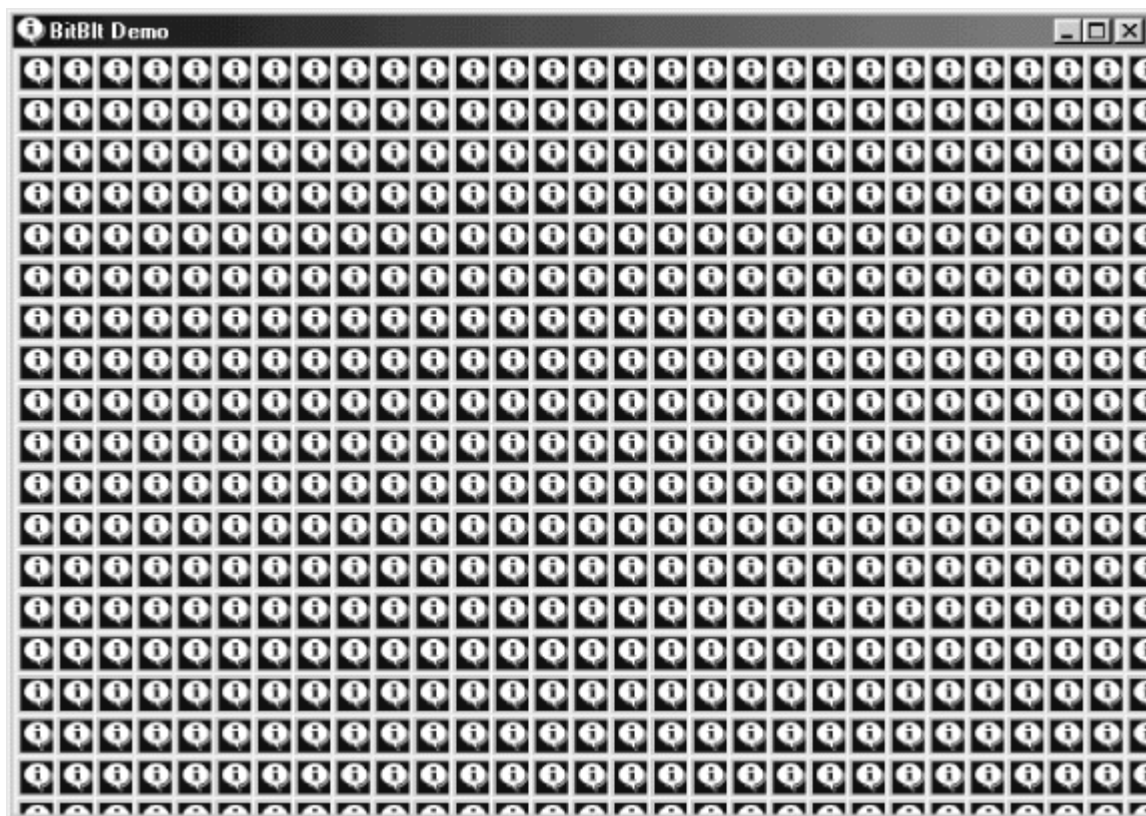


图 14-1 BITBLT 的萤幕显示

BitBlt 函式从称为「来源」的装置内容中将一个矩形区的图素传输到称为「目的(destination)」的另一个装置内容中相同大小的矩形区。此函式的语法如下：

```
BitBlt (hdcDst, xDst, yDst, cx, cy, hdcSrc, xSrc, ySrc, dwROP) ;
```

来源和目的装置内容可以相同。

在 BITBLT 程式中，目的装置内容是视窗的显示区域，装置内容代号从 BeginPaint 函式获得。来源装置内容是应用程式的整个视窗，此装置内容代号从 GetWindowDC 获得的。很明显地，这两个装置内容指的是同一个实际设备（视讯显示器）。不过，这两个装置内容的座标原点不同。

xSrc 和 ySrc 参数指明了来源图像左上角的座标位置。在 BITBLT 中，这两个参数设为 0，表示图像从来源装置内容（也就是整个视窗）的左上角开始，cx 和 cy 参数是图像的宽度和高度。BITBLT 根据从 GetSystemMetrics 函式获得的资讯来计算这些值。

xDst 和 yDst 参数表示了复制图像位置左上角的座标位置。在 BITBLT 中，这两个参数设定为不同的值以便多次复制图像。对于第一次 BitBlt 呼叫，这两个参数设为 0，将图像复制到显示区域的左上角位置。

BitBlt 的最后一个参数是位元映射操作型态。我将简短地讨论一下这个值。

请注意，BitBlt 是从实际视讯显示记忆体传输图素，而不是从系统功能表图示的其他图像传输。如果您移动 BITBLT 视窗以使部分系统功能表图示移出萤

幕，然後调整 BITBLT 视窗的尺寸使其重画，这时您将发现 BITBLT 显示区域中显示的是功能表图示的一部分。BitBlt 函式不再存取整个图像。

在 BitBlt 函式中，来源和目的装置内容可以相同。您可以重新编写 BITBLT 以使 WM_PAINT 处理执行以下内容：

```
BitBlt (hdcClient, 0, 0, cxSource, cySource,
        hdcWindow, 0, 0, SRCCOPY) ;
for (y = 0 ; y < cyClient ; y += cySource)
for (x = 0 ; x < cxClient ; x += cxSource)
{
    if (x > 0 || y > 0)
        BitBlt (hdcClient, x, y, cxSource, cySource,
                hdcClient, 0, 0, SRCCOPY) ;
}
```

这将与前面显示的 BITBLT 一样产生相同的效果，只是显示区域左上角比较模糊。

在 BitBlt 内的最大限制是两个装置内容必须是相容的。这意味著或者其中之一必须是单色的，或者两者的每个图素都相同的位元数。总而言之，您不能用此方法将萤幕上的某些图形复制到印表机。

拉伸点阵图

在 BitBlt 函式中，目的图像与来源图像的尺寸是相同的，因为函式只有两个参数来说明宽度和高度。如果您想在复制时拉伸或者压缩图像尺寸，可以使用 StretchBlt 函式。StretchBlt 函式的语法如下：

```
StretchBlt (    hdcDst, xDst, yDst, cxDst, cyDst,
                hdcSrc, xSrc, ySrc, cxSrc, cySrc, dwROP) ;
```

此函式添加了两个参数。现在的函式就分别包含了目的和来源各自的宽度和高度。STRETCH 程式展示了 StretchBlt 函式，如程式 14-2 所示。

程式 14-2 STRETCH

```
STRETCH.C
/*-----
    STRETCH.C --      StretchBlt Demonstration
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR    szAppName [] = TEXT ("Stretch") ;
```

```

    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS            wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_INFORMATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("StretchBlt Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int          cxClient, cyClient, cxSource, cySource ;
    HDC                hdcClient, hdcWindow ;
    PAINTSTRUCT         ps ;

```

```
switch (message)
{
case WM_CREATE:
    cxSource = GetSystemMetrics (SM_CXSIZEFRAME) +
        GetSystemMetrics (SM_CXSIZE) ;

    cySource = GetSystemMetrics (SM_CYSIZEFRAME) +
        GetSystemMetrics (SM_CYCAPTION) ;
    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdcClient = BeginPaint (hwnd, &ps) ;
    hdcWindow = GetWindowDC (hwnd) ;

    StretchBlt (hdcClient, 0, 0, cxClient, cyClient,
        hdcWindow, 0, 0, cxSource, cySource, MERGECOPY) ;

    ReleaseDC (hwnd, hdcWindow) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

此程式只有呼叫了 StretchBlt 函式一次，但是利用此函式以系统功能表图示填充了整个显示区域，如图 14-2 所示。

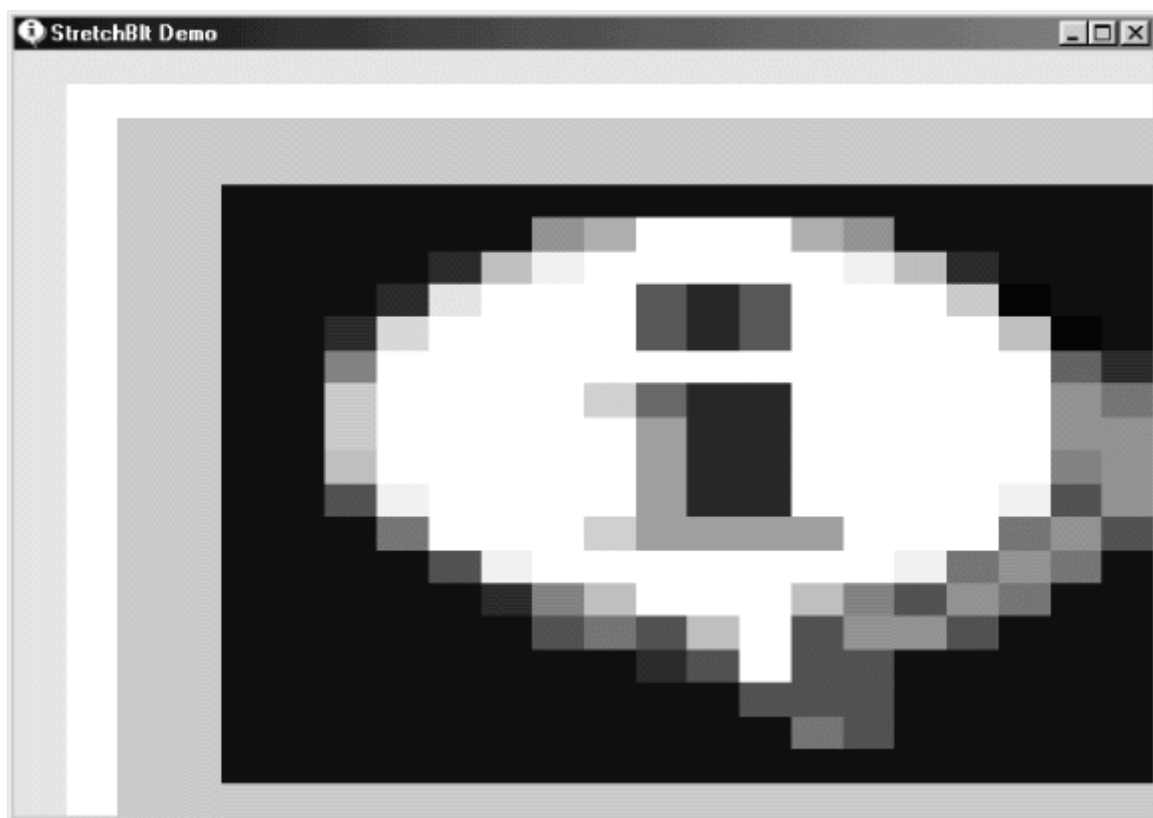


图 14-2 STRETCH 的萤幕显示

BitBlt 和 StretchBlt 函式中所有的座标与大小都是依据逻辑单位的。但是当您在 BitBlt 函式中定义了两个不同的装置内容，而这两个装置内容虽然参考同一个实际设备，却各自有著不同的映射模式，这时将发生什么结果呢？如果出现这种情况，呼叫 BitBlt 产生的结果就显得不明确了：cx 和 cy 参数都是逻辑单位，而它们同样应用於来源装置内容和目的装置内容中的矩形区。所有的座标和尺寸必须在实际的位元传输之前转换为装置座标。因为 cx 和 cy 值同时用於来源和目的装置内容，所以此值必须转换为装置内容自己的单位。

当来源和目的装置内容相同，或者两个装置内容都使用 MM_TEXT 图像模式时，装置单位下的矩形尺寸在两个装置内容中会是相同的，然後才由 Windows 进行图素对图素的转换。不过，如果装置单位下的矩形尺寸在两个装置内容中不同时，则 Windows 就把此工作转交给更通用的 StretchBlt 函式。

StretchBlt 也允许水平或垂直旋转图像。如果 cxSrc 和 cxDst 标记（转换成装置单位以後）不同，那么 StretchBlt 就建立一个镜像：左右旋转。在 STRETCH 程式中，通过将 xDst 参数改为 cxClient 并将 cxDst 参数改成 -cxClient，您就可以做到这一点。如果 cySrc 和 cyDst 不同，则 StretchBlt 会上下旋转图像。要在 STRETCH 程式中测试这一点，可将 yDst 参数改为 cyClient 并将 cyDst 参数改成 -cyClient。

StretchBlt 模式

使用 StretchBlt 会碰到一些与点阵图大小缩放相关的一些根本问题。在扩展一个点阵图时，StretchBlt 必须复制图素行或列。如果放大倍数不是原图的整数倍，那么此操作会造成产生的图像有些失真。

如果目的矩形比来源矩形小，那么 StretchBlt 在缩小图像时就必须把两行（或列）或者多行（或列）的图素合并到一行（或列）。完成此操作有四种方法，它根据装置内容伸展模式属性来选择其中一种方法。您可使用 SetStretchBltMode 函式来修改这个属性。

```
SetStretchBltMode (hdc, iMode) ;
```

iMode 可取下列值：

- BLACKONWHITE 或者 STRETCH_ANDSCANS（内定） 如果两个或多个图素得合并成一个图素，那么 StretchBlt 会对图素执行一个逻辑 AND 运算。这样的结果是只有全部的原始图素是白色时该图素才为白色，其实际意义是黑色图素控制了白色图素。这适用于白背景中主要是黑色的单色点阵图。
- WHITEONBLACK 或 STRETCH_ORSCANS 如果两个或多个图素得合并成一个图素，那么 StretchBlt 执行逻辑 OR 运算。这样的结果是只有全部的原-始图素都是黑色时才是黑色，也就是说由白色图素决定颜色。这适用于黑色背景中主要是白色的单色点阵图。
- COLORONCOLOR 或 STRETCH_DELETESCANS StretchBlt 简单地消除图素行或列，而没有任何逻辑组合。这是通常是处理彩色点阵图的最佳方法。
- HALFTONE 或 STRETCH_HALFTONE Windows 根据组合起来的来源颜色来计算目的平均颜色。这将与半调调色盘联合使用，第十六章将展示这一程序。

Windows 还包括用于取得目前伸展模式的 GetStretchBltMode 函式。

位元映射操作

BITBLT 和 STRETCH 程式简单地将来源点阵图复制给了目的点阵图，在过程中也可能进行了缩放。这是把 SRCCOPY 作为 BitBlt 和 StretchBlt 函式最后一个参数的结果。SRCCOPY 只是您能在这些函式中使用的 256 个位元映射操作中的一个。让我们先在 STRETCH 程式中做一个别的实验，然后再系统地研究位元映射操作。

尽量用 NOTSRCCOPY 来代替 SRCCOPY。与它们名称一样，位元映射操作在复制点阵图时转换其颜色。在显示区域视窗，所有的颜色转换：黑色变成白色、

白色变成黑色，蓝色变成黄色。现在试一下 SRCINVERT，您将得到同样效果。如果试一下 BLACKNESS，正如其名称一样，整个显示区域都将变成黑色，而 WHITENESS 则使其变成白色。

现在试一试下列三条叙述来代替 StretchBlt 呼叫：

```
SelectObject (hdcClient, CreateHatchBrush (HS_DIAGCROSS, RGB (0, 0, 0)));
StretchBlt (      hdcClient, 0, 0, cxClient, cyClient,
              hdcWindow, 0, 0, cxSource, cySource, MERGECOPY) ;

DeleteObject (hdcClient, GetStockObject (WHITE_BRUSH)) ;
```

这次，您将在图像上看到一个菱形的画刷，这是什么？

我在前面说过，BitBlt 和 StretchBlt 函式不是简单的位元块传输。此函式实际在下面三种图像间执行位元操作。

Source 来源点阵图，拉伸或压缩（如果有必要）到目的矩形的尺寸。

Destination 在 BitBlt 或 StretchBlt 呼叫之前的目的矩形。

Pattern 在目的装置内容中选择的目前画刷，水平或垂直地复制到目的矩形范围内。

结果是复制到了目的矩形中。

位元映射操作与我们在第五章遇到的绘图模式在概念上相似。绘图模式采用图像物件的控制项方式，例如一条线就组合成一个目的。我们知道有 16 种绘图模式——也就是说，物件中的 0 和 1 画出时，唯一结果就是目的中 0 和 1 的组合。

使用 BitBlt 和 StretchBlt 的位元映射操作包含了三个物件的组合，这将产生 256 种位元映射操作。有 256 种方法来组合来源点阵图、目的点阵图和图案。有 15 种位元映射操作已经命名——其中一些名称其实还不能够清楚清楚说明其意义——它们定义在 WINGDI.H 里头，其余的都有数值，列在/Platform SDK/Graphics and Multimedia Services/GDI/Raster Operation Codes/Ternary Raster Operations 之中。

有名称的 15 种 ROP 代码见表 14-4。

表 14-4

图案 (P) : 1 1 1 1 0 0 0 0 来源 (s) : 1 1 0 0 1 1 0 0 目的 (D) : 1 0 1 0 1 0 1 0		布林操作	ROP 代码	名称
结果:	0 0 0 0 0 0 0 0	0	0x000042	BLACKNESS
	0 0 0 1 0 0 0 1	~(S D)	0x1100A6	NOTSRCERASE
	0 0 1 1 0 0 1 1	~S	0x330008	NOTSRCCOPY

	0 1 0 0 0 1 0 0	$S \& \sim D$	0x440328	SRCERASE
	0 1 0 1 0 1 0 1	$\sim D$	0x550009	DSTINVERT
	0 1 0 1 1 0 1 0	$P \wedge D$	0x5A0049	PATINVERT
	0 1 1 0 0 1 1 0	$S \wedge D$	0x660046	SRCINVERT
	1 0 0 0 1 0 0 0	$S \& D$	0x8800C6	SRCAND
	1 0 1 1 1 0 1 1	$\sim S D$	0xBB0226	MERGEPAINT
	1 1 0 0 0 0 0 0	$P \& S$	0xC000CA	MERGECOPY
	1 1 0 0 1 1 0 0	S	0xCC0020	SRCCOPY
	1 1 1 0 1 1 1 0	$S D$	0xEE0086	SRCPAINT
	1 1 1 1 0 0 0 0	P	0xF00021	PATCOPY
	1 1 1 1 1 0 1 1	$P \sim S D$	0xFB0A09	PATPAINT
	1 1 1 1 1 1 1 1	1	0xFF0062	WHITENESS

此表格对於理解和使用位元映射操作非常重要，因此我们应花点时间来研究。

在这个表格中，「ROP 代码」行的值将传递给 BitBlt 或 StretchBlt 的最后一个参数；在「名称」行中的值在 WINGDI.H 定义。ROP 代码的低字组协助装置驱动程式传输位元映射操作。高字组是 0 到 255 之间的数值。此数值与第 2 列的图案的位元相同，这是在图案、来源和显示在顶部的目的之间进行位元操作的结果。「布林运算」列按 C 语法显示图案、来源和目的的组合方式。

要开始了解此表，最简单的办法是假定您正处理一个单色系统（每图素 1 位元）其中 0 代表黑色，1 代表白色。BLACKNESS 操作的结果是不管是来源、目的和图案是什么，全部为零，因此目的将显示黑色。类似地，WHITENESS 总导致目的呈白色。

现在假定您使用位元映射操作 PATCOPY。这导致结果位元与图案位元相同，而忽略了来源和目的点阵图。换句话说，PATCOPY 简单地将目前图案复制给了目的矩形。

PATPAINT 位元映射操作包含一个更复杂的操作。其结果相同於在图案、目的和反转的来源之间进行位元或操作。当来源点阵图是黑色（0）时，其结果总是白色（1）；当来源是白色（1）时，只要图案或目的为白色，则结果就是白色。换句话说，只有来源为白色而图案和目的都是黑色时，结果才是黑色。

彩色显示时每个图素都使用了多个位元。BitBlt 和 StretchBlt 函式对每个颜色位元都分别提供了位元操作。例如，如果目的是红色而来源为蓝色，SRCPAINT 位元映射操作把目的变成洋红色。注意，操作实际是按显示卡内储存

的位元执行的。这些位元所对应的颜色取决於显示卡的调色盘的设定。Windows 完成了此操作，以便位元映射操作能达到您预计的结果。不过，如果您修改了调色盘（我将在第十六章讨论），位元映射操作将产生无法预料的结果。

如要得到位元映射操作较好的应用程式，请参见本章後面的「非矩形点阵图图像」一节。

图案 Blt

除了 BitBlt 和 StretchBlt 以外，Windows 还包括一个称为 PatBlt（「pattern block transfer：图案块传输」）的函式。这是三个「blt」函式中最简单的。与 BitBlt 和 StretchBlt 不同，它只使用一个目的装置内容。PatBlt 语法是：

```
PatBlt (hdc, x, y, cx, cy, dwROP) ;
```

x、y、cx 和 cy 参数位於逻辑单位。逻辑点 (x,y) 指定了矩形的左上角。矩形宽为 cx 单位，高为 cy 单位。这是 PatBlt 修改的矩形区域。PatBlt 在画刷与目的装置内容上执行的逻辑操作由 dwROP 参数决定，此参数是 ROP 代码的子集——也就是说，您可以只使用那些不包括来源目的装置内容的 ROP 代码。下表列出了 PatBlt 支援的 16 个位元映射操作：

表 14-5

图案 (P) : 1 1 0 0 目的 (D) : 1 0 1 0		布林操作	ROP 代码	名称
结果:	0 0 0 0	0	0x000042	BLACKNESS
	0 0 0 1	$\sim(P \mid D)$	0x0500A9	
	0 0 1 0	$\sim P \ \& \ D$	0x0A0329	
	0 0 1 1	$\sim P$	0x0F0001	
	0 1 0 0	$P \ \& \ \sim D$	0x500325	
	0 1 0 1	$\sim D$	0x550009	DSTINVERT
	0 1 1 0	$P \ \wedge \ D$	0x5A0049	PATINVERT
	0 1 1 1	$\sim(P \ \& \ D)$	0x5F00E9	
	1 0 0 0	$P \ \& \ D$	0xA000C9	
	1 0 0 1	$\sim(P \ \wedge \ D)$	0xA50065	
	1 0 1 0	D	0xAA0029	
	1 0 1 1	$\sim P \ \mid \ D$	0xAF0229	

	1 1 0 0	P	0xF00021	PATCOPY
	1 1 0 1	P ~D	0xF50225	
	1 1 1 0	P D	0xFA0089	
	1 1 1 1	1	0xFF0062	WHITENESS

下面列出了 PatBlt 一些更常见用途。如果想画一个黑色矩形，您可呼叫

```
PatBlt (hdc, x, y, cx, cy, BLACKNESS) ;
```

要画一个白色矩形，请用

```
PatBlt (hdc, x, y, cx, cy, WHITENESS) ;
```

函式

```
PatBlt (hdc, x, y, cx, cy, DSTINVERT) ;
```

用於改变矩形的颜色。如果目前装置内容中选择了 WHITE_BRUSH，那么函式

```
PatBlt (hdc, x, y, cx, cy, PATINVERT) ;
```

也改变矩形。

您可以再次呼叫 FillRect 函式来用画笔充满一个矩形区域：

```
FillRect (hdc, &rect, hBrush) ;
```

FillRect 函式相同於下列代码：

```
hBrush = SelectObject (hdc, hBrush) ;
PatBlt (hdc,      rect.left, rect.top,
        rect.right - rect.left,
        rect.bottom - rect.top, PATCOPY) ;
SelectObject (hdc, hBrush) ;
```

实际上，此程式码是 Windows 用於执行 FillRect 函式的动作。如果您呼叫

```
InvertRect (hdc, &rect) ;
```

Windows 将其转换成函式：

```
PatBlt (hdc,      rect.left, rect.top,
        rect.right - rect.left,
        rect.bottom - rect.top, DSTINVERT) ;
```

在介绍 PatBlt 函式的语法时，我说过点 (x, y) 指出了矩形的左上角，而且此矩形宽度为 cx 单位，高度为 cy 单位。此叙述并不完全正确。BitBlt、PatBlt 和 StretchBlt 是最合适的 GDI 画图函式，它们根据从一个角测得的逻辑宽度和高度来指定逻辑直角座标。矩形边框用到的其他所有 GDI 画图函式都要求根据左上角和右下角的座标来指定座标。对于 MM_TEXT 映射模式，上面讲述的 PatBlt 参数就是正确的。然而对于公制的映射模式来说，就不正确。如果您使用一的 cx 和 cy 值，那么点 (x, y) 将是矩形的左下角。如果希望点 (x, y) 是矩形的左上角，那么 cy 参数必须设为矩形的负高度。

如果想更精确，用 PatBlt 修改颜色的矩形将通过 cx 的绝对值获得逻辑宽度，通过 cy 的绝对值获得逻辑高度。这两个参数可以是负值。由逻辑点 (x, y)

和 $(x + cx, y + cy)$ 给定的两个角定义了矩形。矩形的左上角通常属于 PatBlt 修改的区域。右上角则超出了矩形的范围。根据映射模式和 cx 、 cy 参数的符号，矩形左上角的点应为 (x, y) 、 $(x, y + cy)$ 、 $(x + cx, y)$ 或者 $(x + cx, y + cy)$ 。

如果给 MM_LOENGLISH 设定了映射模式，并且您想在显示区域左上角的一小块正方形上使用 PatBlt，您可以使用

```
PatBlt (hdc, 0, 0, 100, -100, dwROP) ;
```

或

```
PatBlt (hdc, 0, -100, 100, 100, dwROP) ;
```

或

```
PatBlt (hdc, 100, 0, -100, -100, dwROP) ;
```

或

```
PatBlt (hdc, 100, -100, -100, 100, dwROP) ;
```

给 PatBlt 设定正确参数最容易的方法是将 x 和 y 设为矩形左上角。如果映射模式定义 y 座标随著向上卷动显示而增加，那么请使用负的 cy 参数。如果映射模式定义 x 座标向左增加（很少有人用），则需要使用负的 cx 参数。

GDI 点阵图物件

我在本章前面已提到过 Windows 从 1.0 开始就支援 GDI 点阵图物件。因为在 Windows 3.0 发表了装置无关点阵图，GDI 点阵图物件有时也称为装置相关点阵图，或者 DDB。我尽量不全部引用 device-dependent bitmap 的全文，因为它看上去与 device-independent bitmap 类似。缩写 DDB 会好一些，因为我们很容易把它与 DIB 区别开来。

对程式写作者来说，现存的两种不同型态的点阵图从 Windows 3.0 开始就更为混乱。许多有经验的 Windows 程式写作者都不能准确地理解 DIB 和 DDB 之间的关系。（恐怕本书的 Windows 3.0 版本不能澄清这个问题）。诚然，DIB 和 DDB 在许多方面是相关的：DIB 与 DDB 能相互转换（尽管转换程序中会丢失一些资讯）。然而 DIB 和 DDB 是不可以相互替换的，并且不能简单地选择一种方法来表示同一个可视资料。

如果我们能假设说 DIB 一定会替代 DDB，那以后就会很方便了。但现实并不是如此，DDB 还在 Windows 中扮演著很重要角色，尤其是您在乎程式执行表现好坏时。

建立 DDB

DDB 是 Windows 图形装置介面的图形物件之一（其中还包括绘图笔、画刷、

字体、metafile 和调色盘)。这些图形物件储存在 GDI 模组内部, 由应用程式软体以代号数字的方式引用。您可以将 DDB 代号储存在一个 HBITMAP (「handle to a bitmap: 点阵图代号」) 型态的变数中, 例如:

```
HBITMAP hBitmap ;
```

然後通过呼叫 DDB 建立的一个函式来获得代号, 例如: CreateBitmap。这些函式配置并初始化 GDI 记忆体中的一些记忆体来储存关于点阵图的资讯, 以及实际点阵图位元的资讯。应用程式不能直接存取这段记忆体。点阵图与装置内容无关。当程式使用完点阵图以後, 就要清除这段记忆体:

```
DeleteObject (hBitmap) ;
```

如果程式执行时您使用了 DDB, 那么程式终止时, 您可以完成上面的操作。

CreateBitmap 函式用法如下:

```
hBitmap = CreateBitmap (cx, cy, cPlanes, cBitsPixel, bits) ;
```

前两个参数是点阵图的宽度和高度 (以图素为单位), 第三个参数是颜色面的数目, 第四个参数是每图素的位元数, 第五个参数是指向一个以特定颜色格式存放的位元阵列的指标, 阵列内存放有用来初始化该 DDB 的图像。如果您不想用一张现有的图像来初始化 DDB, 可以将最後一个参数设为 NULL。以後您还是可以设定该 DDB 内图素的内容。

使用此函式时, Windows 也允许建立您喜欢的特定型态 GDI 点阵图物件。例如, 假设您希望点阵图宽 7 个图素、高 9 个图素、5 个颜色位元面, 并且每个图素占 3 位元, 您只需要执行下面的操作:

```
hBitmap = CreateBitmap (7, 9, 5, 3, NULL) ;
```

这时 Windows 会好好给您一个有效的点阵图代号。

在此函式呼叫期间, Windows 将储存您传递给函式的资讯, 并为图素位元配置记忆体。粗略的计算是此点阵图需要 $7 \times 9 \times 5 \times 3$, 即 945 位元, 这要比 118 个位元组还多几个位元。

然而, Windows 为点阵图配置好记忆体以後, 每行图素都占用许多连贯的位元组, 这样

```
iWidthBytes = 2 * ((cx * cBitsPixel + 15) / 16) ;
```

或者 C 程式写作者更倾向於写成:

```
iWidthBytes = (cx * cBitsPixel + 15) & ~15) >> 3 ;
```

因此, 为 DDB 配置的记忆体就是:

```
iBitmapBytes = cy * cPlanes * iWidthBytes ;
```

本例中, iWidthBytes 占 4 位元组, iBitmapBytes 占 180 位元组。

现在, 知道一张点阵图有 5 个颜色位元面, 每图素占 3 个颜色位有什么意义吗? 真是见鬼了, 这甚至不能把它称作一个习题作业。虽然您让 GDI 内部配置了些记忆体, 并且让这些记忆体有一定结构的内容, 但是您这张点阵图完全

作不出任何有用的事情来。

实际上, 您将用两种型态的参数来呼叫 CreateBitmap。

cPlanes 和 cBitsPixel 都等於 1 (表示单色点阵图); 或者

cPlanes 和 cBitsPixel 都等於某个特定装置内容的值, 您可以使用 PLANES 和 BITSPIXEL 索引来从 GetDeviceCaps 函式获得。

更现实的情况下, 您只会在第一种情况下呼叫 CreateBitmap。对於第二种情况, 您可以用 CreateCompatibleBitmap 来简化问题:

```
hBitmap = CreateCompatibleBitmap (hdc, cx, cy) ;
```

此函式建立了一个与设备相容的点阵图, 此设备的装置内容代号由第一个参数给出。CreateCompatibleBitmap 用装置内容代号来获得 GetDeviceCaps 资讯, 然後将此资讯传递给 CreateBitmap。除了与实际的装置内容有相同的记忆体组织之外, DDB 与装置内容没有其他联系。

CreateDiscardableBitmap 函式与 CreateCompatibleBitmap 的参数相同, 并且功能上相同。在早期的 Windows 版本中, CreateDiscardableBitmap 建立的点阵图可以在记忆体减少时由 Windows 将其从记忆体中清除, 然後程式再重建点阵图资料。

第三个点阵图建立函式是 CreateBitmapIndirect:

```
hBitmap CreateBitmapIndirect (&bitmap) ;
```

其中 bitmap 是 BITMAP 型态的结构。BITMAP 结构定义如下:

```
typedef struct _tagBITMAP
{
    LONG          bmType ;           // set to 0
    LONG          bmWidth ;          // width in pixels
    LONG          bmHeight ;         // height in pixels
    LONG          bmWidthBytes ;     // width of row in bytes
    WORD          bmPlanes ;         // number of color planes
    WORD          bmBitsPixel ;      // number of bits per pixel
    LPVOID        bmBits ;           // pointer to pixel bits
}
BITMAP, * PBITMAP ;
```

在呼叫 CreateBitmapIndirect 函式时, 您不需要设定 bmWidthBytes 栏位。Windows 将为您计算, 您也可以将 bmBits 栏位设定为 NULL, 或者设定为初始化点阵图时用的图素位元位址。

GetObject 函式内也使用 BITMAP 结构, 首先定义一个 BITMAP 型态的结构。

```
BITMAP bitmap ;
```

并呼叫函式如下:

```
GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;
```

Windows 将用点阵图资讯填充 BITMAP 结构的栏位, 不过, bmBits 栏位等於

NULL。

您最後应呼叫 DeleteObject 来清除程式内建立的所有点阵图。

点阵图位元

用 CreateBitmap 或 CreateBitmapIndirect 来建立设备相关 GDI 点阵图物件时，您可以给点阵图图素位元指定一个指标。或者您也可以让点阵图维持未初始化的状态。在建立点阵图以後，Windows 还提供两个函式来获得并设定图素位元。

要设定图素位元，请呼叫：

```
SetBitmapBits (hBitmap, cBytes, &bits) ;
```

GetBitmapBits 函式有相同的语法：

```
GetBitmapBits (hBitmap, cBytes, &bits) ;
```

在这两个函式中，cBytes 指明要复制的位元组数，bits 是最少 cBytes 大小的缓冲区。

DDB 中的图素位元从顶列开始排列。我在前面说过，每列的位元组数都是偶数。除此之外，没什么好说明的了。如果点阵图是单色的，也就是说它有 1 个位元面并且每个图素占 1 位元，则每个图素不是 1 就是 0。每列最左边的图素是本列第一个位元组最高位元的位元。我们在本章的後面讲完如何显示单色 DDB 之後，将做一个单色的 DDB。

對於非单色点阵图，应避免出现您需要知道图素位元含义的状况。例如，假定在 8 位颜色的 VGA 上执行 Windows，您可以呼叫 CreateCompatibleBitmap。通过 GetDeviceCaps，您能够确定您正处理一个有 1 个颜色位元面和每图素 8 位元的设备。一个位元组储存一个图素。但是图素值 0x37 是什么意思呢？很明显是某种颜色，但到底是什么颜色呢？

图素实际上并不涉及任何固定的颜色，它只是一个值。DDB 没有颜色表。问题的关键在於：当 DDB 显示在萤幕上时，图素的颜色是什么。它肯定是某种颜色，但具体是什么颜色呢？显示的图素将与在显示卡上的调色盘查看表里的 0x37 索引值代表的 RGB 颜色有关。这就是您现在碰到的装置依赖性。

不过，不要只因为我们不知道图素值的含义，就假定非单色 DDB 没用。我们将简要看一下它们的用途。下一章，我们将看到 SetBitmapBits 和 GetBitmapBits 函式是如何被更有用的 SetDIBits 和 GetDIBits 函式所取代的。

因此，基本的规则是这样的：不要用 CreateBitmap、CreateBitmapIndirect 或 SetBitmapBits 来设定彩色 DDB 的位元，您只能安全地使用这些函式来设定单色 DDB 的位元。（如果您在呼叫 GetBitmapBits 期间，从其他相同格式的 DDB 中获得位元，那么这些规则例外。）

在继续之前，让我再讨论一下 `SetBitmapDimensionEx` 和 `GetBitmapDimensionEx` 函式。这些函式让您设定（和获得）点阵图的测量尺寸（以 0.1 毫米为单位）。这些资讯与点阵图解析度一起储存在 GDI 中，但不用于任何操作。它只是您与 DDB 联系的一个测量尺寸标识。

记忆体装置内容

我们必须解决的下一个概念是记忆体装置内容。您需要用记忆体装置内容来处理 GDI 点阵图物件。

通常，装置内容指的是特殊的图形输出设备（例如视讯显示器或者印表机）及其装置驱动程式。记忆体装置内容只位于记忆体中，它不是真正的图形输出设备，但可以说与指定的真正设备「相容」。

要建立一个记忆体装置内容，您必须首先有实际设备的装置内容代号。如果是 `hdc`，那么您可以像下面那样建立记忆体装置内容：

```
hdcMem = CreateCompatibleDC (hdc) ;
```

通常，函式的呼叫比这更简单。如果您将参数设为 `NULL`，那么 Windows 将建立一个与视讯显示器相相容的记忆体装置内容。应用程式建立的任何记忆体装置内容最终都通过呼叫 `DeleteDC` 来清除。

记忆体装置内容有一个与实际位元映射设备相同的显示平面。不过，最初此显示平面非常小——单色、1 图素宽、1 图素高。显示平面就是单独 1 位元。

当然，用 1 位元的显示平面，您不能做更多的工作，因此下一步就是扩大显示平面。您可以通过将一个 GDI 点阵图物件选进记忆体装置内容来完成这项工作，例如：

```
SelectObject (hdcMem, hBitmap) ;
```

此函式与您将画笔、画刷、字体、区域和调色盘选进装置内容的函式相同。然而，记忆体装置内容是您可以选进点阵图的唯一一种装置内容型态。（如果需要，您也可以将其他 GDI 物件选进记忆体装置内容。）

只有选进记忆体装置内容的点阵图是单色的，或者与记忆体装置内容相容设备有相同的色彩组织时，`SelectObject` 才会起作用。这也是建立特殊的 DDB（例如有 5 个位元面，且每图素 3 位元）没有用的原因。

现在情况是这样：`SelectObject` 呼叫以后，DDB 就是记忆体装置内容的显示平面。处理实际装置内容的每项操作，您几乎都可以用于记忆体装置内容。例如，如果用 GDI 画图函式在记忆体装置内容中画图，那么图像将画在点阵图上。这是非常有用的。还可以将记忆体装置内容作为来源，把视讯装置内容作为目的来呼叫 `BitBlt`。这就是在显示器上绘制点阵图的方法。如果把视讯装置内容作为来源，把记忆体装置内容作为目的，那么呼叫 `BitBlt` 可将萤幕上的一

些内容复制给点阵图。我们将看到这些都是可能的。

载入点阵图资源

除了各种各样的点阵图建立函式以外，获得 GDI 点阵图物件代号的另一个方法就是呼叫 LoadBitmap 函式。使用此函式，您不必担心点阵图格式。在程式中，您只需简单地按资源来建立点阵图，这与建立图示或者滑鼠游标的方法类似。LoadBitmap 函式的语法与 LoadIcon 和 LoadCursor 相同：

```
hBitmap = LoadBitmap (hInstance, szBitmapName) ;
```

如果想载入系统点阵图，那么将第一个参数设为 NULL。这些不同的点阵图是 Windows 视觉介面（例如关闭方块和勾选标记）的一小部分，它们的识别字以字母 OBM 开始。如果点阵图与整数识别字而不是与名称有联系，那么第二个参数就可以使用 MAKEINTRESOURCE 巨集。由 LoadBitmap 载入的所有点阵图最终应用 DeleteObject 清除。

如果点阵图资源是单色的，那么从 LoadBitmap 传回的代号将指向一个单色的点阵图物件。如果点阵图资源不是单色，那么从 LoadBitmap 传回的代号将指向一个 GDI 点阵图物件，该物件与执行程式的视讯显示器有相同的色彩组织。因此，点阵图始终与视讯显示器相容，并且总是选进与视讯显示器相容的记忆体装置内容中。采用 LoadBitmap 呼叫後，就不用担心任何色彩转换的问题了。在下一章中，我们就知道 LoadBitmap 的具体运作方式了。

程式 14-3 所示的 BRICKS1 程式示范了载入一小张单色点阵图资源的方法。此点阵图本身不像砖块，但当它水平和垂直重复时，就与砖墙相似了。

程式 14-3 BRICKS1

```
BRICKS1.C
/*-----
    BRICKS1.C -- LoadBitmap Demonstration
                                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR    szAppName [] = TEXT ("Bricks1") ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS         wndclass ;
```

```

    wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc          = WndProc ;
    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance            = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName         = NULL ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("LoadBitmap Demo"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static HBITMAP hBitmap ;
    static int      cxClient, cyClient, cxSource, cySource ;
    BITMAP          bitmap ;
    HDC              hdc, hdcMem ;
    HINSTANCE        hInstance ;
    int              x, y ;
    PAINTSTRUCT      ps ;

    switch (message)
    {

```

```

    case WM_CREATE:
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;

        hBitmap = LoadBitmap (hInstance, TEXT ("Bricks")) ;

        GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

        cxSource = bitmap.bmWidth ;
        cySource = bitmap.bmHeight ;

        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;

        for (y = 0 ; y < cyClient ; y += cySource)
            for (x = 0 ; x < cxClient ; x += cxSource)
            {
                BitBlt (hdc, x, y, cxSource, cySource, hdcMem, 0, 0,
SRCCOPY) ;
            }

        DeleteDC (hdcMem) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        DeleteObject (hBitmap) ;
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

BRICKS1.RC (摘录)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/

```

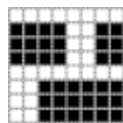
```
// Bitmap
```

```
BRICKS
```

```
BITMAP DISCARDABLE
```

```
"Bricks.bmp"
```

BRICKS.BMP



在 Visual C++ Developer Studio 中建立点阵图时，应指明点阵图的高度和宽度都是 8 个图素，是单色，名称是「Bricks」。BRICKS1 程式在 WM_CREATE 讯息处理期间载入了点阵图并用 GetObject 来确定点阵图的图素尺寸（以便当点阵图不是 8 图素见方时程式仍能继续工作）。以後，BRICKS1 将在 WM_DESTROY 讯息中删除此点阵图。

在 WM_PAINT 讯息处理期间，BRICKS1 建立了一个与显示器相容的记忆体装置内容，并且选进了点阵图。然後是从记忆体装置内容到显示区域装置内容一系列的 BitBlt 函式呼叫，再删除记忆体装置内容。图 14-3 显示了程式的执行结果。

顺便说一下，Developer Studio 建立的 BRICKS.BMP 档案是一个装置无关点阵图。您可能想在 Developer Studio 内建立一个彩色的 BRICKS.BMP 档案（您可自己选定颜色），并且保证一切工作正常。

我们看到 DIB 能转换成与视讯显示器相容的 GDI 点阵图物件。我们将在下一章看到这是如何操作的。

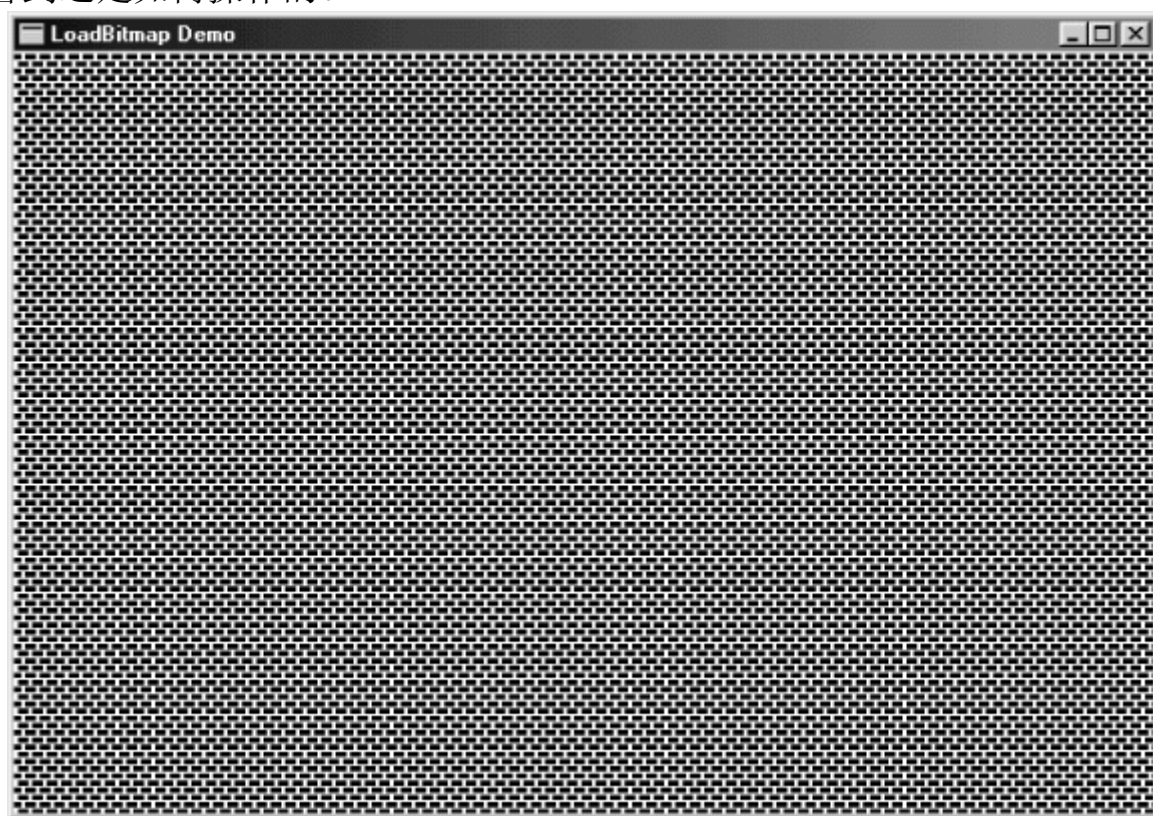
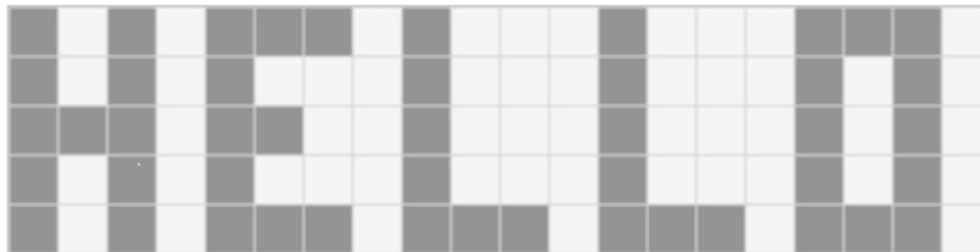


图 14-3 BRICKS1 的萤幕显示

单色点阵图格式

如果您在处理小块单色图像，那么您不必把它们当成资源来建立。与彩色点阵图物件不同，单色位元的格式相对简单一些，而且几乎能全部从您要建立的图像中分离出来。例如，假定您要建立下图所示的点阵图：



您能写下一系列的位元（0 代表黑色，1 代表白色），这些位元直接对应於网格。从左到右读这些位元，您能给每 8 位元组配置一个十六进位元的位元组值。如果点阵图的宽度不是 16 的倍数，在位元组的右边用零填充，以得到偶数个位元组：

```
0 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 = 51 77 10 00
0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 57 77 50 00
0 0 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 13 77 50 00
0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 57 77 50 00
0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 = 51 11 10 00
```

图素宽为 20，扫描线高为 5，位元组宽为 4。您可以用下面的叙述来设定此点阵图的 BITMAP 结构：

```
static BITMAP bitmap = { 0, 20, 5, 4, 1, 1 } ;
```

并且可以将位元储存在 BYTE 阵列中：

```
static BYTE bits [] = { 0x51, 0x77, 0x10, 0x00,
                        0x57, 0x77, 0x50, 0x00,
                        0x13, 0x77, 0x50, 0x00,
                        0x57, 0x77, 0x50, 0x00,
                        0x51, 0x11, 0x10, 0x00 } ;
```

用 CreateBitmapIndirect 来建立点阵图需要下面两条叙述：

```
bitmap.bmBits = (PSTR) bits ;
hBitmap = CreateBitmapIndirect (&bitmap) ;
```

另一种方法是：

```
hBitmap = CreateBitmapIndirect (&bitmap) ;
SetBitmapBits (hBitmap, sizeof bits, bits) ;
```

您也可以用一道叙述来建立点阵图：

```
hBitmap = CreateBitmap (20, 5, 1, 1, bits) ;
```

在程式 14-4 显示的 BRICKS2 程式利用此技术直接建立了砖块点阵图，而没有使用资源。

[程式 14-4 BRICKS2](#)

BRICKS2.C

```

/*-----
    BRICKS2.C --      CreateBitmap Demonstration
                                (c) Charles Petzold, 1998
-----*/

/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName [] = TEXT ("Bricks2") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("CreateBitmap Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

```



```

    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND  hwnd,   UINT  message,   WPARAM  wParam,LPARAM
lParam)
{
    static BITMAPINFO      Pbitmap = {           0,  8,  8,  2,  1,  1 } ;
    static BYTE            bits  [8][2]={        0xFF, 0, 0x0C, 0, 0x0C, 0, 0x0C,
0,
                                0xFF, 0, 0xC0, 0, 0xC0, 0, 0xC0, 0 } ;

    static HBITMAP hBitmap ;
    static int      cxClient, cyClient, cxSource, cySource ;
    HDC             hdc, hdcMem ;
    int             x, y ;
    PAINTSTRUCT     ps ;

    switch (message)
    {
    case  WM_CREATE:
        bitmap.bmBits = bits ;
        hBitmap      = CreateBitmapIndirect (&bitmap) ;
        cxSource      = bitmap.bmWidth ;
        cySource      = bitmap.bmHeight ;
        return 0 ;

    case  WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case  WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;

        for (y = 0 ; y < cyClient ; y += cySource)
        for (x = 0 ; x < cxClient ; x += cxSource)
        {
            BitBlt (hdc, x, y, cxSource, cySource, hdcMem,
0, 0, SRCCOPY) ;
        }

        DeleteDC (hdcMem) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;
    }
}

```

```

case WM_DESTROY:
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

您可以尝试一下与彩色点阵图相似的物件。例如，如果您的视讯显示器执行在 256 色模式下，那么您可以根据表 14-2 来定义彩色砖的每个图素。不过，当程式执行在其他显示模式下时，此程式码不起作用。以装置无关方式处理彩色点阵图需要使用下章讨论的 DIB。

点阵图中的画刷

BRICKS 系列的最後一个专案是 BRICKS3，如程式 14-5 所示。乍看此程式，您可能会有这种感觉：程式码哪里去了呢？

程式 14-5 BRICKS3

```

BRICKS3.C
/*-----
    BRICKS3.C -- CreatePatternBrush Demonstration
                                   (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR          szAppName [] = TEXT ("Bricks3") ;
    HBITMAP               hBitmap ;
    HBRUSH                 hBrush ;
    HWND                  hwnd ;
    MSG                   msg ;
    WNDCLASS               wndclass ;

    hBitmap = LoadBitmap (hInstance, TEXT ("Bricks")) ;
    hBrush = CreatePatternBrush (hBitmap) ;
    DeleteObject (hBitmap) ;

    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance ;

```

```

    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = hBrush ;
    wndclass.lpszMenuName    = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("CreatePatternBrush Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }

    DeleteObject (hBrush) ;
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BRICKS3.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

```

```

////////////////////////////////////
/
// Bitmap
BRICKS                                BITMAP DISCARDABLE    "Bricks.bmp"

```

此程式与 BRICKS1 使用同一个 BRICKS.BMP 档案，而且视窗看上去也相同。

正如您看到的一样，视窗讯息处理程式没有更多的内容。BRICKS3 实际上使用砖块图案作为视窗类别背景画刷，它在 WNDCLASS 结构的 hbrBackground 栏位中定义。

您现在可能猜想 GDI 画刷是很小的点阵图，通常是 8 个图素见方。如果将 LOGBRUSH 结构的 lbStyle 栏位设定为 BS_PATTERN，然後呼叫 CreatePatternBrush 或 CreateBrushIndirect，您就可以在点阵图外面来建立画刷了。此点阵图至少是宽高各 8 个图素。如果再大，Windows 98 将只使用点阵图的左上角作为画刷。而 Windows NT 不受此限制，它会使用整个点阵图。

请记住，画刷和点阵图都是 GDI 物件，而且您应该在程式终止前删除您在程式中建立画刷和点阵图。如果您依据点阵图建立画刷，那么在用画刷画图时，Windows 将复制点阵图位元到画刷所绘制的区域内。呼叫 CreatePatternBrush（或者 CreateBrushIndirect）之後，您可以立即删除点阵图而不会影响到画笔。类似地，您也可以删除画刷而不会影响到您选进的原始点阵图。注意，BRICKS3 在建立画刷後删除了点阵图，并在程式终止前删除了画刷。

绘制点阵图

在视窗中绘图时，我们已经将点阵图当成绘图来源使用过了。这要求先将点阵图选进记忆体装置内容，并呼叫 BitBlt 或者 StretchBlt。您也可以记忆体装置内容代号作为所有实际呼叫的 GDI 函式中的第一参数。记忆体装置内容的动作与实际的装置内容相同，除非显示平面是点阵图。

程式 14-6 所示的 HELLOBIT 程式展示了此项技术。程式在一个小点阵图上显示了字串「Hello, world!」，然後从点阵图到程式显示区域执行 BitBlt 或 StretchBlt（依照选择的功能表选项而定）。

程式 14-6 HELLOBIT

```

HELLOBIT.C
/*-----
HELLOBIT.C --          Bitmap Demonstration
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

```

```

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR          szAppName [] = TEXT ("HelloBit") ;
    HWND                  hwnd ;
    MSG                    msg ;
    WNDCLASS               wndclass ;

    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName  = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("HelloBit"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }

    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{

```

```

static HBITMAP hBitmap ;
static HDC hdcMem ;
static int cxBitmap, cyBitmap, cxClient, cyClient, iSize =
IDM_BIG ;
static TCHAR * szText = TEXT (" Hello, world! ") ;
HDC hdc ;
HMENU hMenu ;
int x, y ;
PAINTSTRUCT ps ;
SIZE size ;

switch (message)
{
case WM_CREATE:
    hdc = GetDC (hwnd) ;
    hdcMem = CreateCompatibleDC (hdc) ;

    GetTextExtentPoint32 (hdc, szText, lstrlen (szText),
&size) ;

    cxBitmap = size.cx ;
    cyBitmap = size.cy ;
    hBitmap = CreateCompatibleBitmap (hdc, cxBitmap,
cyBitmap) ;

    ReleaseDC (hwnd, hdc) ;

    SelectObject (hdcMem, hBitmap) ;
    TextOut (hdcMem, 0, 0, szText, lstrlen (szText)) ;
    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
case IDM_BIG:
case IDM_SMALL:
        CheckMenuItem (hMenu, iSize,
MF_UNCHECKED) ;

        iSize = LOWORD (wParam) ;
        CheckMenuItem (hMenu, iSize,
MF_CHECKED) ;

        InvalidateRect (hwnd, NULL, TRUE) ;

```

```

                                break ;
                                }
                                return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    switch (iSize)
    {
    case IDM_BIG:
        StretchBlt (hdc, 0, 0, cxClient,
cyClient,
        hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
        break ;

    case IDM_SMALL:
        for (y = 0 ; y < cyClient ; y += cyBitmap)
            for (x = 0 ; x < cxClient ; x += cxBitmap)
            {
                BitBlt (hdc, x, y, cxBitmap, cyBitmap,
hdcMem, 0, 0, SRCCOPY) ;
            }
        break ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    DeleteDC (hdcMem) ;
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

HELLOBIT.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////

/

// Menu

HELLOBIT MENU DISCARDABLE

BEGIN

POPUP "&Size"

BEGIN

```

        MENUITEM "&Big",          IDM_BIG, CHECKED
        MENUITEM "&Small",        IDM_SMALL
    END
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by HelloBit.rc

#define IDM_BIG          40001
#define IDM_SMALL        40002

```

程式从呼叫 `GetTextExtentPoint32` 确定字串的图素尺寸开始。这些尺寸将成为与视讯显示相容的点阵图尺寸。当此点阵图被选进记忆体装置内容（也与视讯显示相容）後，再呼叫 `TextOut` 将文字显示在点阵图上。记忆体装置内容在程式执行期间保留。在处理 `WM_DESTROY` 资讯期间，`HELLOBIT` 删除了点阵图和记忆体装置内容。

`HELLOBIT` 中的一条功能表选项允许您显示点阵图尺寸，此尺寸或者是显示区域中水平和垂直方向平铺的实际尺寸，或者是缩放成显示区域大小的尺寸，如图 14-4 所示。正与您所见到的一样，这不是显示大尺寸字元的好方法！它只是小字体的放大版，并带有放大时产生的锯齿线。



图 14-4 `HELLOBIT` 的萤幕显示

您可能想知道一个程式，例如 `HELLOBIT`，是否需要处理 `WM_DISPLAYCHANGE` 讯息。只要使用者（或者其他应用程式）修改了视讯显示大小或者颜色深度，应用程式就接收到此讯息。其中颜色深度的改变会导致记忆体装置内容和视讯

装置内容不相容。但这并不会发生，因为当显示模式修改後，Windows 自动修改了记忆体装置内容的颜色解析度。选进记忆体装置内容的点阵图仍然保持原样，但不会造成任何问题。

阴影点阵图

在记忆体装置内容绘图(也就是点阵图)的技术是执行「阴影点阵图(shadow bitmap)」的关键。此点阵图包含视窗显示区域中显示的所有内容。这样，对 WM_PAINT 讯息的处理就简化到简单的 BitBlt。

阴影点阵图在绘画程式中最有用。程式 14-7 所示的 SKETCH 程式并不是一个最完美的绘画程式，但它是一个开始。

程式 14-7 SKETCH

```
SKETCH.C
/*-----
    SKETCH.C -- Shadow Bitmap Demonstration
                                           (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR          szAppName [] = TEXT ("Sketch") ;
    HWND                  hwnd ;
    MSG                    msg ;
    WNDCLASS               wndclass ;

    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName  = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows
```

```

NT!"),
        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Sketch"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    if (hwnd == NULL)
    {
        MessageBox (    NULL, TEXT ("Not enough memory to create
bitmap!"),
                        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void GetLargestDisplayMode (int * pcxBitmap, int * pcyBitmap)
{
    DEVMODE    devmode ;
    int        iModeNum = 0 ;

    * pcxBitmap = * pcyBitmap = 0 ;

    ZeroMemory (&devmode, sizeof (DEVMODE)) ;
    devmode.dmSize = sizeof (DEVMODE) ;

    while (EnumDisplaySettings (NULL, iModeNum++, &devmode))
    {
        * pcxBitmap = max (* pcxBitmap, (int) devmode.dmPelsWidth) ;
        * pcyBitmap = max (* pcyBitmap, (int) devmode.dmPelsHeight) ;
    }
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)

```

```
{
    static BOOL      fLeftButtonDown, fRightButtonDown ;
    static HBITMAP hBitmap ;
    static HDC        hdcMem ;
    static int        cxBitmap, cyBitmap, cxClient, cyClient, xMouse,
yMouse ;
    HDC                hdc ;
    PAINTSTRUCT        ps ;

    switch (message)
    {
    case WM_CREATE:
        GetLargestDisplayMode (&cxBitmap, &cyBitmap) ;

        hdc = GetDC (hwnd) ;
        hBitmap = CreateCompatibleBitmap (hdc, cxBitmap,
cyBitmap) ;

        hdcMem = CreateCompatibleDC (hdc) ;
        ReleaseDC (hwnd, hdc) ;

        if (!hBitmap) // no memory for
bitmap
        {
            DeleteDC (hdcMem) ;
            return -1 ;
        }

        SelectObject (hdcMem, hBitmap) ;
        PatBlt (hdcMem, 0, 0, cxBitmap, cyBitmap, WHITENESS) ;
        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_LBUTTONDOWN:
        if (!fRightButtonDown)
            SetCapture (hwnd) ;

        xMouse = LOWORD (lParam) ;
        yMouse = HIWORD (lParam) ;
        fLeftButtonDown = TRUE ;
        return 0 ;

    case WM_LBUTTONUP:
        if (fLeftButtonDown)
            SetCapture (NULL) ;
    }
```

```
        fLeftButtonDown = FALSE ;
        return 0 ;

case WM_RBUTTONDOWN:
    if (!fLeftButtonDown)
        SetCapture (hwnd) ;

    xMouse = LOWORD (lParam) ;
    yMouse = HIWORD (lParam) ;
    fRightButtonDown = TRUE ;
    return 0 ;

case WM_RBUTTONUP:
    if (fRightButtonDown)
        SetCapture (NULL) ;

    fRightButtonDown = FALSE ;
    return 0 ;

case WM_MOUSEMOVE:
    if (!fLeftButtonDown && !fRightButtonDown)
        return 0 ;

    hdc = GetDC (hwnd) ;

    SelectObject (hdc,
        GetStockObject (fLeftButtonDown ? BLACK_PEN :
WHITE_PEN)) ;

    SelectObject (hdcMem,
        GetStockObject (fLeftButtonDown ? BLACK_PEN :
WHITE_PEN)) ;

    MoveToEx (hdc,    xMouse, yMouse, NULL) ;
    MoveToEx (hdcMem, xMouse, yMouse, NULL) ;

    xMouse = (short) LOWORD (lParam) ;
    yMouse = (short) HIWORD (lParam) ;

    LineTo (hdc,    xMouse, yMouse) ;
    LineTo (hdcMem, xMouse, yMouse) ;

    ReleaseDC (hwnd, hdc) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
```

```

        BitBlt (hdc, 0, 0, cxClient, cyClient, hdcMem, 0, 0,
SRCCOPY) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        DeleteDC (hdcMem) ;
        DeleteObject (hBitmap) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

要想在 SKETCH 中画线，请按下滑鼠左键并拖动滑鼠。要擦掉画过的东西（更确切地说，是画白线），请按下滑鼠右键并拖动滑鼠。要清空整个视窗，请结束程式，然後重新载入，一切从头再来。图 14-5 中显示的 SKETCH 程式图样表达了对苹果公司的麦金塔电脑早期广告的敬意。

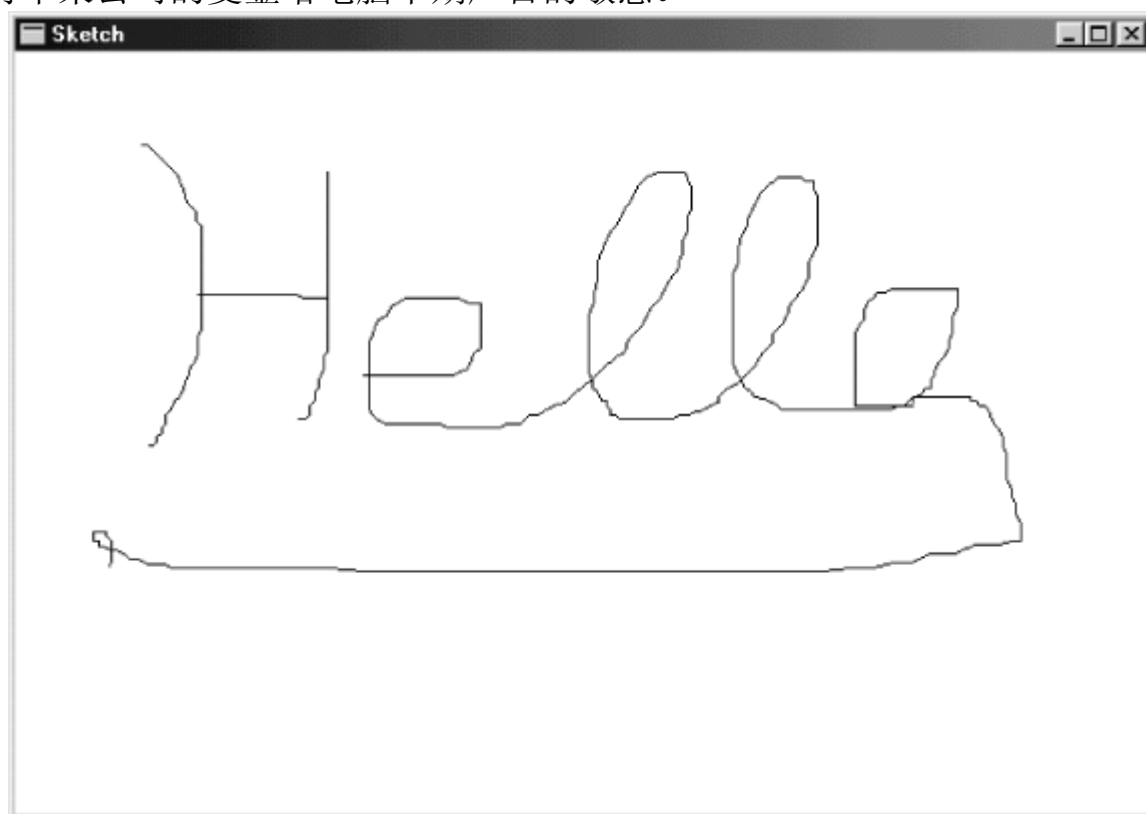


图 14-5 SKETCH 的萤幕显示

此阴影点阵图应多大？在本程式中，它应该大到能包含最大化视窗的整个显示区域。这一问题很容易根据 `GetSystemMetrics` 资讯计算得出，但如果使用者修改了显示设定後再显示，进而扩大了最大化时视窗的尺寸，这时将发生什么呢？SKETCH 程式在 `EnumDisplaySettings` 函式的帮助下解决了此问题。此函式使用 `DEVMODE` 结构来传回全部有效视讯显示模式的资讯。第一次呼叫此函式

时，应将 EnumDisplaySettings 的第二参数设为 0，以後每次呼叫此值都增加。EnumDisplaySettings 传回 FALSE 时完成。

与此同时，SKETCH 将建立一个阴影点阵图，它比目前视讯显示模式的表面还多四倍，而且需要几百万位元组的记忆体。由於如此，SKETCH 将检查点阵图是否建立成功了，如果没有建立，就从 WM_CREATE 传回-1，以表示错误。

在 WM_MOUSEMOVE 讯息处理期间，按下滑鼠左键或者右键，并在记忆体装置内容和显示区域装置内容中画线时，SKETCH 拦截滑鼠。如果画线方式更复杂的话，您可能想在一个函式中实作，程式将呼叫此函式两次——一次画在视讯装置内容上，一次画在记忆体装置内容上。

下面是一个有趣的实验：使 SKETCH 视窗小於全画面尺寸。随著滑鼠左键的按下，将滑鼠拖出视窗的右下角。因为 SKETCH 拦截滑鼠，所以它继续接收并处理 WM_MOUSEMOVE 讯息。现在扩大视窗，您将看到阴影点阵图包含您在 SKETCH 视窗外所画的内容。

在功能表中使用点阵图

您也可以用点阵图在功能表上显示选项。如果您联想起功能表中档案夹、剪贴簿和资源回收筒的图片，那么不要再想那些图片了。您应该考虑一下，功能表上显示点阵图对画图程式用途有多大，想像一下在功能表中使用不同字体和字体大小、线宽、阴影图案以及颜色。

GRAFMENU 是展示图形功能表选项的范例程式。此程式顶层功能表如图 14-6 所示。放大的字母来自於 40×16 图素的单色点阵图档案，该档案在 Visual C++ Developer Studio 建立。从功能表上选择「FONT」将弹出三个选择项——「Courier New」、「Arial」和「Times New Roman」。它们是标准的 Windows TrueType 字体，并且每一个都按其相关的字体显示，如图 14-7 所示。这些点阵图在程式中用记忆体装置内容建立。

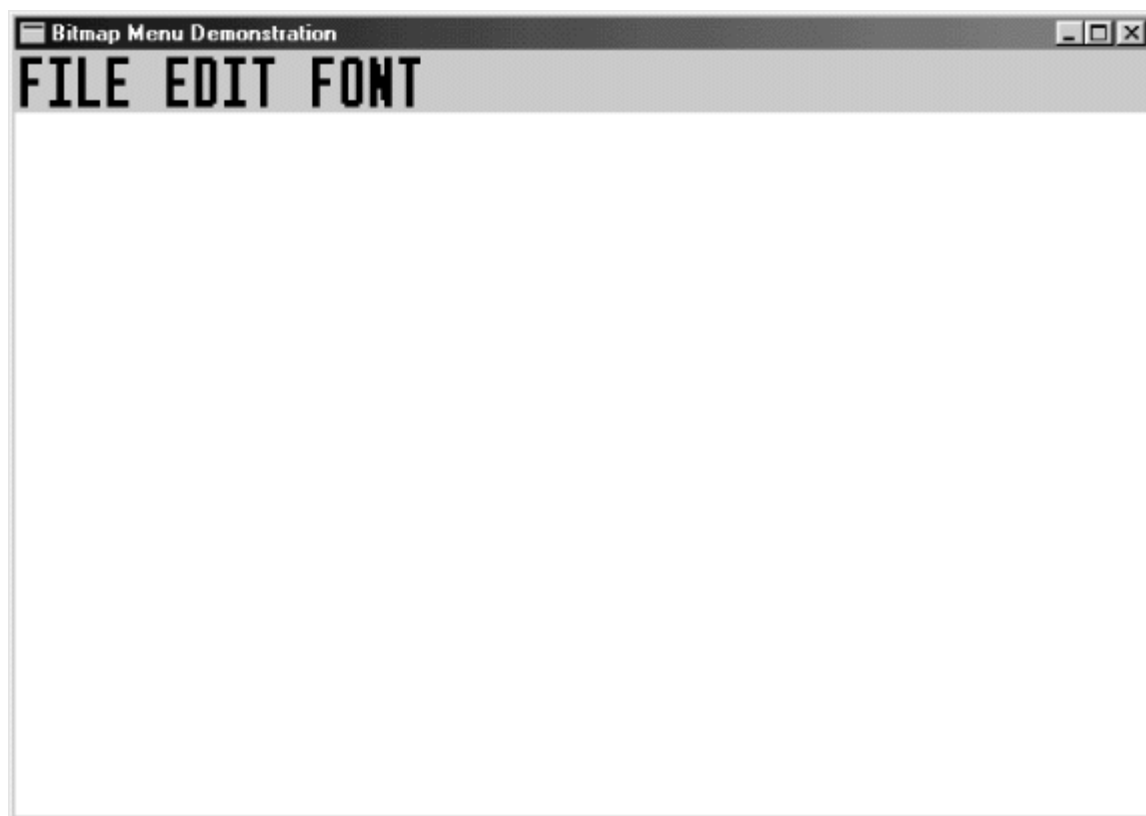


图 14-6 GRAFMENU 程式的顶层功能表

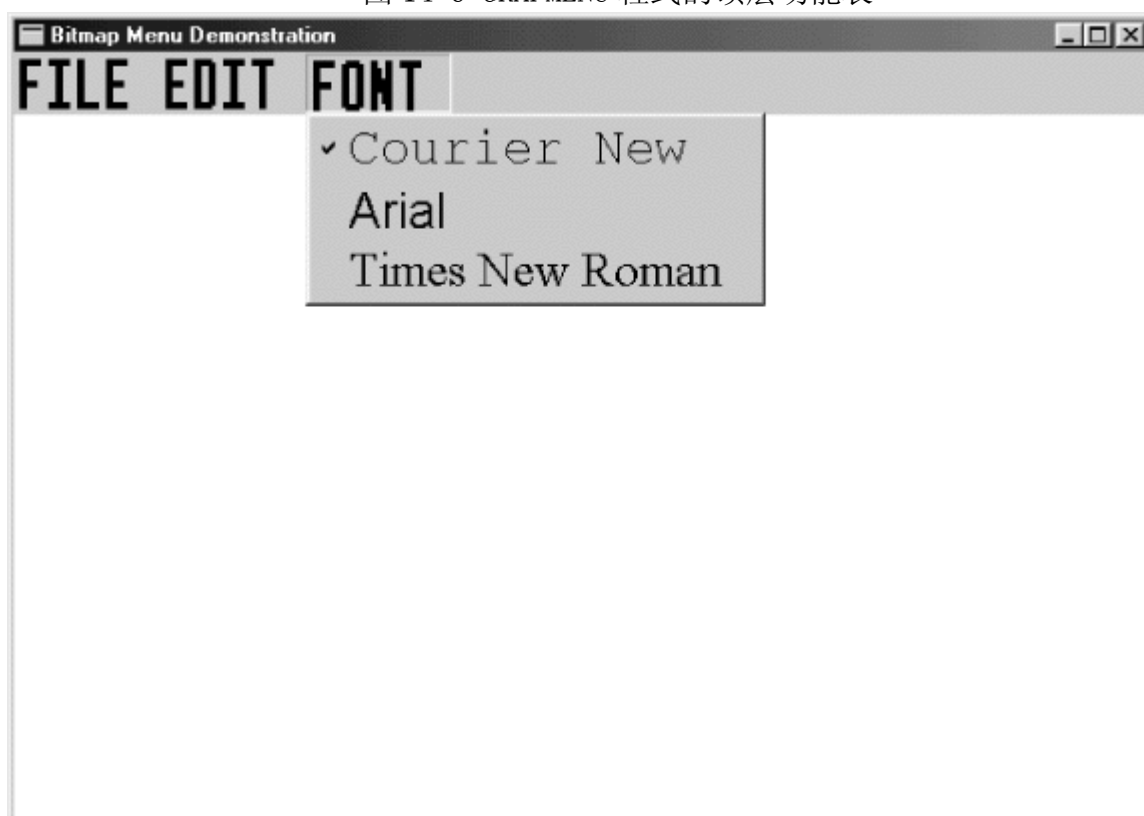


图 14-7 GRAFMENU 程式弹出的「FONT」功能表

最後，在拉下系统功能表时，您将获得一些「辅助」资讯，用「HELP」表示了新使用者的线上求助项目（参见图 14-8）。此 64×64 图素的单色点阵图是在 Developer Studio 中建立的。

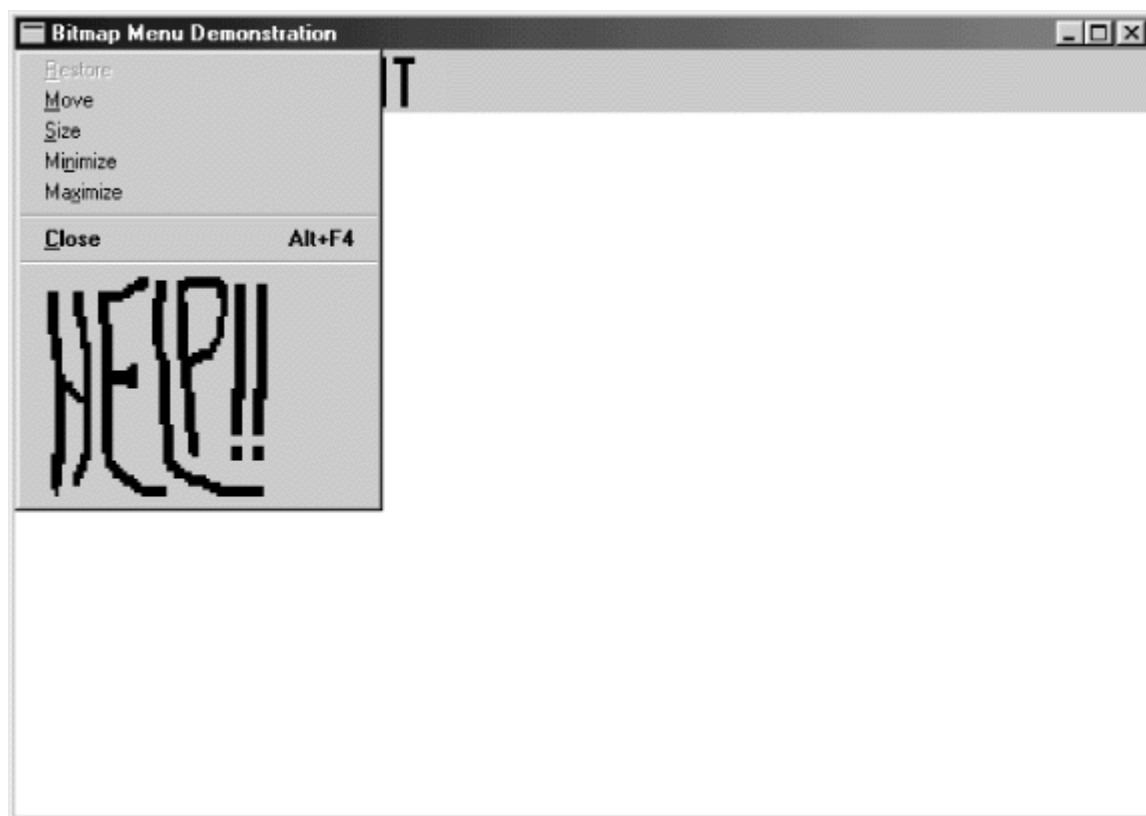


图 14-8 GRAFMENU 程式系统功能表

GRAFMENU 程式，包括四个 Developer Studio 中建立的点阵图，如程式 14-8 所示。

程式 14-8 GRAFMENU

```

GRAFMENU.C
/*-----
--
--          GRAFMENU.C --          Demonstrates Bitmap Menu Items
--                                     (c) Charles Petzold, 1998
--*/

#include <windows.h>
#include "resource.h"

LRESULT          CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
void             AddHelpToSys              (HINSTANCE, HWND) ;
HMENU            CreateMyMenu              (HINSTANCE) ;
HBITMAP          StretchBitmap             (HBITMAP) ;
HBITMAP          GetBitmapFont             (int) ;
void             DeleteAllBitmaps          (HWND) ;
TCHAR szAppName[] = TEXT ("GrafMenu") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;

```



```

MSG                msg ;
WNDCLASS           wndclass ;

wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc = WndProc ;
wndclass.cbClsExtra  = 0 ;
wndclass.cbWndExtra  = 0 ;
wndclass.hInstance  = hInstance ;
wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName = NULL ;
wndclass.lpszClassName = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Bitmap Menu Demonstration"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HMENU                hMenu ;
    static int           iCurrentFont = IDM_FONT_COUR ;

    switch (iMsg)
    {
    case WM_CREATE:
        AddHelpToSys    (((LPCREATESTRUCT)    lParam)->hInstance,

```

```

hwnd) ;

                                hMenu      =      CreateMyMenu      (((LPCREATESTRUCT)
lParam)->hInstance) ;

                                SetMenu (hwnd, hMenu) ;
                                CheckMenuItem (hMenu, iCurrentFont, MF_CHECKED) ;
                                return 0 ;

    case WM_SYSCOMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_HELP:
                MessageBox (hwnd, TEXT ("Help not yet
implemented!"),
                            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                return 0 ;
        }
        break ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_FILE_NEW:
            case IDM_FILE_OPEN:
            case IDM_FILE_SAVE:
            case IDM_FILE_SAVE_AS:
            case IDM_EDIT_UNDO:
            case IDM_EDIT_CUT:
            case IDM_EDIT_COPY:
            case IDM_EDIT_PASTE:
            case IDM_EDIT_CLEAR:
                MessageBeep (0) ;
                return 0 ;

            case IDM_FONT_COUR:
            case IDM_FONT_ARIAL:
            case IDM_FONT_TIMES:
                hMenu = GetMenu (hwnd) ;
                CheckMenuItem (hMenu, iCurrentFont, MF_UNCHECKED) ;
                iCurrentFont = LOWORD (wParam) ;
                CheckMenuItem (hMenu, iCurrentFont,
MF_CHECKED) ;

                return 0 ;
        }
        break ;

    case WM_DESTROY:
        DeleteAllBitmaps (hwnd) ;
        PostQuitMessage (0) ;

```

```

        return 0 ;

    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

/*-----
    AddHelpToSys: Adds bitmap Help item to system menu
    -----*/

void AddHelpToSys (HINSTANCE hInstance, HWND hwnd)
{
    HBITMAP      hBitmap ;
    HMENU        hMenu ;

    hMenu = GetSystemMenu (hwnd, FALSE);
    hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapHelp"))) ;
    AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
    AppendMenu (hMenu, MF_BITMAP, IDM_HELP, (PTSTR) (LONG) hBitmap) ;
}

/*-----
    CreateMyMenu: Assembles menu from components
    -----*/

HMENU CreateMyMenu (HINSTANCE hInstance)
{
    HBITMAP      hBitmap ;
    HMENU        hMenu, hMenuPopup ;
    int          i ;

    hMenu = CreateMenu () ;
    hMenuPopup = LoadMenu (hInstance, TEXT ("MenuFile")) ;
    hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFile"))) ;
    AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
                (PTSTR) (LONG) hBitmap) ;
    hMenuPopup = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
    hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapEdit"))) ;
    AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
                (PTSTR) (LONG) hBitmap) ;
    hMenuPopup = CreateMenu () ;
    for (i = 0 ; i < 3 ; i++)
    {
        hBitmap = GetBitmapFont (i) ;
        AppendMenu (hMenuPopup, MF_BITMAP, IDM_FONT_COUR + i,
                    (PTSTR) (LONG) hBitmap) ;
    }

    hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFont"))) ;

```

```

AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
            (PTSTR) (LONG) hBitmap) ;
return hMenu ;
}

/*-----
   StretchBitmap: Scales bitmap to display resolution
-----*/

/

HBITMAP StretchBitmap (HBITMAP hBitmap1)
{
    BITMAP          bm1, bm2 ;
    HBITMAP         hBitmap2 ;
    HDC             hdc, hdcMem1, hdcMem2 ;
    int             cxChar, cyChar ;

    // Get the width and height of a system font character

    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;

    // Create 2 memory DCs compatible with the display
    hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
    hdcMem1 = CreateCompatibleDC (hdc) ;
    hdcMem2 = CreateCompatibleDC (hdc) ;
    DeleteDC (hdc) ;

    // Get the dimensions of the bitmap to be stretched
    GetObject (hBitmap1, sizeof (BITMAP), (PTSTR) &bm1) ;
    // Scale these dimensions based on the system font size
    bm2 = bm1 ;
    bm2.bmWidth      = (cxChar * bm1.bmWidth) / 4 ;
    bm2.bmHeight     = (cyChar * bm1.bmHeight) / 8 ;
    bm2.bmWidthBytes = ((bm2.bmWidth + 15) / 16) * 2 ;

    // Create a new bitmap of larger size

    hBitmap2 = CreateBitmapIndirect (&bm2) ;
    // Select the bitmaps in the memory DCs and do a StretchBlt
    SelectObject (hdcMem1, hBitmap1) ;
    SelectObject (hdcMem2, hBitmap2) ;
    StretchBlt (hdcMem2, 0, 0, bm2.bmWidth, bm2.bmHeight,
                hdcMem1, 0, 0, bm1.bmWidth, bm1.bmHeight, SRCCOPY) ;
    // Clean up
    DeleteDC (hdcMem1) ;
    DeleteDC (hdcMem2) ;
    DeleteObject (hBitmap1) ;
}

```

```

        return hBitmap2 ;
    }

/*-----
-
    GetBitmapFont: Creates bitmaps with font names
-----
-*/

HBITMAP GetBitmapFont (int i)
{
    static TCHAR * szFaceName[3]= {    TEXT ("Courier New"), TEXT ("Arial"),
        TEXT ("Times New Roman") } ;
    HBITMAP          hBitmap ;
    HDC               hdc, hdcMem ;
    HFONT             hFont ;
    SIZE              size ;
    TEXTMETRIC        tm ;

    hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
    GetTextMetrics (hdc, &tm) ;

    hdcMem          = CreateCompatibleDC (hdc) ;
    hFont            = CreateFont (2 * tm.tmHeight, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                                szFaceName[i]) ;

    hFont = (HFONT) SelectObject (hdcMem, hFont) ;
    GetTextExtentPoint32 (hdcMem, szFaceName[i],
        lstrlen (szFaceName[i]), &size);
    hBitmap = CreateBitmap (size.cx, size.cy, 1, 1, NULL) ;
    SelectObject (hdcMem, hBitmap) ;

    TextOut (hdcMem, 0, 0, szFaceName[i], lstrlen (szFaceName[i])) ;
    DeleteObject (SelectObject (hdcMem, hFont)) ;
    DeleteDC (hdcMem) ;
    DeleteDC (hdc) ;

    return hBitmap ;
}

/*-----
-
    DeleteAllBitmaps: Deletes all the bitmaps in the menu
-----
-*/

```

```

void DeleteAllBitmaps (HWND hwnd)
{
    HMENU          hMenu ;
    int             i ;
    MENUITEMINFO mii = { sizeof (MENUITEMINFO), MIIM_SUBMENU | MIIM_TYPE } ;
    // Delete Help bitmap on system menu
    hMenu = GetSystemMenu (hwnd, FALSE);
    GetMenuItemInfo (hMenu, IDM_HELP, FALSE, &mii) ;
    DeleteObject ((HBITMAP) mii.dwTypeData) ;

    // Delete top-level menu bitmaps
    hMenu = GetMenu (hwnd) ;
    for (i = 0 ; i < 3 ; i++)
    {
        GetMenuItemInfo (hMenu, i, TRUE, &mii) ;
        DeleteObject ((HBITMAP) mii.dwTypeData) ;
    }

    // Delete bitmap items on Font menu
    hMenu = mii.hSubMenu ;;
    for (i = 0 ; i < 3 ; i++)
    {
        GetMenuItemInfo (hMenu, i, TRUE, &mii) ;
        DeleteObject ((HBITMAP) mii.dwTypeData) ;
    }
}

```

GRAFMENU.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

MENUEFILE MENU DISCARDABLE

BEGIN

MENUITEM "&New",	IDM_FILE_NEW
MENUITEM "&Open...",	IDM_FILE_OPEN
MENUITEM "&Save",	IDM_FILE_SAVE
MENUITEM "Save &As...",	IDM_FILE_SAVE_AS

END

MENUEEDIT MENU DISCARDABLE

BEGIN

MENUITEM "&Undo",	IDM_EDIT_UNDO
MENUITEM SEPARATOR	
MENUITEM "Cu&t",	IDM_EDIT_CUT
MENUITEM "&Copy",	IDM_EDIT_COPY
MENUITEM "&Paste",	IDM_EDIT_PASTE

```

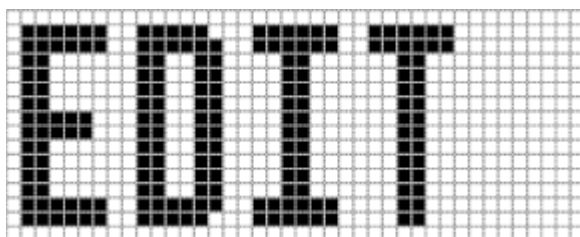
        MENUITEM "De&lete",          IDM_EDIT_CLEAR
END

////////////////////////////////////
/
// Bitmap
BITMAPFONT      BITMAP      DISCARDABLE      "Fontlabl.bmp"
BITMAPHELP      BITMAP      DISCARDABLE      "Bighelp.bmp"
BITMAPEDIT      BITMAP      DISCARDABLE      "Editlabl.bmp"
BITMAPFILE      BITMAP      DISCARDABLE      "Filelabl.bmp"
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by GrafMenu.rc

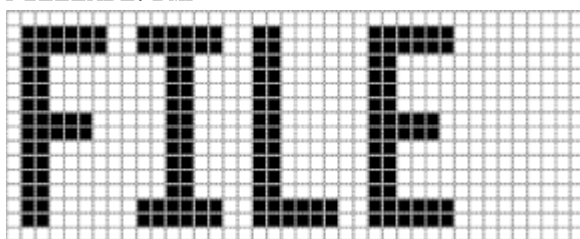
#define IDM_FONT_COUR      101
#define IDM_FONT_ARIAL      102
#define IDM_FONT_TIMES      103
#define IDM_HELP          104
#define IDM_EDIT_UNDO      40005
#define IDM_EDIT_CUT      40006
#define IDM_EDIT_COPY      40007
#define IDM_EDIT_PASTE      40008
#define IDM_EDIT_CLEAR      40009
#define IDM_FILE_NEW      40010
#define IDM_FILE_OPEN      40011
#define IDM_FILE_SAVE      40012
#define IDM_FILE_SAVE_AS 40013

```

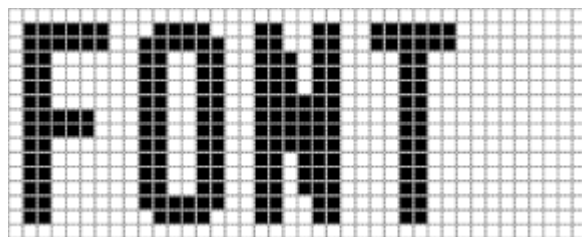
EDITLABL. BMP



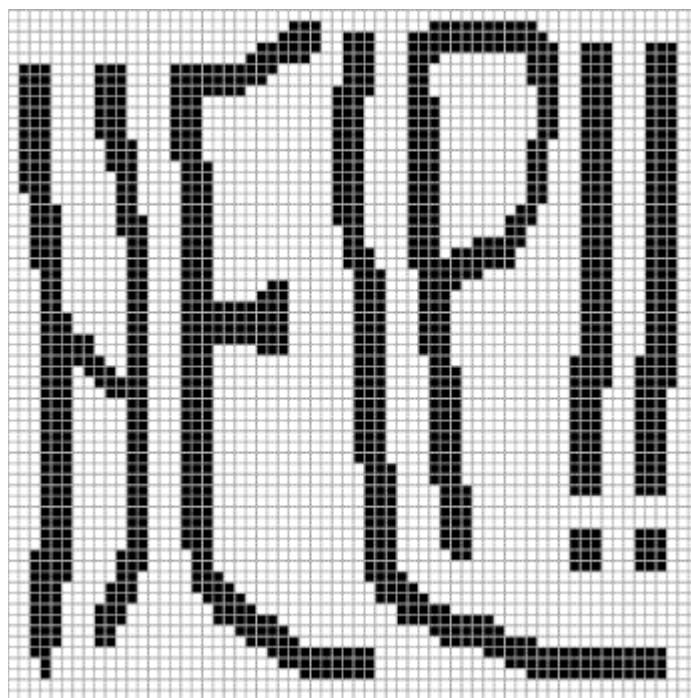
FILELABL. BMP



FONTLABL. BMP



BIGHELP.BMP



要将点阵图插入功能表，可以利用 `AppendMenu` 或 `InsertMenu`。点阵图有两个来源：可以在 Visual C++ Developer Studio 建立点阵图，包括资源脚本中的点阵图档案，并在程式使用 `LoadBitmap` 时将点阵图资源载入到记忆体，然後呼叫 `AppendMenu` 或 `InsertMenu` 将点阵图附加到功能表上。但是用这种方法会有一些问题：点阵图不适於所有显示模式的解析度和纵横比；有时您需要缩放载入的点阵图以解决此问题。另一种方法是：在程式内部建立点阵图，并将它选进记忆体装置内容，画出来，然後再附加到功能表中。

GRAFMENU 中的 `GetBitmapFont` 函式的参数为 0、1 或 2，传回一个点阵图代号。此点阵图包含字串「Courier New」、「Arial」或「Times New Roman」，而且字体是各自对应的字体，大小是正常系统字体的两倍。让我们看看 `GetBitmapFont` 是怎么做的。（下面的程式码与 GRAFMENU.C 档案中的有些不同。为了清楚起见，我用「Arial」字体相应的值代替了引用 `szFaceName` 阵列。）

第一步是用 `TEXTMETRIC` 结构来确定目前系统字体的大小，并建立一个与目前萤幕相容的记忆体装置内容：

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;  
GetTextMetrics (hdc, &tm) ;  
hdcMem = CreateCompatibleDC (hdc) ;
```


CreateFont 函式建立了一种逻辑字体，该字体高是系统字体的两倍，而且逻辑名称为「Arial」：

```
hFont = CreateFont (2 * tm.tmHeight, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  TEXT ("Arial")) ;
```

从记忆体装置内容中选择该字体，然後储存内定字体代号：

```
hFont = (HFONT) SelectObject (hdcMem, hFont) ;
```

现在，当我们向记忆体装置内容写一些文字时，Windows 就会使用选进装置内容的 TrueType Arial 字体了。

但这个记忆体装置内容最初只有一个单图素单色设备平面。我们必须建立一个足够大的点阵图以容纳我们所要显示的文字。通过 GetTextExtentPoint32 函式，可以取得文字的大小，而用 CreateBitmap 可以根据这些尺寸来建立点阵图：

```
GetTextExtentPoint32 (hdcMem, TEXT ("Arial"), 5, &size) ;
hBitmap = CreateBitmap (size.cx, size.cy, 1, 1, NULL) ;
SelectObject (hdcMem, hBitmap) ;
```

现在这个装置内容是一个单色的显示平面，大小也是严格的文字尺寸。我们现在要做的就是书写文字：

```
TextOut (hdcMem, 0, 0, TEXT ("Arial"), 5) ;
```

除了清除，所有的工作都完成了。要清除，我们可以用 SelectObject 将系统字体（带有代号 hFont）重新选进装置内容，然後删除 SelectObject 传回的前一个字体代号，也就是 Arial 字体代号：

```
DeleteObject (SelectObject (hdcMem, hFont)) ;
```

现在可以删除两个装置内容：

```
DeleteDC (hdcMem) ;
DeleteDC (hdc) ;
```

这样，我们就获得了一个点阵图，该点阵图上有 Arial 字体的字串「Arial」。

当我们需要缩放字体以适应不同显示解析度或纵横比时，记忆体装置内容也能解决问题。在 GRAFMENU 程式中，我建立了四个点阵图，这些点阵图只适用於系统字体高 8 图素、宽 4 图素的显示。對於其他尺寸的系统字体，只能缩放点阵图。GRAFMENU 中的 StretchBitmap 函式完成此功能。

第一步是获得显示的装置内容，然後取得系统字体的文字规格，接下来建立两个记忆体装置内容：

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
GetTextMetrics (hdc, &tm) ;
hdcMem1 = CreateCompatibleDC (hdc) ;
hdcMem2 = CreateCompatibleDC (hdc) ;
DeleteDC (hdc) ;
```

传递给函式的点阵图代号是 hBitmap1。程式能用 GetObject 获得点阵图的

大小:

```
GetObject (hBitmap1, sizeof (BITMAP), (PSTR) &bm1) ;
```

此操作将尺寸复制到 BITMAP 型态的结构 bm1 中。结构 bm2 等於结构 bm1, 然後根据系统字体大小来修改某些栏位:

```
bm2 = bm1 ;
bm2.bmWidth      = (tm.tmAveCharWidth * bm2.bmWidth) / 4 ;
bm2.bmHeight     = (tm.tmHeight * bm2.bmHeight) / 8 ;
bm2.bmWidthBytes = ((bm2.bmWidth + 15) / 16) * 2 ;
```

下一个点阵图带有代号 hBitmap2, 可以根据动态的尺寸建立:

```
hBitmap2 = CreateBitmapIndirect (&bm2) ;
```

然後将这两个点阵图选进两个记忆体装置内容中:

```
SelectObject (hdcMem1, hBitmap1) ;
SelectObject (hdcMem2, hBitmap2) ;
```

我们想把第一个点阵图复制给第二个点阵图, 并在此程序中进行拉伸。这包括 StretchBlt 呼叫:

```
StretchBlt (      hdcMem2, 0, 0, bm2.bmWidth, bm2.bmHeight,
                  hdcMem1, 0, 0, bm1.bmWidth, bm1.bmHeight, SRCCOPY) ;
```

现在第二幅图适当地缩放了, 我们可将其用到功能表中。剩下的清除工作很简单:

```
DeleteDC (hdcMem1) ;
DeleteDC (hdcMem2) ;
DeleteObject (hBitmap1) ;
```

在建造功能表时, GRAFMENU 中的 CreateMyMenu 函式呼叫了 StretchBitmap 和 GetBitmapFont 函式。GRAFMENU 在资源档案中定义了两个功能表, 在选择「File」和「Edit」选项时会弹出这两个功能表。函式开始先取得一个空功能表的代号:

```
hMenu = CreateMenu () ;
```

从资源档案载入「File」的突现式功能表(包括四个选项:「New」、「Open」、「Save」和「Save as」):

```
hMenuPopup = LoadMenu (hInstance, TEXT ("MenuFile")) ;
```

从资源档案还载入了包含「FILE」的点阵图, 并用 StretchBitmap 进行了拉伸:

```
hBitmapFile = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFile"))) ;
```

点阵图代号和突现式功能表代号都是 AppendMenu 呼叫的参数:

```
AppendMenu (hMenu, MF_BITMAP | MF_POPUP, hMenuPopup, (PTSTR) (LONG)
hBitmapFile) ;
```

「Edit」功能表类似程序如下:

```
hMenuPopup = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
hBitmapEdit = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapEdit"))) ;
AppendMenu (hMenu, MF_BITMAP | MF_POPUP, hMenuPopup, (PTSTR) (LONG) hBitmapEdit) ;
```

呼叫 GetBitmapFont 函式可以构造这三种不同字体的突现式功能表:

```
hMenuPopup = CreateMenu ();
for (i = 0 ; i < 3 ; i++)
{
    hBitmapPopFont [i] = GetBitmapFont (i) ;
    AppendMenu (hMenuPopup, MF_BITMAP, IDM_FONT_COUR + i,
                (PTSTR) (LONG) hMenuPopupFont [i]) ;
}
```

然後将突现式功能表添加到功能表中:

```
hBitmapFont = StretchBitmap (LoadBitmap (hInstance, "BitmapFont")) ;
AppendMenu (      hMenu, MF_BITMAP | MF_POPUP, hMenuPopup, (PTSTR) (LONG)
                hBitmapFont) ;
```

WndProc 通过呼叫 SetMenu, 完成了视窗功能表的建立工作。

GRAFMENU 还改变了 AddHelpToSys 函式中的系统功能表。此函式首先获得一个系统功能表代号:

```
hMenu = GetSystemMenu (hwnd, FALSE) ;
```

这将载入「HELP」点阵图, 并将其拉伸到适当尺寸:

```
hBitmapHelp = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapHelp"))) ;
```

这将给系统功能表添加一条分隔线和拉伸的点阵图:

```
AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenu, MF_BITMAP, IDM_HELP, (PTSTR) (LONG) hBitmapHelp) ;
```

GRAFMENU 在退出之前呼叫一个函式来清除并删除所有点阵图。

下面是在功能表中使用点阵图的一些注意事项。

在顶层功能表中, Windows 调整功能表列的高度以适应最高的点阵图。其他点阵图(或字串)是根据功能表列的顶端对齐的。如果在顶层功能表中使用了点阵图, 那么从使用常数 SM_CYMENU 的 GetSystemMetrics 得到的功能表列大小将不再有效。

执行 GRAFMENU 期间可以看到: 在突现式功能表中, 您可使用带有点阵图功能表项的勾选标记, 但勾选标记是正常尺寸。如果不满意, 您可以建立一个自订的勾选标记, 并使用 SetMenuItemBitmaps。

在功能表中使用非文字(或者使用非系统字体的文字)的另一种方法是「拥有者绘制」功能表。

功能表的键盘介面是另一个问题。当功能表含有文字时, Windows 会自动添加键盘介面。要选择一个功能表项, 可以使用 Alt 与字串中的一个字母的组合键。而一旦在功能表中放置了点阵图, 就删除了键盘介面。即使点阵图表达了一定的含义, 但 Windows 并不知道。

目前我们可以使用 WM_MENUCHAR 讯息。当您按下 Alt 和与功能表项不相符的一个字元键的组合键时, Windows 将向您的视窗讯息处理程式发送一个

WM_MENUCHAR 讯息。GRAFMENU 需要截取 WM_MENUCHAR 讯息并检查 wParam 的值(即按键的 ASCII 码)。如果这个值对应一个功能表项,那么向 Windows 传回双字组:其中高字组为 2,低字组是与该键相关的功能表项索引值。然後由 Windows 处理余下的事。

非矩形点阵图图像

点阵图都是矩形,但不需要都显示成矩形。例如,假定您有一个矩形点阵图图像,但您却想将它显示成椭圆形。

首先,这听起来很简单。您只需将图像载入 Visual C++ Developer Studio 或者 Windows 的「画图」程式(或者更昂贵的應用程式),然後用白色的画笔将图像四周画上白色。这时将获得一幅椭圆形的图像,而椭圆的外面就成了白色。只有当背景色为白色时此点阵图才能正确显示,如果在其他背景色上显示,您就会发现椭圆形的图像和背景之间有一个白色的矩形。这种效果不好。

有一种非常通用的技术可解决此类问题。这种技术包括「遮罩(mask)」点阵图和一些位元映射操作。遮罩是一种单色点阵图,它与您要显示的矩形点阵图图像尺寸相同。每个遮罩的图素都对应点阵图图像的一个图素。遮罩图素是 1(白色),对应著点阵图图素显示;是 0(黑色),则显示背景色。(或者遮罩点阵图与此相反,这根据您使用的位元映射操作而有一些相对应的变化。)

让我们看看 BITMASK 程式是如何实作这一技术的。如程式 14-9 所示。

程式 14-9 BITMASK

```
BITMASK.C
/*-----
-
    BITMASK.C --      Bitmap Masking Demonstration
                        (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR          szAppName [] = TEXT ("BitMask") ;
    HWND                  hwnd ;
    MSG                   msg ;
    WNDCLASS               wndclass ;

    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
```

```

    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance           = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (LTGRAY_BRUSH) ;
    wndclass.lpszMenuName         = NULL ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Bitmap Masking Demo"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP          hBitmapImag, hBitmapMask ;
    static HINSTANCE        hInstance ;
    static int              cxClient, cyClient, cxBitmap, cyBitmap ;
    static BITMAP            bitmap ;
    static HDC              hdc, hdcMemImag, hdcMemMask ;
    static int              x, y ;
    static PAINTSTRUCT       ps ;

    switch (message)
    {
    case WM_CREATE:
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
        // Load the original image and get its size

```

```

        hBitmapImag = LoadBitmap (hInstance, TEXT ("Matthew")) ;
        GetObject (hBitmapImag, sizeof (BITMAP), &bitmap) ;
        cxBitmap = bitmap.bmWidth ;
        cyBitmap = bitmap.bmHeight ;

        // Select the original image into a memory DC
        hdcMemImag = CreateCompatibleDC (NULL) ;
        SelectObject (hdcMemImag, hBitmapImag) ;
        // Create the monochrome mask bitmap and memory
DC
        hBitmapMask = CreateBitmap (cxBitmap, cyBitmap, 1, 1,
NULL) ;

        hdcMemMask = CreateCompatibleDC (NULL) ;
        SelectObject (hdcMemMask, hBitmapMask) ;

        // Color the mask bitmap black with a white
ellipse
        SelectObject (hdcMemMask, GetStockObject
(BLACK_BRUSH)) ;

        Rectangle (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
        SelectObject (hdcMemMask, GetStockObject
(WHITE_BRUSH)) ;

        Ellipse (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;

        // Mask the original image
        BitBlt (hdcMemImag, 0, 0, cxBitmap, cyBitmap,
                hdcMemMask, 0, 0, SRCAND) ;
        DeleteDC (hdcMemImag) ;
        DeleteDC (hdcMemMask) ;
        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        // Select bitmaps into memory DCs

        hdcMemImag = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMemImag, hBitmapImag) ;

        hdcMemMask = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMemMask, hBitmapMask) ;

        // Center image

```

```

        x = (cxClient - cxBitmap) / 2 ;
        y = (cyClient - cyBitmap) / 2 ;

        // Do the bitblts

        BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemMask, 0, 0,
0x220326) ;

        BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemImag, 0, 0,
SRCPAINT) ;

        DeleteDC (hdcMemImag) ;
        DeleteDC (hdcMemMask) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        DeleteObject (hBitmapImag) ;
        DeleteObject (hBitmapMask) ;
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BITMASK.RC
// Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Bitmap
MATTHEW          BITMAP          DISCARDABLE          "matthew.bmp"

```

资源档案中的 MATTHEW.BMP 档案是我侄子的一幅黑白数位照片，宽 200 图素，高 320 图素，每图素 8 位元。不过，另外制作个 BITMASK 只是因为此档案的内容是任何东西都可以。

注意，BITMASK 将视窗背景设为亮灰色。这样就确保我们能正确地遮罩点阵图，而不只是将其涂成白色。

下面让我们看一下 WM_CREATE 的处理程序：BITMASK 用 LoadBitmap 函式获得 hBitmapImag 变数中原始图像的代号。用 GetObject 函式可取得点阵图的宽度高度。然后将点阵图代号选进代号为 hdcMemImag 的记忆体装置内容中。

程式建立的下一个单色点阵图与原来的图大小相同，其代号储存在 hBitmapMask，并选进代号为 hdcMemMask 的记忆体装置内容中。在记忆体装置内容中，使用 GDI 函式，遮罩点阵图就涂成了黑色背景和一个白色的椭圆：

```
SelectObject (hdcMemMask, GetStockObject (BLACK_BRUSH)) ;
Rectangle (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
SelectObject (hdcMemMask, GetStockObject (WHITE_BRUSH)) ;
Ellipse (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
```

因为这是一个单色的点阵图，所以黑色区域的位元是 0，而白色区域的位元是 1。

然後 BitBlt 呼叫就按此遮罩修改了原图像：

```
BitBlt (hdcMemImag, 0, 0, cxBitmap, cyBitmap,
        hdcMemMask, 0, 0, SRCAND) ;
```

SRCAND 位元映射操作在来源位元（遮罩点阵图）和目的位元（原图像）之间执行了位元 AND 操作。只要遮罩点阵图是白色，就显示目的；只要遮罩是黑色，则目的就也是黑色。现在原图像中就形成了一个黑色包围的椭圆区域。

现在让我们看一下 WM_PAINT 处理程序。此程序同时改变了选进记忆体装置内容中的图像点阵图和遮罩点阵图。两次 BitBlt 呼叫完成了这个魔术，第一次在视窗上执行遮罩点阵图的 BitBlt：

```
BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemMask, 0, 0, 0x220326) ;
```

这里使用了一个没有名称的位元映射操作。逻辑运算符是 $D \& \sim S$ 。回忆来源——即遮罩点阵图——是黑色（位元值 0）包围的一个白色（位元值 1）椭圆。位元映射操作首先将来源反色，也就是改成白色包围的黑色椭圆。然後位元操作在这个已转换的来源和目的（即视窗上）之间执行位元 AND 操作。当目的和位元值 1「AND」时保持不变；与位元值 0「AND」时，目的将变黑。因此，BitBlt 操作将在视窗上画一个黑色的椭圆。

第二次的 BitBlt 呼叫则在视窗中绘制图像点阵图：

```
BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemImag, 0, 0, SRCPAINT) ;
```

位元映射操作在来源和目的之间执行位元「OR」操作。由於来源点阵图的外面是黑色，因此保持目的不变；而在椭圆区域内，目的是黑色，因此图像就原封不动地复制了过来。执行结果如图 14-9 所示。

注意事项：

有时您需要一个很复杂的遮罩——例如，抹去原始图像的整个背景。您将需要在画图程式中手工建立然後将其储存到成档案。

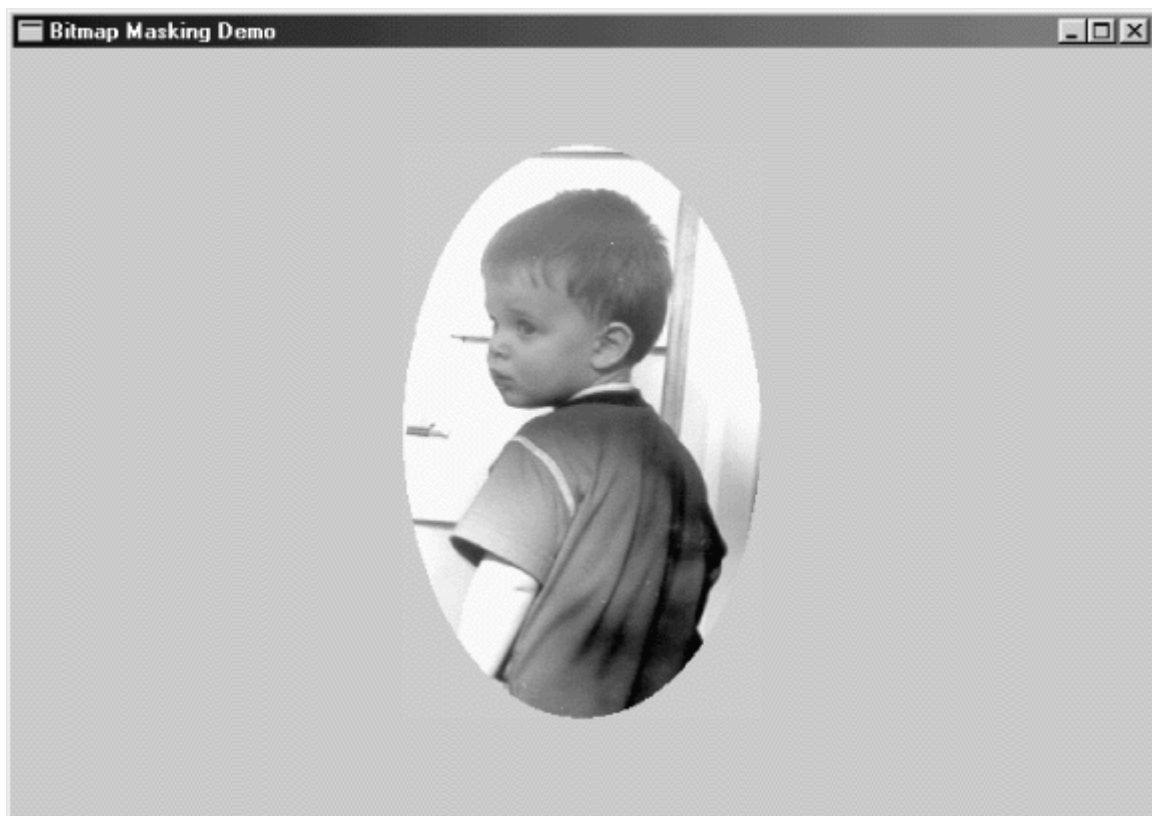


图 14-9 BITMASK 的萤幕显示

如果正在为 Windows NT 编写类似的应用程式，那么您可以使用与 MASKBIT 程式类似的 MaskBlt 函式，而只需要更少的函式呼叫。Windows NT 还包括另一个类似 BitBlt 的函式，Windows 98 不支援该函式。此函式是 PlgBlt（「平行四边形位元块移动：parallelogram blt」）。这个函式可以对图像进行旋转或者倾斜点阵图图像。

最後，如果在您的机器上执行 BITMASK 程式，您就只会看见黑色、白色和两个灰色的阴影，这是因为您执行的显示模式是 16 色或 256 色。对于 16 色模式，显示效果无法改进，但在 256 色模式下可以改变调色盘以显示灰阶。您将在第十六章学会如何设定调色盘。

简单的动画

小张的点阵图显示起来非常快，因此可以将点阵图和 Windows 计时器联合使用，来完成一些基本的动画。

现在开始这个弹球程式。

BOUNCE 程式，如程式 14-10 所示，产生了一个在视窗显示区域弹来弹去的小球。该程式利用计时器来控制小球的行进速度。小球本身是一幅点阵图，程式首先通过建立点阵图来建立小球，将其选进记忆体装置内容，然後呼叫一些简单的 GDI 函式。程式用 BitBlt 从一个记忆体装置内容将这个点阵图小球画到显示器上。

程式 14-10 BOUNCE

```

BOUNCE.C
/*-----
-
      BOUNCE.C --      Bouncing Ball Program
                                  (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#define ID_TIMER      1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Bounce") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
        = NULL ;
    wndclass.lpszClassName
        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Bouncing Ball"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

```

```

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP    hBitmap ;
    static int        cxClient, cyClient, xCenter, yCenter, cxTotal,
cyTotal,
                    cxRadius,          cyRadius,  cxMove,  cyMove,  xPixel,
yPixel ;
    HBRUSH            hBrush ;
    HDC               hdc, hdcMem ;
    int               iScale ;

    switch (iMsg)
    {
    case WM_CREATE:
        hdc = GetDC (hwnd) ;
        xPixel = GetDeviceCaps (hdc, ASPECTX) ;
        yPixel = GetDeviceCaps (hdc, ASPECTY) ;
        ReleaseDC (hwnd, hdc) ;

        SetTimer (hwnd, ID_TIMER, 50, NULL) ;
        return 0 ;

    case WM_SIZE:
        xCenter = (cxClient = LOWORD (lParam)) / 2 ;
        yCenter = (cyClient = HIWORD (lParam)) / 2 ;

        iScale = min (cxClient * xPixel, cyClient * yPixel) / 16 ;

        cxRadius = iScale / xPixel ;
        cyRadius = iScale / yPixel ;

        cxMove = max (1, cxRadius / 2) ;
        cyMove = max (1, cyRadius / 2) ;

        cxTotal = 2 * (cxRadius + cxMove) ;
        cyTotal = 2 * (cyRadius + cyMove) ;

        if (hBitmap)

```

```

        DeleteObject (hBitmap) ;
        hdc = GetDC (hwnd) ;
        hdcMem = CreateCompatibleDC (hdc) ;
        hBitmap=CreateCompatibleBitmap (hdc, cxTotal, cyTotal) ;
        ReleaseDC (hwnd, hdc) ;

        SelectObject (hdcMem, hBitmap) ;
        Rectangle (hdcMem, -1, -1, cxTotal + 1, cyTotal + 1) ;

        hBrush = CreateHatchBrush (HS_DIAGCROSS, 0L) ;
        SelectObject (hdcMem, hBrush) ;
        SetBkColor (hdcMem, RGB (255, 0, 255)) ;
        Ellipse (hdcMem, cxMove, cyMove, cxTotal - cxMove, cyTotal
- cyMove) ;

        DeleteDC (hdcMem) ;
        DeleteObject (hBrush) ;
        return 0 ;

    case WM_TIMER:
        if (!hBitmap)
            break ;

        hdc = GetDC (hwnd) ;
        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;

        BitBlt (hdc, xCenter - cxTotal / 2,
                yCenter - cyTotal / 2, cxTotal,
cyTotal,
                hdcMem, 0, 0, SRCCOPY) ;

        ReleaseDC (hwnd, hdc) ;
        DeleteDC (hdcMem) ;

        xCenter += cxMove ;
        yCenter += cyMove ;

        if ((xCenter + cxRadius >= cxClient) || (xCenter - cxRadius
<= 0))
            cxMove = -cxMove ;

        if ((yCenter + cyRadius >= cyClient) || (yCenter - cyRadius
<= 0))
            cyMove = -cyMove ;

        return 0 ;

    case WM_DESTROY:

```

```

        if (hBitmap)
            DeleteObject (hBitmap) ;

        KillTimer (hwnd, ID_TIMER) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

```

BOUNCE 每次收到一个 WM_SIZE 讯息时都重画小球。这就需要与视讯显示器相容的记忆体装置内容：

```
hdcMem = CreateCompatibleDC (hdc) ;
```

小球的直径设为视窗显示区域高度或宽度中较短者的十六分之一。不过，程式构造的点阵图却比小球大：从点阵图中心到点阵图四个边的距离是小球半径的 1.5 倍：

```
hBitmap = CreateCompatibleBitmap (hdc, cxTotal, cyTotal) ;
```

将点阵图选进记忆体装置内容後，整个点阵图背景设成白色：

```
Rectangle (hdcMem, -1, -1, xTotal + 1, yTotal + 1) ;
```

那些不固定的座标使矩形边框在点阵图之外著色。一个对角线开口的画刷选进记忆体装置内容，并将小球画在点阵图的中央：

```
Ellipse (hdcMem, xMove, yMove, xTotal - xMove, yTotal - yMove) ;
```

当小球移动时，小球边界的空白会有效地删除前一时刻的小球图像。在另一个位置重画小球只需在 BitBlt 呼叫中使用 SRCCOPY 的 ROP 代码：

```
BitBlt (hdc, xCenter - cxTotal / 2, yCenter - cyTotal / 2, cxTotal, cyTotal,
        hdcMem, 0, 0, SRCCOPY) ;
```

BOUNCE 程式只是展示了在显示器上移动图像的最简单的方法。在一般情况下，这种方法并不能令人满意。如果您对动画感兴趣，那么除了在来源和目的之间执行或操作以外，您还应该研究其他的 ROP 代码（例如 SRCINVERT）。其他动画技术包括 Windows 调色盘（以及 AnimatePalette 函式）和 CreateDIBSection 函式。对於更高级的动画您只好放弃 GDI 而使用 DirectX 介面了。

视窗外的点阵图

SCRAMBLE 程式，如程式 14-11 所示，编写非常粗糙，我本来不应该展示这个程式，但它示范了一些有趣的技术，而且在交换两个显示矩形内容的 BitBlt 操作的程序中，用记忆体装置内容作为临时储存空间。

程式 14-11 SCRAMBLE

```
SCRAMBLE.C
```

```

/*-----
-

```

```

SCRAMBLE.C -- Scramble (and Unscramble) Screen
                                           (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>

#define NUM 300

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static int      iKeep [NUM][4] ;
    HDC             hdcScr, hdcMem ;
    int             cx, cy ;
    HBITMAP         hBitmap ;
    HWND            hwnd ;
    int             i, j, x1, y1, x2, y2 ;

    if (LockWindowUpdate (hwnd = GetDesktopWindow ()))
    {
        hdcScr      =      GetDCEX      (hwnd,      NULL,      DCX_CACHE      |
DCX_LOCKWINDOWUPDATE) ;
        hdcMem       = CreateCompatibleDC (hdcScr) ;
        cx           = GetSystemMetrics (SM_CXSCREEN) / 10 ;
        cy           = GetSystemMetrics (SM_CYSCREEN) / 10 ;
        hBitmap = CreateCompatibleBitmap (hdcScr, cx, cy) ;

        SelectObject (hdcMem, hBitmap) ;
        srand ((int) GetCurrentTime ()) ;

        for (i = 0 ; i < 2 ; i++)
            for (j = 0 ; j < NUM ; j++)
            {
                if (i == 0)
                {
                    iKeep [j] [0] = x1 = cx * (rand () % 10) ;
                    iKeep [j] [1] = y1 = cy * (rand () % 10) ;
                    iKeep [j] [2] = x2 = cx * (rand () % 10) ;
                    iKeep [j] [3] = y2 = cy * (rand () % 10) ;
                }
                else
                {
                    x1 = iKeep [NUM - 1 - j] [0] ;
                    y1 = iKeep [NUM - 1 - j] [1] ;

```

```

                x2 = iKeep [NUM - 1 - j] [2] ;
                y2 = iKeep [NUM - 1 - j] [3] ;
            }
            BitBlt (hdcMem, 0, 0, cx, cy, hdcScr, x1, y1,
SRCCOPY) ;

            BitBlt (hdcScr, x1, y1, cx, cy, hdcScr, x2, y2,
SRCCOPY) ;

            BitBlt (hdcScr, x2, y2, cx, cy, hdcMem, 0, 0,
SRCCOPY) ;

            Sleep (10) ;
        }

        DeleteDC (hdcMem) ;
        ReleaseDC (hwnd, hdcScr) ;
        DeleteObject (hBitmap) ;

        LockWindowUpdate (NULL) ;
    }
    return FALSE ;
}

```

SCRAMBLE 没有视窗讯息处理程式。在 WinMain 中，它首先呼叫带有桌面视窗代号的 LockWindowUpdate。此函式暂时防止其他程式更新萤幕。然後 SCRAMBLE 通过呼叫带有参数 DCX_LOCKWINDOWUPDATE 的 GetDCEX 来获得整个萤幕的装置内容。这样就只有 SCRAMBLE 可以更新萤幕了。

然後 SCRAMBLE 确定全萤幕的尺寸，并将长宽分别除以 10。程式用这个尺寸（名称是 cx 和 cy）来建立一个点阵图，并将该点阵图选进记忆体装置内容。

使用 C 语言的 rand 函式，SCRAMBLE 计算出四个随机值（两个座标点）作为 cx 和 cy 的倍数。程式透过三次呼叫 BitBlt 函式来交换两个矩形块中显示的内容。第一次将从第一个座标点开始的矩形复制到记忆体装置内容。第二次 BitBlt 将从第二座标点开始的矩形复制到第一点开始的位置。第三次将记忆体装置内容中的矩形复制到第二个座标点开始的区域。

此程序将有效地交换显示器上两个矩形中的内容。SCRAMBLE 执行 300 次交换，这时的萤幕显示肯定是一团糟。但不用担心，因为 SCRAMBLE 记得是怎么把显示弄得这样一团糟的，接著在退出前它会按相反的次序恢复原来的桌面显示（锁定萤幕前的画面）！

您也可以使用记忆体装置内容将一个点阵图复制给另一个点阵图。例如，假定您要建立一个点阵图，该点阵图只包含另一个点阵图左上角的图形。如果原来的图像代号为 hBitmap，那么您可以将其尺寸复制到一个 BITMAP 型态的结构中：

```
GetObject (hBitmap, sizeof (BITMAP), &bm) ;
```

然後建立一个未初始化的新点阵图，该点阵图的尺寸是原来图的 1/4：

```
hBitmap2 = CreateBitmap (    bm.bmWidth / 2, bm.bmHeight / 2,
                           bm.bmPlanes, bm.bmBitsPixel, NULL) ;
```

现在建立两个记忆体装置内容，并将原来点阵图和新点阵图选分别进这两个记忆体装置内容：

```
hdcMem1 = CreateCompatibleDC (hdc) ;
hdcMem2 = CreateCompatibleDC (hdc) ;

SelectObject (hdcMem1, hBitmap) ;
SelectObject (hdcMem2, hBitmap2) ;
```

最後，将第一个点阵图的左上角复制给第二个：

```
BitBlt (    hdcMem2, 0, 0, bm.bmWidth / 2, bm.bmHeight / 2,
           hdcMem1, 0, 0, SRCCOPY) ;
```

剩下的只是清除工作：

```
DeleteDC (hdcMem1) ;
DeleteDC (hdcMem2) ;
DeleteObject (hBitmap) ;
```

BLOWUP.C 程式，如图 14-21 所示，也用视窗更新锁定来在程式视窗之外显示一个捕捉的矩形。此程式允许使用者用滑鼠圈选萤幕上的矩形区域，然後 BLOWUP 将该区域的内容复制到点阵图。在 WM_PAINT 讯息处理期间，点阵图复制到程式的显示区域，必要时将拉伸或压缩。（参见程式 14-12。）

程式 14-12 BLOWUP

```
BLOWUP.C
/*-----
    BLOWUP.C --      Video Magnifier Program
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <stdlib.h>                // for abs definition
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR          szAppName [] = TEXT ("Blowup") ;
    HACCEL                hAccel ;
    HWND                  hwnd ;
    MSG                   msg ;
    WNDCLASS              wndclass ;
```



```

    wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc          = WndProc ;
    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance            = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName          = szAppName ;
    wndclass.lpszClassName        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Blow-Up Mouse Demo"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    hAccel = LoadAccelerators (hInstance, szAppName) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

void InvertBlock (HWND hwndScr, HWND hwnd, POINT ptBeg, POINT ptEnd)
{
    HDC hdc ;
    hdc = GetDCEx (hwndScr, NULL, DCX_CACHE | DCX_LOCKWINDOWUPDATE) ;
    ClientToScreen (hwnd, &ptBeg) ;
    ClientToScreen (hwnd, &ptEnd) ;
    PatBlt (hdc, ptBeg.x, ptBeg.y, ptEnd.x - ptBeg.x, ptEnd.y - ptBeg.y,

```

```

                                DSTINVERT) ;

    ReleaseDC (hwndScr, hdc) ;
}

HBITMAP CopyBitmap (HBITMAP hBitmapSrc)
{
    BITMAP          bitmap ;
    HBITMAP          hBitmapDst ;
    HDC              hdcSrc, hdcDst ;

    GetObject (hBitmapSrc, sizeof (BITMAP), &bitmap) ;
    hBitmapDst = CreateBitmapIndirect (&bitmap) ;

    hdcSrc = CreateCompatibleDC (NULL) ;
    hdcDst = CreateCompatibleDC (NULL) ;

    SelectObject (hdcSrc, hBitmapSrc) ;
    SelectObject (hdcDst, hBitmapDst) ;

    BitBlt (hdcDst, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
            hdcSrc, 0, 0, SRCCOPY) ;

    DeleteDC (hdcSrc) ;
    DeleteDC (hdcDst) ;

    return hBitmapDst ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static BOOL          bCapturing, bBlocking ;
    static HBITMAP        hBitmap ;
    static HWND          hwndScr ;
    static POINT          ptBeg, ptEnd ;
    BITMAP                bm ;
    HBITMAP                hBitmapClip ;
    HDC                    hdc, hdcMem ;
    int                    iEnable ;
    PAINTSTRUCT            ps ;
    RECT                  rect ;

    switch (message)
    {
    case WM_LBUTTONDOWN:
        if (!bCapturing)
        {
            if (LockWindowUpdate (hwndScr) =
GetDesktopWindow ()))

```

```

        {
            bCapturing = TRUE ;
            SetCapture (hwnd) ;
            SetCursor      (LoadCursor      (NULL,
IDC_CROSS)) ;

        }
        else
            MessageBeep (0) ;
    }
    return 0 ;

case WM_RBUTTONDOWN:
    if (bCapturing)
    {
        bBlocking = TRUE ;
        ptBeg.x = LOWORD (lParam) ;
        ptBeg.y = HIWORD (lParam) ;
        ptEnd = ptBeg ;
        InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
    }
    return 0 ;

case WM_MOUSEMOVE:
    if (bBlocking)
    {
        InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
        ptEnd.x = LOWORD (lParam) ;
        ptEnd.y = HIWORD (lParam) ;
        InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
    }
    return 0 ;

case WM_LBUTTONUP:
case WM_RBUTTONUP:
    if (bBlocking)
    {
        InvertBlock (hwndScr, hwnd, ptBeg,
ptEnd) ;

        ptEnd.x = LOWORD (lParam) ;
        ptEnd.y = HIWORD (lParam) ;

        if (hBitmap)
        {
            DeleteObject (hBitmap) ;
            hBitmap = NULL ;
        }

        hdc = GetDC (hwnd) ;

```

```

                                hdcMem      =      CreateCompatibleDC
(hdc) ;

                                hBitmap      =      CreateCompatibleBitmap
(hdc,

                                abs (ptEnd.x - ptBeg.x),
                                abs (ptEnd.y - ptBeg.y)) ;

                                SelectObject (hdcMem, hBitmap) ;

                                StretchBlt (hdcMem, 0, 0,          abs (ptEnd.x -
ptBeg.x),
                                abs (ptEnd.y - ptBeg.y),
                                hdc, ptBeg.x,      ptBeg.y, ptEnd.x - ptBeg.x,
                                ptEnd.y - ptBeg.y, SRCCOPY) ;
                                DeleteDC (hdcMem) ;
                                ReleaseDC (hwnd, hdc) ;
                                InvalidateRect (hwnd, NULL, TRUE) ;
                                }
                                if (bBlocking || bCapturing)
                                {
                                    bBlocking = bCapturing = FALSE ;
                                    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
                                    ReleaseCapture () ;
                                    LockWindowUpdate (NULL) ;
                                }
                                return 0 ;

                                case WM_INITMENUPOPUP:
                                    iEnable = IsClipboardFormatAvailable (CF_BITMAP) ?
MF_ENABLED : MF_GRAYED ;

                                    EnableMenuItem ((HMENU) wParam, IDM_EDIT_PASTE,
iEnable) ;

                                    iEnable = hBitmap ? MF_ENABLED : MF_GRAYED ;

                                    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT,
iEnable) ;
                                    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY,
iEnable) ;
                                    EnableMenuItem ((HMENU) wParam, IDM_EDIT_DELETE,
iEnable) ;

                                    return 0 ;

                                case WM_COMMAND:
                                    switch (LOWORD (wParam))
                                    {
                                        case IDM_EDIT_CUT:

```

```
        case IDM_EDIT_COPY:
            if (hBitmap)
            {
                hBitmapClip = CopyBitmap (hBitmap) ;
                OpenClipboard (hwnd) ;
                EmptyClipboard () ;
                SetClipboardData (CF_BITMAP, hBitmapClip) ;
            }
            if (LOWORD (wParam) == IDM_EDIT_COPY)
                return 0 ;

            //fall through for IDM_EDIT_CUT
        case IDM_EDIT_DELETE:
            if (hBitmap)
            {
                DeleteObject (hBitmap) ;
                hBitmap = NULL ;
            }

            InvalidateRect (hwnd, NULL,
TRUE) ;

            return 0 ;

        case IDM_EDIT_PASTE:
            if (hBitmap)
            {
                DeleteObject (hBitmap) ;
                hBitmap = NULL ;
            }
            OpenClipboard (hwnd) ;
            hBitmapClip = GetClipboardData (CF_BITMAP) ;

            if (hBitmapClip)
                hBitmap = CopyBitmap (hBitmapClip) ;

            CloseClipboard () ;
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;
        }
        break ;
    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        if (hBitmap)
        {
            GetClientRect (hwnd, &rect) ;

            hdcMem = CreateCompatibleDC (hdc) ;
            SelectObject (hdcMem, hBitmap) ;
            GetObject (hBitmap, sizeof (BITMAP), (PSTR)
```

```

&bm) ;

                                SetStretchBltMode (hdc, COLORONCOLOR) ;

                                StretchBlt (hdc,          0,  0,  rect.right,
rect.bottom,
                                hdcMem, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY) ;

                                DeleteDC (hdcMem) ;
                                }
                                EndPaint (hwnd, &ps) ;
                                return 0 ;

case WM_DESTROY:
    if (hBitmap)
        DeleteObject (hBitmap) ;

        PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

BLOWUP.RC (摘录)

//Microsoft Developer Studio generated resource script.

```
#include "resource.h"
```

```
#include "afxres.h"
```

```

////////////////////////////////////
/

```

// Menu

BLOWUP MENU DISCARDABLE

BEGIN

POPUP "&Edit"

BEGIN

MENUITEM "Cu&t\tCtrl+X", IDME_EDIT_CUT

MENUITEM "&Copy\tCtrl+C", IDME_EDIT_COPY

MENUITEM "&Paste\tCtrl+V", IDME_EDIT_PASTE

MENUITEM "De&lete\tDelete", IDME_EDIT_DELETE

END

END

```

////////////////////////////////////
/

```

// Accelerator

BLOWUP ACCELERATORS DISCARDABLE

BEGIN

"C", IDME_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT

"V", IDME_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT

VK_DELETE, IDME_EDIT_DELETE, VIRTKEY, NOINVERT

```
"X",      IDM_EDIT_CUT,      VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
// Used by Blowup.rc
```

```
#define IDM_EDIT_CUT      40001
#define IDM_EDIT_COPY     40002
#define IDM_EDIT_PASTE    40003
#define IDM_EDIT_DELETE   40004
```

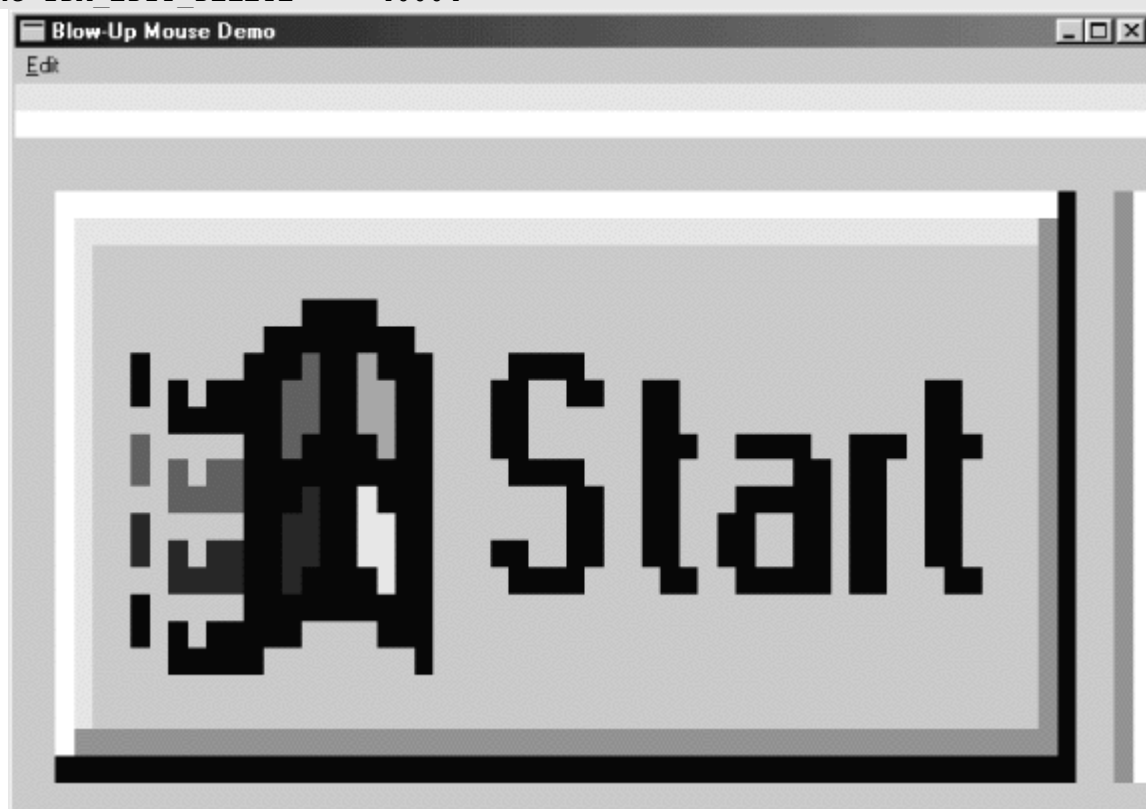


图 14-10 BLOWUP 显示的一个范例

由於滑鼠拦截的限制，所以开始使用 BLOWUP 时会有些困难，需要逐渐适应。下面是使用本程式的方法：

在 BLOWUP 显示区域按下滑鼠左键不放，滑鼠指标会变成「+」字型。

继续按住左键，将滑鼠移到萤幕上的任何其他位置。滑鼠游标的位置就是您要圈选的矩形区域的左上角。

继续按住左键，按下滑鼠右键，然後拖动滑鼠到您要圈选的矩形区域的右下角。释放滑鼠左键和右键。（释放滑鼠左、右键次序无关紧要。）

滑鼠游标恢复成箭头状，这时您圈选的矩形区域已复制到了 BLOWUP 的显示区域，并作了适当的压缩或拉伸变化。

如果您从右上角到左下角选取的话，BLOWUP 将显示矩形区域的镜像。如果从左下到右上角选取，BLOWUP 将显示颠倒的图像。如果从右上角至左上角选取，程式将综合两种效果。

BLOWUP 还包含将点阵图复制到剪贴簿，以及将剪贴簿中的点阵图复制到程式的处理功能。BLOWUP 处理 WM_INITMENUPOPUP 讯息来启用或禁用「Edit」功能表中的不同选项，并通过 WM_COMMAND 讯息来处理这些功能表项。您应该对这些程式码的结构比较熟悉，因为它们与第十二章中的复制和粘贴文字项目的处理方式在本质上是一样的。

不过，对于点阵图，剪贴簿物件不是整体代号而是点阵图代号。当您使用 CF_BITMAP 时，GetClipboardData 函式传回一个 HBITMAP 物件，而且 SetClipboardData 函式接收一个 HBITMAP 物件。如果您想将点阵图传送给剪贴簿又想保留副本以供程式本身使用，那么您必须复制点阵图。同样，如果您从剪贴簿上粘贴了一幅点阵图，也应该做一个副本。BLOWUP 中的 CopyBitmap 函式是通过取得现存点阵图的 BITMAP 结构，并在 CreateBitmapIndirect 函式中用这个结构建立一个新点阵图来完成此项操作的。（变数名的尾码 Src 和 Dst 分别代表「来源」和「目的」。）两个点阵图都被选进记忆体装置内容，而且通过呼叫 BitBlt 来复制点阵图内容。（另一种复制位元的方法，可以先按点阵图大小配置一块记忆体，然后为来源点阵图呼叫 GetBitmapBits，为目的点阵图呼叫 SetBitmapBits。）

我发现 BLOWUP 对于检查 Windows 及其應用程式中大量分散的小点阵图和图片非常有用。

第十五章 与装置无关的点阵图

在上一章我们了解到 Windows GDI 点阵图物件（也称为与装置相关的点阵图，或 DDB）有许多程式设计用途。但是我并没有展示把这些点阵图储存到磁片档案或把它们载入记忆体的方法。这是以前在 Windows 中使用的方法，现在根本不用了。因为点阵图的位元格式相当依赖於设备，所以 DDB 不适用于图像交换。DDB 内没有色彩对照表来指定点阵图的位与色彩之间的联系。DDB 只有在 Windows 开机到关机的生命期内被建立和清除时才有意义。

在 Windows 3.0 中发表了与装置无关的点阵图 (DIB)，提供了适用于交换的图像档案格式。正如您所知的，像 .GIF 或 .JPEG 之类的其他图像档案格式在 Internet 上比 DIB 档案更常见。这主要是因为 .GIF 和 .JPEG 格式进行了压缩，明显地减少了下载的时间。尽管有一个用于 DIB 的压缩方案，但极少使用。DIB 内的点阵图几乎都没有被压缩。如果您想在程式中操作点阵图，这实际上是一个优点。DIB 不像 .GIF 和 .JPEG 档案，Windows API 直接支援 DIB。如果在记忆体中有 DIB，您就可以提供指向该 DIB 的指标作为某些函式的参数，来显示 DIB 或把 DIB 转化为 DDB。

DIB 档案格式

有意思的是，DIB 格式并不是源自於 Windows。它首先定义在 OS/2 的 1.1 版中，该作业系统最初由 IBM 和 Microsoft 在八十年代中期开始开发。OS/2 1.1 在 1988 年发布，并且是第一个包含了类似 Windows 的图形使用者界面的 OS/2 版本，该图形使用者界面被称之为「Presentation Manager (PM)」。

「Presentation Manager」包含了定义点阵图格式的「图形程式介面」(GPI)。

然后在 Windows 3.0 中（发布于 1990）使用了 OS/2 点阵图格式，这时称之为 DIB。Windows 3.0 也包含了原始 DIB 格式的变体，并在 Windows 下成为标准。在 Windows 95（以及 Windows NT 4.0）和 Windows 98（以及 Windows NT 5.0）下也定义了一些其他的增强能力，我会在本章讨论它们。

DIB 首先作为一种档案格式，它的副档名为 .BMP，在极少情况下为 .DIB。Windows 應用程式使用的点阵图图像被当做 DIB 档案建立，并作为唯读资源储存在程式的可执行档案中。图示和滑鼠游标也是形式稍有不同的 DIB 档案。

程式能将 DIB 档案减去前 14 个位元组载入连续的记忆体块中。这时就可以称它为「packed DIB (packed-DIB) 格式的点阵图」。在 Windows 下执行的應用程式能使用 packed DIB 格式，通过 Windows 剪贴簿来交换图像或建立画刷。

程式也可以完全存取 DIB 的内容并以任意方式修改 DIB。

程式也能在记忆体中建立自己的 DIB 然後把它们存入档案。程式使用 GDI 函式呼叫就能「绘制」这些 DIB 内的图像，也能在程序中利用别的记忆体 DIB 直接设定和操作图素位元。

在记忆体中载入了 DIB 後，程式也能通过几个 Windows API 函式呼叫来使用 DIB 资料，我将在本章中讨论有关内容。与 DIB 相关的 API 呼叫是很少的，并且主要与视讯显示器或印表机页面上显示 DIB 相关，还与转换 GDI 点阵图物件有关。

除了这些内容以外，还有许多应用程式需要完成的 DIB 任务，而这些任务 Windows 作业系统并不支援。例如，程式可能存取了 24 位元 DIB 并且想把它转化为带有最佳化的 256 色调色盘的 8 位元 DIB，而 Windows 不会为您执行这些操作。但是在本章和下一章将向您显示 Windows API 之外的操作 DIB 的方式。

OS/2 样式的 DIB

先不要陷入太多的细节，让我们看一下与首先在 OS/2 1.1 中出现的点阵图格式相容的 Windows DIB 格式。

DIB 档案有四个主要部分：

- 档案表头
- 资讯表头
- RGB 色彩对照表（不一定有）
- 点阵图图素位元

您可以把前两部分看成是 C 的资料结构，把第三部分看成是资料结构的阵列。在 Windows 表头档案 WINGDI.H 中说明了这些结构。在记忆体中的 packed DIB 格式内有三个部分：

- 资讯表头
- RGB 色彩对照表（不一定有）
- 点阵图图素位元

除了没有档案表头外，其他部分与储存在档案内的 DIB 相同。

DIB 档案（不是记忆体中的 packed DIB）以定义为如下结构的 14 个位元组的档案表头开始：

```
typedef struct tagBITMAPFILEHEADER // bmfh
{
    WORD        bfType ;           // signature word "BM" or 0x4D42
    DWORD       bfSize ;           // entire size of file
    WORD        bfReserved1 ;      // must be zero
    WORD        bfReserved2 ;      // must be zero
```

```

        DWORD          bfOffsetBits ; // offset in file of DIB pixel bits
    }
    BITMAPFILEHEADER, * PBITMAPFILEHEADER ;

```

在 WINGDI.H 内定义的结构可能与这不完全相同，但在功能上是相同的。第一个注释（就是文字「bmfh」）指出了给这种资料型态的资料变数命名时推荐的缩写。如果在我的程式内看到了名为 pbmfh 的变数，这可能是一个指向 BITMAPFILEHEADER 型态结构的指标或指向 PBITMAPFILEHEADER 型态变数的指标。

结构的长度为 14 位元组，它以两个字母「BM」开头以指明是点阵图档案。这是一个 WORD 值 0x4D42。紧跟在「BM」後的 DWORD 以位元组为单位指出了包括档案表头在内的档案大小。下两个 WORD 栏位设定为 0。（在与 DIB 档案格式相似的滑鼠游标档案内，这两个栏位指出游标的「热点（hot spot）」）。结构还包含一个 DWORD 栏位，它指出了档案中图素位元开始位置的位元组偏移量。此数值来自 DIB 资讯表头中的资讯，为了使用的方便提供在这里。

在 OS/2 样式的 DIB 内，BITMAPFILEHEADER 结构後紧跟了 BITMAPCOREHEADER 结构，它提供了关于 DIB 图像的基本资讯。紧缩的 DIB（Packed DIB）开始於 BITMAPCOREHEADER：

```

typedef struct tagBITMAPCOREHEADER // bmch
{
    DWORD          bcSize ; // size of the structure = 12
    WORD           bcWidth ; // width of image in pixels
    WORD           bcHeight ; // height of image in pixels
    WORD           bcPlanes ; // = 1
    WORD           bcBitCount ; // bits per pixel (1, 4, 8, or 24)
}
    BITMAPCOREHEADER, * PBITMAPCOREHEADER ;

```

「core（核心）」用在这里看起来有点奇特，它是指这种格式是其他由它所衍生的点阵图格式的基础。

BITMAPCOREHEADER 结构中的 bcSize 栏位指出了资料结构的大小，在这种情况下是 12 位元组。

bcWidth 和 bcHeight 栏位包含了以图素为单位的点阵图大小。尽管这些栏位使用 WORD 意味著一个 DIB 可能为 65,535 图素高和宽，但是我们几乎不会用到那么大的单位。

bcPlanes 栏位的值始终是 1。这个栏位是我们在上一章中遇到的早期 Windows GDI 点阵图物件的残留物。

bcBitCount 栏位指出了每图素的位元数。对于 OS/2 样式的 DIB，这可能是 1、4、8 或 24。DIB 图像中的颜色数等於 2^{bmch.bcBitCount}，或用 C 的语法表示为：

```
1 << bmch.bcBitCount
```

这样，bcBitCount 栏位等於：

- 1 代表 2 色 DIB
- 4 代表 16 色 DIB
- 8 代表 256 色 DIB
- 24 代表 full -Color DIB

当我提到「8 位元 DIB」时，就是说每图素占 8 位元的 DIB。

对于前三种情况（也就是位元数为 1、4 和 8 时），BITMAPCOREHEADER 後紧跟色彩对照表，24 位元 DIB 没有色彩对照表。色彩对照表是一个 3 位元组 RGBTRIPLE 结构的阵列，阵列中的每个元素代表图像中的每种颜色：

```
typedef struct tagRGBTRIPLE // rgbt
{
    BYTE rgbtBlue ;           // blue level
    BYTE rgbtGreen ;          // green level
    BYTE rgbtRed ;            // red level
}
RGBTRIPLE ;
```

这样排列色彩对照表以便 DIB 中最重要的颜色首先显示，我们将在下一章说明原因。

WINGDI.H 表头档案也定义了下面的结构：

```
typedef struct tagBITMAPCOREINFO // bmci
{
    BITMAPCOREHEADER bmciHeader ;           // core-header structure
    RGBTRIPLE          bmciColors[1] ;       // color table array
}
BITMAPCOREINFO, * PBITMAPCOREINFO ;
```

这个结构把资讯表头与色彩对照表结合起来。虽然在这个结构中 RGBTRIPLE 结构的数量等於 1，但在 DIB 档案内您绝对不会发现只有一个 RGBTRIPLE。根据每个图素的位元数，色彩对照表的大小始终是 2、16 或 256 个 RGBTRIPLE 结构。如果需要为 8 位元 DIB 配置 PBITMAPCOREINFO 结构，您可以这样做：

```
pbmci = malloc (sizeof (BITMAPCOREINFO) + 255 * sizeof (RGBTRIPLE)) ;
```

然後可以这样存取 RGBTRIPLE 结构：

```
pbmci->bmciColors[i]
```

因为 RGBTRIPLE 结构的长度是 3 位元组，许多 RGBTRIPLE 结构可能在 DIB 中以奇数位址开始。然而，因为在 DIB 档案内始终有偶数个的 RGBTRIPLE 结构，所以紧跟在色彩对照表阵列後的资料块总是以 WORD 位址边界开始。

紧跟在色彩对照表（24 位元 DIB 中是资讯表头）後的资料是图素位元本身。

由下而上

像大多数点阵图格式一样，DIB 中的图素位元是以水平行组织的，用视讯显示器硬体的术语称作「扫描线」。行数等於 BITMAPCOREHEADER 结构的 bcHeight 栏位。然而，与大多数点阵图格式不同的是，DIB 从图像的底行开始，往上表示图像。

在此应定义一些术语，当我们说「顶行」和「底行」时，指的是当其正确显示在显示器或印表机的页面上时出现在虚拟图像的顶部和底部。就好像肖像的顶行是头发，底行是下巴，在 DIB 档案中的「第一行」指的是 DIB 档案的色彩对照表後的图素行，「最後行」指的是档案最末端的图素行。

因此，在 DIB 中，图像的底行是档案的第一行，图像的顶行是档案的最後一行。这称之为由下而上的组织。因为这种组织和直觉相反，您可能会问：为什么要这么做？

好，现在我们回到 OS/2 的 Presentation Manager。IBM 的人认为 PM 内的座标系统——包括视窗、图形和点阵图——应该是一致的。这引起了争论：大多数人，包括在全画面文字方式下编程和视窗环境下工作的程式写作者认为应使用垂直座标在萤幕上向下增加的座标。然而，电脑图形程式写作者认为应使用解析几何的数学方法进行视讯显示，这是一个垂直座标在空间中向上增加的直角（或笛卡尔）座标系。

简而言之，数学方法赢了。PM 内的所有事物都以左下角为原点（包括视窗座标），因此 DIB 也就有了那种方式。

DIB 图素位元

DIB 档案的最後部分（在大多数情况下是 DIB 档案的主体）由实际的 DIB 的图素位元组成。图素位元是由从图像的底行开始并沿著图像向上增长的水平行组织的。

DIB 中的行数等於 BITMAPCOREHEADER 结构的 bcHeight 栏位。每一行的图素数等於该结构的 bcWidth 栏位。每一行从最左边的图素开始，直到图像的右边。每个图素的位元数可以从 bcBitCount 栏位取得，为 1、4、8 或 24。

以位元组为单位的每行长度始终是 4 的倍数。行的长度可以计算为：

```
RowLength = 4 * ((bmch.bcWidth * bmch.bcBitCount + 31) / 32) ;
```

或者在 C 内用更有效的方法：

```
RowLength = ((bmch.bcWidth * bmch.bcBitCount + 31) & ~31) >> 3 ;
```

如果需要，可通过在右边补充行（通常是用零）来完成长度。图素资料的总位元组数等於 RowLength 和 bmch.bcHeight 的乘积。

要了解图素编码的方式，让我们分别考虑四种情况。在下面的图表中，每个位元组的位元显示在框内并且编了号，7 表示最高位元，0 表示最低位元。图素也从行的最左端从 0 开始编号。

对于每图素 1 位元的 DIB，每位元组对应为 8 图素。最左边的图素是第一个位元组的最高位元：

Pixel:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

每个图素可以是 0 或 1。0 表示该图素的颜色由色彩对照表中第一个 RGBTRIPLE 项目给出。1 表示图素的颜色由色彩对照表的第二个项目给出。

对于每图素 4 位元的 DIB，每个位元组对应两个图素。最左边的图素是第一个位元组的高 4 位元，以此类推：

Pixel:	— 0 —								— 1 —								— 2 —								— 3 —								— 4 —								— 5 —							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

每图素 4 位元的值的范围从 0 到 15。此值是指向色彩对照表中 16 个项目的索引。

对于每图素 8 位元的 DIB，每个位元组为 1 个图素：

Pixel:	— 0 —								— 1 —								— 2 —							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

位元组的值从 0 到 255。同样，这也是指向色彩对照表中 256 个项目的索引。

对于每图素 24 位元的 DIB，每个图素需要 3 个位元组来代表红、绿和蓝的颜色值。图素位元的每一行，基本上就是 RGBTRIPLE 结构的阵列，可能需要在每行的末端补 0 以便该行为 4 位元组的倍数：

Pixel:	— Blue —								— Green —								— Red —							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

每图素 24 位元的 DIB 没有色彩对照表。

扩展的 Windows DIB

现在我们掌握了 Windows 3.0 中介绍的与 OS/2 相容的 DIB，同时也看一看 Windows 中 DIB 的扩展版本。

这种 DIB 形式跟前面的格式一样，以 BITMAPFILEHEADER 结构开始，但是接著是 BITMAPINFOHEADER 结构，而不是 BITMAPCOREHEADER 结构：

```
typedef struct tagBITMAPINFOHEADER // bmih
{
    DWORD      biSize ;           // size of the structure = 40
    LONG  biWidth ;               // width of the image in pixels
    LONG  biHeight ;             // height of the image in pixels
```

```

    WORD  biPlanes ;                // = 1
    WORD  biBitCount ;              // bits per pixel (1, 4, 8, 16, 24, or 32)
    DWORD  biCompression ;          // compression code
    DWORD  biSizeImage ;            // number of bytes in image
    LONG   biXPelsPerMeter ;         // horizontal resolution
    LONG   biYPelsPerMeter ;         // vertical resolution
    DWORD  biClrUsed ;               // number of colors used
    DWORD  biClrImportant ;          // number of important colors
}
BITMAPINFOHEADER, * PBITMAPINFOHEADER ;

```

您可以通过检查结构的第一栏位区分与 OS/2 相容的 DIB 和 Windows DIB, 前者为 12, 后者为 40。

您将注意到, 在这个结构内有六个附加的栏位, 但是 BITMAPINFOHEADER 不是简单地由 BITMAPCOREHEADER 加上一些新栏位而成。仔细看一下: 在 BITMAPCOREHEADER 结构中, bcWidth 和 bcHeight 栏位是 16 位元 WORD 值; 而在 BITMAPINFOHEADER 结构中它们是 32 位元 LONG 值。这是一个令人讨厌的小变化, 当心它会给您带来麻烦。

另一个变化是: 对于使用 BITMAPINFOHEADER 结构的 1 位元、4 位元和 8 位元 DIB, 色彩对照表不是 RGBTRIPLE 结构的阵列。相反, BITMAPINFOHEADER 结构紧跟著一个 RGBQUAD 结构的阵列:

```

typedef struct tagRGBQUAD // rgb
{
    BYTE rgbBlue ;    // blue level
    BYTE rgbGreen ;   // green level
    BYTE rgbRed ;     // red level
    BYTE rgbReserved ; // = 0
}
RGBQUAD ;

```

除了包括总是设定为 0 的第四个栏位外, 与 RGBTRIPLE 结构相同。WINGDI.H 表头档案也定义了以下结构:

```

typedef struct tagBITMAPINFO // bmi
{
    BITMAPINFOHEADER bmiHeader ;    // info-header structure
    RGBQUAD           bmiColors[1] ; // color table array
}
BITMAPINFO, * PBITMAPINFO ;

```

注意, 如果 BITMAPINFO 结构以 32 位元的位址边界开始, 因为 BITMAPINFOHEADER 结构的长度是 40 位元组, 所以 RGBQUAD 阵列内的每一个项目也以 32 位边界开始。这样就确保通过 32 位元微处理器能更有效地对色彩对照表资料定址。

尽管 BITMAPINFOHEADER 最初是在 Windows 3.0 中定义的, 但是许多栏位在

Windows 95 和 Windows NT 4.0 中又重新定义了, 并且被带入 Windows 98 和 Windows NT 5.0 中。比如现在的文件中说: 「如果 biHeight 是负数, 则点阵图是由上而下的 DIB, 原点在左上角」。这很好, 但是在 1990 年刚开始定义 DIB 格式时, 如果有人做了这个决定, 那会更好。我的建议是避免建立由上而下的 DIB。有一些程式在编写时没有考虑这种新「特性」, 在遇到负的 biHeight 栏位时会当掉。还有如 Microsoft Word 97 带有的 Microsoft Photo Editor 在遇到由上而下的 DIB 时会报告「图像高度不合法」(虽然 Word 97 本身不会出错)。

biPlanes 栏位始终是 1, 但 biBitCount 栏位现在可以是 16 或 32 以及 1、4、8 或 24。这也是在 Windows 95 和 Windows NT 4.0 中的新特性。一会儿我将介绍这些附加格式工作的方式。

现在让我们先跳过 biCompression 和 biSizeImage 栏位, 一会儿再讨论它们。

biXPelsPerMeter 和 biYPelsPerMeter 栏位以每公尺多少图素这种笨拙的单位指出图像的实际尺寸。(「pel」--picture element (图像元素)--是 IBM 对图素的称呼。) Windows 在内部不使用此类资讯。然而, 应用程式能够利用它以准确的大小显示 DIB。如果 DIB 来源於没有方图素的设备, 这些栏位是很有用的。在大多数 DIB 内, 这些栏位设定为 0, 这表示没有建议的实际大小。每英寸 72 点的解析度(有时用於视讯显示器, 尽管实际解析度依赖於显示器的大小)大约相当於每公尺 2835 个图素, 300 DPI 的普通印表机的解析度是每公尺 11,811 个图素。

biClrUsed 是非常重要的栏位, 因为它影响色彩对照表中项目的数量。對於 4 位元和 8 位元 DIB, 它能分别指出色彩对照表中包含了小於 16 或 256 个项目。虽然并不常用, 但这是一种缩小 DIB 大小的方法。例如, 假设 DIB 图像仅包括 64 个灰阶, biClrUsed 栏位设定为 64, 并且色彩对照表为 256 个位元组大小的色彩对照表包含了 64 个 RGBQUAD 结构。图素值的范围从 0x00 到 0x3F。DIB 仍然每图素需要 1 位元组, 但每个图素位元组的高 2 位元为零。如果 biClrUsed 栏位设定为 0, 意味著色彩对照表包含了由 biBitCount 栏位表示的全部项目数。

从 Windows 95 开始, biClrUsed 栏位對於 16 位元、24 位元或 32 位元 DIB 可以为非零。在这些情况下, Windows 不使用色彩对照表解释图素位元。相反地, 它指出 DIB 中色彩对照表的大小, 程式使用该资讯来设定调色盘在 256 色视讯显示器上显示 DIB。您可能想起在 OS/2 相容格式中, 24 位元 DIB 没有色彩对照表。在 Windows 3.0 中的扩展格式中, 也与这一样。而在 Windows 95 中, 24 位元 DIB 有色彩对照表, biClrUsed 栏位指出了它的大小。

总结如下:

對於 1 位元 DIB, biClrUsed 始终是 0 或 2。色彩对照表始终有两个项目。

對於 4 位元 DIB, 如果 biClrUsed 栏位是 0 或 16, 则色彩对照表有 16 个项目。如果是从 2 到 15 的数, 则指的是色彩对照表中的项目数。每个图素的最大值是小於该数的 1。

對於 8 位元 DIB, 如果 biClrUsed 栏位是 0 或 256, 则色彩对照表有 256 个项目。如果是从 2 到 225 的数, 则指的是色彩对照表中的项目数。每个图素的最大值是小於该数的 1。

對於 16 位元、24 位元或 32 位元 DIB, biClrUsed 栏位通常为 0。如果它不为 0, 则指的是色彩对照表中的项目数。执行於 256 色显示卡的应用程式能使用这些项目来为 DIB 设定调色盘。

另一个警告: 原先使用早期 DIB 文件编写的程式不支援 24 位元 DIB 中的色彩对照表, 如果在程式使用 24 位元 DIB 的色彩对照表的话, 就要冒一定的风险。

biClrImportant 栏位实际上没有 biClrUsed 栏位重要, 它通常被设定为 0 以指出色彩对照表中所有的颜色都是重要的, 或者它与 biClrUsed 有相同的值。两种方法意味著同一件事, 如果它被设定为 0 与 biClrUsed 之间的值, 就意味著 DIB 图像能仅仅通过色彩对照表中第一个 biClrImportant 项目合理地取得。当在 256 色显示卡上并排显示两个或更多 8 位元 DIB 时, 这是很有用的。

對於 1 位元、4 位元、8 位元和 24 位元的 DIB, 图素位元的组织和 OS/2 相容的 DIB 是相同的, 一会儿我将讨论 16 位元和 32 位元 DIB。

真实检查

当遇到一个由其他程式或别人建立的 DIB 时, 您希望从中发现什么内容呢?

尽管在 Windows3.0 首次推出时, OS/2 样式的 DIB 已经很普遍了, 但最近这种格式却已经很少出现了。许多程式写作者在实际编写快速 DIB 常式时忽略了它们。您遇到的任何 4 位元 DIB 可能是 Windows 的「小画家」程式使用 16 色视讯显示器建立的, 在这些显示器上色彩对照表具有标准的 16 种颜色。

最普遍的 DIB 可能是每图素 8 位元。8 位元 DIB 分为两类: 灰阶 DIB 和混色 DIB。不幸的是, 表头资讯中并没有指出 8 位元 DIB 的型态。

许多灰阶 DIB 有一个等於 64 的 biClrUsed 栏位, 指出色彩对照表中的 64 个项目。这些项目通常以上升的灰阶层排列, 也就是说色彩对照表以 00-00-00、04-04-04、08-08-08、0C-0C-0C 的 RGB 值开始, 并包括 F0-F0-F0、F4-F4-F4、F8-F8-F8 和 FC-FC-FC 的 RGB 值。此类色彩对照表可用下列公式计算:

```
rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 256 / 64 ;
```

在这里 rgb 是 RGBQUAD 结构的阵列, i 的范围从 0 到 63。灰阶色彩对照表

可用下列公式计算：

```
rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 255 / 63 ;
```

因此此表以 FF-FF-FF 结尾。

实际上使用哪个计算公式并没有什么区别。许多视讯显示卡和显示器没有比 6 位元更大的色彩精确度。第一个公式承认了这个事实。然而当产生小於 64 的灰阶时——可能是 16 或 32（在此情况下公式的除数分别是 15 和 31）——使用第二个公式更适合，因为它确保了色彩对照表的最後一个项目是 FF-FF-FF，也就是白色。

当某些 8 位元灰阶 DIB 在色彩对照表内有 64 个项目时，其他灰阶的 DIB 会有 256 个项目。biClrUsed 栏位实际上可以为 0（指出色彩对照表中有 256 个项目）或者从 2 到 256 的数。当然，biClrUsed 值是 2 的话就没有任何意义（因为这样的 8 位元 DIB 能当作 1 位元 DIB 被重新编码）或者小於或等於 16 的值也没意义（因为它能当作 4 位元 DIB 被重新编码）。任何情况下，色彩对照表中的项目数必须与 biClrUsed 栏位相同（如果 biClrUsed 是 0，则是 256），并且图素值不能超过色彩对照表项目数减 1 的值。这是因为图素值是指向色彩对照表阵列的索引。对于 biClrUsed 值为 64 的 8 位元 DIB，图素值的范围从 0x00 到 0x3F。

在这里应记住一件重要的事情：当 8 位元 DIB 具有由整个灰阶组成的色彩对照表（也就是说，当红色、绿色和蓝色程度相等时），或当这些灰阶层在色彩对照表中递增（像上面描述的那样）时，图素值自身就代表了灰色的程度。也就是说，如果 biClrUsed 是 64，那么 0x00 图素值为黑色，0x20 的图素值是 50% 的灰阶，0x3F 的图素值为白色。

这对于一些图像处理作业是很重要的，因为您可以完全忽略色彩对照表，仅需处理图素值。这是很有用的，如果让我回溯时光去对 BITMAPINFOHEADER 结构做一个简单的更改，我会添加一个旗标指出 DIB 映射是不是灰阶的，如果是，DIB 就没有色彩对照表，并且图素值直接代表灰阶。

混色的 8 位元 DIB 一般使用整个色彩对照表，它的 biClrUsed 栏位为 0 或 256。然而您也可能遇到较小的颜色数，如 236。我们应承认一个事实：程式通常只能在 Windows 颜色面内更改 236 个项目以正确显示这些 DIB，我将在下章讨论这些内容。

biXPelsPerMeter 和 biYPelsPerMeter 很少为非零值，biClrImportant 栏位不为 0 或 biClrUsed 值的情况也很少。

DIB 压缩

前面我没有讨论 BITMAPINFOHEADER 中的 biCompression 和 biSizeImage 栏位，现在我们讨论一下这些值。

biCompression 栏位可以为四个常数之一，它们是：BI_RGB、BI_RLE8、BI_RLE4 或 BI_BITFIELDS。它们定义在 WINGDI.H 表头档案中，值分别为 0 到 3。此栏位有两个用途：对于 4 位元和 8 位元 DIB，它指出图素位元被用一种运行长度 (run-length) 编码方式压缩了。对于 16 位元和 32 位元 DIB，它指出了颜色遮罩 (color masking) 是否用于对图素位元进行编码。这两个特性都是在 Windows 95 中发表的。

首先让我们看一下 RLE 压缩：

- 对于 1 位元 DIB，biCompression 栏位始终是 BI_RGB。
- 对于 4 位元 DIB，biCompression 栏位可以是 BI_RGB 或 BI_RLE4。
- 对于 8 位元 DIB，biCompression 栏位可以是 BI_RGB 或 BI_RLE8。
- 对于 24 位元 DIB，biCompression 栏位始终是 BI_RGB。

如果值是 BI_RGB，图素位元储存的方式和 OS/2 相容的 DIB 一样，否则就使用运行长度编码压缩图素位元。

运行长度编码 (RLE) 是一种最简单的资料压缩形式，它是根据 DIB 映射在一列内经常有相同的图素字串这个事实进行的。RLE 通过对重复图素的值及重复的次数编码来节省空间，而用于 DIB 的 RLE 方案只定义了很少的矩形 DIB 图像，也就是说，矩形的某些区域是未定义的，这能被用于表示非矩形图像。

8 位元 DIB 的运行长度编码在概念上更简单一些，因此让我们从这里入手。表 15-1 会帮助您理解当 biCompression 栏位等于 BI_RGB8 时，图素位元的编码方式。

表 15-1

位元组 1	位元组 2	位元组 3	位元组 4	含义
00	00			行尾
00	01			映射尾
00	02	dx	dy	移到 (x+dx, y+dy)
00	n = 03 到 FF			使用下面 n 个图素
n = 01 到 FF	图素			重复图素 n 次

当对压缩的 DIB 解码时，可成对查看 DIB 资料位元组，例如此表内的「位元组 1」和「位元组 2」。表格以这些位元组值的递增方式排列，但由下而上讨论这个表格会更有意义。

如果第一个位元组非零（表格最後一行的情况），那么它就是运行长度的重复因数。下面的图素值被重复多次，例如，位元组对

0x05 0x27

解码後的图素值为：

0x27 0x27 0x27 0x27 0x27

当然 DIB 会有许多资料不是图素到图素的重复，表格倒数第二行处理这种情况，它指出紧跟著的图素数应逐个使用。例如：考虑序列

0x00 0x06 0x45 0x32 0x77 0x34 0x59 0x90

解码後的图素值为：

0x45 0x32 0x77 0x34 0x59 0x90

这些序列总是以 2 位元组的界限排列。如果第二个位元组是奇数，那么序列内就有一个未使用的多余位元组。例如，序列

0x00 0x05 0x45 0x32 0x77 0x34 0x59 0x00

解码後的图素值为：

0x45 0x32 0x77 0x34 0x59

这就是运行长度编码的工作方式。很明显地，如果在 DIB 图像内没有重复的图素，使用此压缩技术实际上会增加 DIB 档案的大小。

上面表格的前三行指出了矩形 DIB 图像的某些部分可以不被定义的方法。想像一下，您写的程式对已压缩的 DIB 进行解压缩，在这个解压缩的常式中，您将保持一对数字 (x, y)，开始为 (0, 0)。每对一个图素解码，x 的值就增加 1，每完成一行就将 x 重新设为 0 并且增加 y 的值。

当遇到跟著 0x02 的位元组 0x00 时，您读取下两个位元组并把它们作为无正负号的增量添加到目前的 x 和 y 值中，然後继续解码。当遇到跟著 0x00 的 0x00 时，您就解完了一行，应将 x 设 0 并增加 y 值。当遇到跟著 0x01 的 0x00 时，您就完成解码了。这些代码准许 DIB 包含那些未定义的区域，它们用於对非矩形图像编码或在制作数位动画和电影时非常有用（因为几乎每一格影像都有来自前一格的资讯而不需重新编码）。

對於 4 位元 DIB，编码一般是相同的，但更复杂，因为位元组和图素之间不是一对一的关系。

如果读取的第一个位元组非零，那就是一个重复因数 n。第二个位元组（被重复的）包含 2 个图素，在 n 个图素的被解码的序列中交替出现。例如，位元组对

0x07 0x35

被解码为：

0x35 0x35 0x35 0x3?

其中的问号指出图素还未知，如果是上面显示的 0x07 0x35 对紧跟著下面的位元组对：

0x05 0x24

则整个解码的序列为：

0x35 0x35 0x35 0x32 0x42 0x42

如果位元组对中的第一位元组是 0x00，第二个位元组是 0x03 或更大，则使用第二位元组指出的图素数。例如，序列

0x00 0x05 0x23 0x57 0x10 0x00

解码为：

0x23 0x57 0x1?

注意必须填补解码的序列使其成为偶数位元组。

无论 biCompression 栏位是 BI_RLE4 或 BI_RLE8，biSizeImage 栏位都指出了位元组内 DIB 图素资料的大小。如果 biCompression 栏位是 BI_RGB，则 biSizeImage 通常为 0，但是它能被设定为行内位元组长度的 biHeight 倍，就像在本章前面计算的那样。

目前文件说「由上而下的 DIB 不能被压缩」。由上而下的 DIB 是在 biHeight 栏位为负数的情况下出现的。

颜色遮罩 (color masking)

biCompression 栏位也用於连结 Windows 95 中新出现的 16 位元和 32 位元 DIB。对于这些 DIB，biCompression 栏位可以是 BI_RGB 或 BI_BITFIELDS (均定义为值 3)。

让我们看一下 24 位元 DIB 的图素格式，它始终有一个等於 BI_RGB 的 biCompression 栏位：

也就是说，每一行基本上都是 RGBTRIPLE 结构的阵列，在每行末端有可能补充值以使行内的位元组是 4 的倍数。

Pixel: — Blue — — Green — — Red —

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

对于具有 biCompression 栏位等於 BI_RGB 的 16 位元 DIB，每个图素需要两个位元组。颜色是这样来编码的：

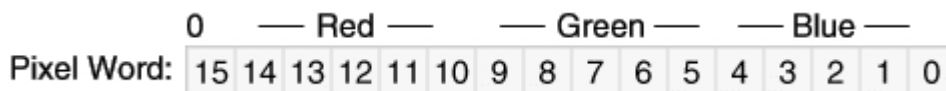
Pixel: ...een — — — Blue — — — 0 — — — Red — — — Gr...

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

每种颜色使用 5 位元。对于行内的第一个图素，蓝色值是第一个位元组的

最低五位元。绿色值在第一和第二个位元组中都有位元：绿色值的两个最高位元是第二个位元组中的两个最低位元，绿色值的三个最低位元是第一个位元组中的三个最高位元。红色值是第二个位元组中的 2 到 6 位元。第二个位元组的最高位元是 0。

当以 16 位元字组存取图素值时，这会更加有意义。因为多个位元组值的最低位元首先被储存，图素字组如下：



假设在 wPixel 内储存了 16 位元图素，您能用下列公式计算红色、绿色和蓝色值：

```

Red      = ((0x7C00 & wPixel) >> 10) << 3 ;
Green    = ((0x03E0 & wPixel) >> 5) << 3 ;
Blue     = ((0x001F & wPixel) >> 0) << 3 ;
  
```

首先，使用遮罩值与图素进行了位元 AND 运算。此结果是：红色向右移动 10 位元，绿色向右移动 5 位元，蓝色向右移动 0 位元。（这些移动值我称之为「右移值」）。这就产生了从 0x00 和 0x1F 的颜色值，这些值必须向左移动 3 位元以合成从 0x00 到 0xF8 的颜色值。（这些移动值我称之为「左移值」。）

请记住：如果 16 位元 DIB 的图素宽度是奇数，每行会在末端补充多余的 2 位元组以使位元组宽度能被 4 整除。

对于 32 位元 DIB，如果 biCompression 等于 BI_RGB，每个图素需要 4 位元组。蓝色值是第一个位元组，绿色为第二个，红色为第三个，第四位元组等于 0。也可这么说，图素是 RGBQUAD 结构的阵列。因为每个图素的长度是 4 位元组，在列末端就不需填补位元组。

若想以 32 位元双字组存取每个图素，它就像这样：



如果 dwPixel 是 32 位元双字组，

```

Red      = ((0x00FF0000 & dwPixel) >> 16) << 0 ;
Green    = ((0x0000FF00 & dwPixel) >> 8) << 0 ;
Blue     = ((0x000000FF & dwPixel) >> 0) << 0 ;
  
```

左移值全为零，因为颜色值在 0xFF 已是最大。注意这个双字组与 Windows GDI 函式呼叫中用于指定 RGB 颜色的 32 位元 COLORREF 值不一致。在 COLORREF 值中，红色占最低位元的位元组。

到目前为止，我们讨论了当 biCompression 栏位为 BI_RGB 时，16 位元和 32 位元 DIB 的内定情况。如果 biCompression 栏位为 BI_BITFIELDS，则紧跟著 DIB 的 BITMAPINFOHEADER 结构的是三个 32 位元颜色遮罩，第一个用于红色，第

二个用於绿色，第三个用於蓝色。可以使用 C 的位元 AND 运算符 (&) 把这些遮罩应用於 16 位元或 32 位元的图素值上。然後通过右移值向右移动结果，这些值只有检查完遮罩後才能知道。颜色遮罩的规则应该很明确：每个颜色遮罩位元串内的 1 必须是连续的，并且 1 不能在三个遮罩位元串中重叠。

让我们来举个例子，如果您有一个 16 位元 DIB，并且 biCompression 栏位为 BI_BITFIELDS。您应该检查 BITMAPINFOHEADER 结构之後的前三个双字组：

0x0000F800

0x000007E0

0x0000001F

注意，因为这是 16 位元 DIB，所以只有位於底部 16 位元的位元值才能被设定为 1。您可以把变数 dwMask[0]、dwMask[1]和 dwMask[2]设定为这些值。现在可以编写从遮罩中计算右移和左移值的一些常式了：

```
int MaskToRShift (DWORD dwMask)
{
    int iShift ;
    if ( dwMask == 0)
        return 0 ;

    for (      iShift = 0 ; !(dwMask & 1) ; iShift++)
        dwMask >>= 1 ;

    return iShift ;
}

int MaskToLShift (DWORD dwMask)
{
    int iShift ;
    if ( dwMask == 0)
        return 0 ;

    while (!(dwMask & 1))
        dwMask >>= 1 ;

    for (iShift = 0 ; dwMask & 1 ; iShift++)
        dwMask >>= 1 ;

    return 8 - iShift ;
}
```

然後呼叫 MaskToRShift 函式三次来获得右移值：

```
iRShift[0] = MaskToRShift (dwMask[0]) ;
iRShift[1] = MaskToRShift (dwMask[1]) ;
iRShift[2] = MaskToRShift (dwMask[2]) ;
```

分别得到值 11、5 和 0。然後呼叫 MaskToLShift:

```
iLShift[0] = MaskToLShift (dwMask[0]) ;
iLShift[1] = MaskToLShift (dwMask[1]) ;
iLShift[2] = MaskToLShift (dwMask[2]) ;
```

分别得到值 3、2 和 3。现在能从图素中提取每种颜色:

```
Red      = ((dwMask[0] & wPixel) >> iRShift[0]) << iLShift[0] ;
Green    = ((dwMask[1] & wPixel) >> iRShift[1]) << iLShift[1] ;
Blue     = ((dwMask[2] & wPixel) >> iRShift[2]) << iLShift[2] ;
```

除了颜色标记能大於 0x0000FFFF (这是 16 位元 DIB 的最大遮罩值) 之外, 程序与 32 位元 DIB 一样。

注意:

對於 16 位元或 32 位元 DIB, 红色、绿色和蓝色值能大於 255。实际上, 在 32 位元 DIB 中, 如果遮罩中有两个为 0, 第三个应为 32 位元颜色值 0xFFFFFFFF。当然, 这有点荒唐, 但不用担心这个问题。

不像 Windows NT, Windows 95 和 Windows 98 在使用颜色遮罩时有许多的限制。可用的值显示在表 15-2 中。

表 15-2

	16 位元 DIB	16 位元 DIB	32 位元 DIB
红色遮罩	0x00007C00	0x0000F800	0x00FF0000
绿色遮罩	0x000003E0	0x000007E0	0x0000FF00
蓝色遮罩	0x0000001F	0x0000001F	0x000000FF
速记为	5-5-5	5-6-5	8-8-8

换句话说, 就是当 biCompression 是 BI_RGB 时, 您能使用内定的两组遮罩, 包括前面例子中显示的遮罩组。表格底行显示了一个速记符号来指出每图素红色、绿色和蓝色的位元数。

第 4 版本的 Header

我说过, Windows 95 更改了一些原始 BITMAPINFOHEADER 栏位的定义。Windows 95 也包括了一个称为 BITMAPV4HEADER 的新扩展的资讯表头。如果您知道 Windows 95 曾经称作 Windows 4.0, 则就会明白此结构的名称了, Windows NT 4.0 也支援此结构。

```
typedef struct
{
    DWORD          bV4Size ;           // size of the structure = 120
```



```

LONG          bV4Width ;           // width of the image in pixels
LONG          bV4Height ;          // height of the image in pixels
WORD          bV4Planes ;          // = 1
WORD          bV4BitCount ;        // bits per pixel (1, 4, 8, 16, 24, or
32)
DWORD         bV4Compression ;     // compression code
DWORD         bV4SizeImage ;        // number of bytes in image
LONG          bV4XPelsPerMeter ;    // horizontal resolution
LONG          bV4YPelsPerMeter ;    // vertical resolution
DWORD         bV4ClrUsed ;          // number of colors used
DWORD         bV4ClrImportant ;     // number of important colors
DWORD         bV4RedMask ;          // Red color mask
DWORD         bV4GreenMask ;        // Green color mask
DWORD         bV4BlueMask ;         // Blue color mask
DWORD         bV4AlphaMask ;        // Alpha mask
DWORD         bV4CSType ;           // color space type
CIEXYZTRIPLE bV4Endpoints ;        // XYZ values
DWORD         bV4GammaRed ;         // Red gamma value
DWORD         bV4GammaGreen ;       // Green gamma value
DWORD         bV4GammaBlue ;        // Blue gamma value
}
BITMAPV4HEADER, * PBITMAPV4HEADER ;

```

注意前 11 个栏位与 BITMAPINFOHEADER 结构中的相同，後 5 个栏位支援 Windows 95 和 Windows NT 4.0 的图像颜色调配技术。除非使用 BITMAPV4HEADER 结构的後四个栏位，否则您应该使用 BITMAPINFOHEADER (或 BITMAPV5HEADER)。

当 bV4Compression 栏位等於 BI_BITFIELDS 时，bV4RedMask、bV4GreenMask 和 bV4BlueMask 可以用於 16 位元和 32 位元 DIB。它们作为定义在 BITMAPINFOHEADER 结构中的颜色遮罩用於相同的函式，并且当使用除了明确的结构栏位之外的原始结构时，它们实际上出现在 DIB 档案的相同位置。就我所知，bV4AlphaMask 栏位不被使用。

BITMAPV5HEADER 结构剩余的栏位包括「Windows 颜色管理 (Image Color Management)」，它的内容超越了本书的范围，但是了解一些背景会对您有益。

为色彩使用 RGB 方案的问题在於，它依赖於视讯显示器、彩色照相机和彩色扫描器的显示技术。如果颜色指定为 RGB 值 (255, 0, 0)，意味著最大的电压应该加到阴极射线管内的红色电子枪上，RGB 值 (128, 0, 0) 表示使用一半电压。不同显示器会产生不同的效果。而且，印表机使用了不同的颜色表示方法，以青色、洋红色、黄色和黑色的组合表示颜色。这些方法称之为 CMY (cyan-magenta-yellow：青色 - 洋红色 - 黄色) 和 CMYK (cyan-magenta-yellow-black：青色-洋红色-黄色-黑色)。数学公式能把 RGB 值转化为 CMY 和 CMYK，但不能保证印表机颜色与显示器颜色相符合。「色

彩调配技术」是把颜色与对装置无关的标准联系起来的一种尝试。

颜色的现象与可见光的波长有关，波长的范围从 380nm（蓝）到 780nm（红）之间。一切我们能察觉的光线是可见光谱内不同波长的组合。1931 年，Commission Internationale de L'Eclairage (International Commission on Illumination) 或 CIE 开发了一种科学度量颜色的方法。这包括使用三个颜色调配函数（名称为 x、y 和 z），它们以其省略的形式（带有每 5nm 的值）发表在 CIE Publication 15.2-1986, 「Colorimetry, Second Edition」的表 2.1 中。

颜色的光谱 (S) 是一组指出每个波长强度的值。如果知道光谱，就能够将与颜色相关的函数应用到光谱来计算 X、Y 和 Z：

$$X = \sum_{\lambda=380}^{780} S(\lambda) \bar{x}(\lambda)$$

$$Y = \sum_{\lambda=380}^{780} S(\lambda) \bar{y}(\lambda)$$

$$Z = \sum_{\lambda=380}^{780} S(\lambda) \bar{z}(\lambda)$$

这些值称为 **大 X、大 Y 和大 Z**。y 颜色匹配函数等於肉眼对范围在可见光谱内光线的反应。（他看上去像一条由 380nm 和 780nm 到 0 的时钟形曲线）。Y 称之为 CIE 亮度，因为它指出了光线的总体强度。

如果使用 BITMAPV5HEADER 结构，bV4CSType 栏位就必须设定为 LCS_CALIBRATED_RGB，其值为 0。後四个位元组必须设定为有效值。

CIEXYZTRIPLE 结构按照如下方式定义：

```
typedef struct tagCIEXYZTRIPLE
{
    CIEXYZ  cixyzRed ;
    CIEXYZ  cixyzGreen ;
    CIEXYZ  cixyzBlue ;
}
CIEXYZTRIPLE, * LPCIEXYZTRIPLE ;
```

而 CIEXYZ 结构定义如下：

```
typedef struct tagCIEXYZ
{
    FXPT2DOT30  cixyzX ;
    FXPT2DOT30  cixyzY ;
    FXPT2DOT30  cixyzZ ;
}
CIEXYZ, * LPCIEXYZ ;
```

这三个栏位定义为 FXPT2DOT30 值, 意味著它们是带有 2 位元整数部分和 30 位元小数部分的定点值。这样, 0x40000000 是 1.0, 0x48000000 是 1.5。最大值 0xFFFFFFFF 仅比 4.0 小一点点。

bV4Endpoints 栏位提供了三个与 RGB 颜色 (255, 0, 0)、(0, 255, 0) 和 (0, 0, 255) 相关的 X、Y 和 Z 值。这些值应该由建立 DIB 的应用程式插入以指明这些 RGB 颜色的装置无关的意义。

BITMAPV4HEADER 剩余的三个栏位指「伽马值」(希腊的小写字母 γ)，它指出颜色等级规格内的非线性。在 DIB 内, 红、绿、蓝的范围从 0 到 255。在显示卡上, 这三个数值被转化为显示器使用的三个类比电压, 电压决定了每个图素的强度。然而, 由於阴极射线管中电子枪的电子特性, 图素的强度 (I) 并不与电压 (V) 线性相关, 它们的关系为:

$$I = (V + \epsilon)^\gamma$$

ϵ 是由显示器的「亮度」控制设定的黑色等级 (理想值为 0)。指数 γ 由显示器的「图像」或「对比度」控制设定的。对于大多数显示器, γ 大约在 2.5 左右。

为了对此非线性作出补偿, 摄影机在线路内包含了「伽马修正」。指数 0.45 修正了进入摄影机的光线, 这意味著视讯显示器的伽马为 2.2。(视讯显示器的高伽马值增加了对比度, 这通常是不需要的, 因为周围的光线更适合於低对比度。)

视讯显示器的这个非线性反应实际上是很适当的, 这是因为人类对光线的反应也是非线性的。我曾提过, Y 被称为 CIE 亮度, 这是线性的光线度量。CIE 也定义了一个接近於人类感觉的亮度值。亮度是 L^* (发音为 "ell star"), 通过使用如下公式从 Y 计算得到的:

$$L^* = \begin{cases} 903.3 \frac{Y}{Y_n} & \frac{Y}{Y_n} \leq 0.008856 \\ 116 \left(\frac{Y}{Y_n} \right)^{\frac{1}{3}} - 16 & 0.008856 < \frac{Y}{Y_n} \end{cases}$$

在此 Y_n 是白色等级。公式的第一部分是一个小的线性部分。一般，人类的亮度感觉是与线性亮度的立方根相关的，这由第二个公式指出。 L^* 的范围从 0 到 100，每次 L^* 的增加都假定是人类能感觉到的亮度的最小变化。

根据知觉亮度而不是线性亮度对光线强度编码要更好一些。这使得位元的数量减少到一个合理的程度并且在类比线路上也降低了杂讯。

让我们来看一下整个程序。图素值 (P) 范围从 0 到 255，它被线性转化成电压等级，我们假定标准化为 0.0 到 1.0 之间的值。假设显示器的黑色级设定为 0，则图素的强度为：

$$I = V^r = \left(\frac{P}{255} \right)^r$$

这里 r 大约为 2.5。人类感觉的亮度 (L^*) 依赖于此强度的立方根和变化从 0 到 100 的范围，因此大约是：

$$L^* = 100 \left(\frac{P}{255} \right)^{\frac{r}{3}}$$

指数值大约为 0.85。如果指数值为 1，那么 CIE 亮度与图素值完全匹配。当然不完全是那种情况，但是如果图素值指出了线性亮度就非常接近。

BITMAPV4HEADER 的最後三个栏位为建立 DIB 的程式提供了一种为图素值指出假设的伽马值的方法。这些值由 16 位元整数值和 16 位元的小数值说明。例如，0x10000 为 1.0。如果 DIB 是捕捉实际影像而建立的，影像捕捉硬体就可能包含这个伽马值，并且可能是 2.2（编码为 0x23333）。如果 DIB 是由程式通过演算法产生的，程式会使用一个函式将它使用的任何线性亮度转化为 CIE 亮度。

第 5 版的 Header

为 Windows 98 和 Windows NT 5.0(即 Windows 2000)编写的程式能使用拥有新的 BITMAPV5HEADER 资讯结构的 DIB：

```
typedef struct
{
    DWORD          bV5Size ;           // size of the structure = 120
    LONG           bV5Width ;          // width of the image in pixels
    LONG           bV5Height ;         // height of the image in pixels
    WORD           bV5Planes ;         // = 1
    WORD           bV5BitCount ;       // bits per pixel (1,4,8,16,24,or32)
```

```

DWORD      bV5Compression ;          // compression code
DWORD      bV5SizeImage ;            // number of bytes in image
LONG       bV5XPelsPerMeter ;        // horizontal resolution
LONG       bV5YPelsPerMeter ;        // vertical resolution
DWORD      bV5ClrUsed ;              // number of colors used
DWORD      bV5ClrImportant ;         // number of important colors
DWORD      bV5RedMask ;              // Red color mask
DWORD      bV5GreenMask ;            // Green color mask
DWORD      bV5BlueMask ;             // Blue color mask
DWORD      bV5AlphaMask ;            // Alpha mask
DWORD      bV5CSType ;               // color space type
CIEXYZTRIPLE bV5Endpoints ;         // XYZ values
DWORD      bV5GammaRed ;             // Red gamma value
DWORD      bV5GammaGreen ;           // Green gamma value
DWORD      bV5GammaBlue ;           // Blue gamma value
DWORD      bV5Intent ;               // rendering intent
DWORD      bV5ProfileData ;          // profile data or filename
DWORD      bV5ProfileSize ;          // size of embedded data or filename
DWORD      bV5Reserved ;
}
BITMAPV5HEADER, * PBITMAPV5HEADER ;

```

这里有四个新栏位，只有其中三个有用。这些栏位支援 ICC Profile Format Specification，这是由「国际色彩协会 (International Color Consortium)」(由 Adobe、Agfa、Apple、Kodak、Microsoft、Silicon Graphics、Sun Microsystems 及其他公司组成) 建立的。您能在 <http://www.icc.org> 上取得这个标准的副本。基本上，每个输入（扫描器和摄影机）、输出（印表机和胶片记录器）以及显示（显示器）设备与将原始装置相关颜色（一般为 RGB 或 CMYK）联系到装置无关颜色规格的设定档案有关，最终依据 CIE XYZ 值来修正颜色。这些设定档案的副档名是 .ICM（指「图像颜色管理：image color management」）。设定档案能嵌入 DIB 档案中或从 DIB 档案连结以指出建立 DIB 的方式。您能在 /Platform SDK/Graphics and Multimedia Services/Color Management 中取得有关 Windows「图像颜色管理」的详细资讯。

BITMAPV5HEADER 的 bV5CSType 栏位能拥有几个不同的值。如果是 LCS_CALIBRATED_RGB，那么它就和 BITMAPV4HEADER 结构相容。bV5Endpoints 栏位和伽马栏位必须有效。

如果 bV5CSType 栏位是 LCS_sRGB，就不用设定剩余的栏位。预设的颜色空间是「标准」的 RGB 颜色空间，这是由 Microsoft 和 Hewlett-Packard 主要为 Internet 设计的，它包含装置无关的内容而不需要大量的设定档案。此文件位於 <http://www.color.org/contrib/sRGB.html>。

如果 bV5CSType 栏位是 LCS_Windows_COLOR_SPACE，就不用设定剩余的栏位。

Windows 通过 API 函式呼叫使用预设的颜色空间显示点阵图。

如果 bV5CSType 栏位是 PROFILE_EMBEDDED, 则 DIB 档案包含一个 ICC 设定档案。如果栏位是 PROFILE_LINKED, DIB 档案就包含了 ICC 设定档案的完整路径和档案名称。在这两种情况下, bV5ProfileData 都是从 BITMAPV5HEADER 开始到设定档案资料或档案名称起始位置的偏移量。bV5ProfileSize 栏位给出了资料或档案名的大小。不必设定 bV5Endpoints 和伽马栏位。

显示 DIB 资讯

现在让我们来看一些程式码。实际上我们并不未充分了解显示 DIB 的知识, 但至少能表从头结构上显示有关 DIB 的资讯。如程式 15-1 DIBHEADS 所示。

程式 15-1 DIBHEADS

```
DIBHEADS.C
/*-----
-
-      DIBHEADS.C --      Displays DIB Header Information
-                        (c) Charles Petzold, 1998
-      -----*
/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("DibHeads") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG              msg ;
    WNDCLASS         wndclass ;

    wndclass.style           = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName   = szAppName ;
```

```
if (!RegisterClass (&wndclass))
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("DIB Headers"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

hAccel = LoadAccelerators (hInstance, szAppName) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return msg.wParam ;
}

void Printf (HWND hwnd, TCHAR * szFormat, ...)
{
    TCHAR          szBuffer [1024] ;
    va_list        pArgList ;

    va_start (pArgList, szFormat) ;
    wvsprintf (szBuffer, szFormat, pArgList) ;
    va_end (pArgList) ;

    SendMessage (hwnd, EM_SETSEL, (WPARAM) -1, (LPARAM) -1) ;
    SendMessage (hwnd, EM_REPLACESEL, FALSE, (LPARAM) szBuffer) ;
    SendMessage (hwnd, EM_SCROLLCARET, 0, 0) ;
}

void DisplayDibHeaders (HWND hwnd, TCHAR * szFileName)
{
    static TCHAR    * szInfoName []= { TEXT ("BITMAPCOREHEADER"),
                                        TEXT ("BITMAPINFOHEADER"),
                                        TEXT ("BITMAPV4HEADER"),
                                        TEXT ("BITMAPV5HEADER") } ;
    Static TCHAR    * szCompression []={TEXT ("BI_RGB"),
```

```

TEXT      ("BI_RLE8"),

                                TEXT      ("BI_RLE4"),
                                TEXT      ("BI_BITFIELDS"),
                                TEXT      ("unknown") } ;

BITMAPCOREHEADER *    pbmch ;
BITMAPFILEHEADER *    pbmfh ;
BITMAPV5HEADER *      pbmih ;
BOOL              bSuccess ;
DWORD             dwFileSize, dwHighSize, dwBytesRead ;
HANDLE            hFile ;
int               i ;
PBYTE             pFile ;
TCHAR             * szV ;

                                // Display the file name

Printf (hwnd, TEXT ("File: %s\r\n\r\n"), szFileName) ;
                                // Open the file
hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
if (hFile == INVALID_HANDLE_VALUE)
{
    Printf (hwnd, TEXT ("Cannot open file.\r\n\r\n")) ;
    return ;
}

                                // Get the size of the file
dwFileSize = GetFileSize (hFile, &dwHighSize) ;
if (dwHighSize)
{
    Printf (hwnd, TEXT ("Cannot deal with >4G files.\r\n\r\n")) ;
    CloseHandle (hFile) ;
    return ;
}

                                // Allocate memory for the file
pFile = malloc (dwFileSize) ;
if (!pFile)
{
    Printf (hwnd, TEXT ("Cannot allocate memory.\r\n\r\n")) ;
    CloseHandle (hFile) ;
    return ;
}

                                // Read the file
SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
ShowCursor (TRUE) ;

bSuccess = ReadFile (hFile, pFile, dwFileSize, &dwBytesRead, NULL) ;

```



```

ShowCursor (FALSE) ;
SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

if (!bSuccess || (dwBytesRead != dwFileSize))
{
    Printf (hwnd, TEXT ("Could not read file.\r\n\r\n")) ;
    CloseHandle (hFile) ;
    free (pFile) ;
    return ;
}

// Close the file
CloseHandle (hFile) ;

// Display file size
Printf (hwnd, TEXT ("File size = %u bytes\r\n\r\n"), dwFileSize) ;
// Display BITMAPFILEHEADER structure
pbmfh = (BITMAPFILEHEADER *) pFile ;
Printf (hwnd, TEXT ("BITMAPFILEHEADER\r\n")) ;
Printf (hwnd, TEXT ("\t.bfType = 0x%X\r\n"), pbmfh->bfType) ;
Printf (hwnd, TEXT ("\t.bfSize = %u\r\n"), pbmfh->bfSize) ;
Printf (hwnd, TEXT ("\t.bfReserved1 = %u\r\n"),
pbmfh->bfReserved1) ;
Printf (hwnd, TEXT ("\t.bfReserved2 = %u\r\n"),
pbmfh->bfReserved2) ;
Printf (hwnd, TEXT ("\t.bfOffBits = %u\r\n\r\n"),
pbmfh->bfOffBits) ;

// Determine which information structure we have

pbmih = (BITMAPV5HEADER *) (pFile + sizeof (BITMAPFILEHEADER)) ;
switch (pbmih->bV5Size)
{
case sizeof (BITMAPCOREHEADER): i= 0 ; break ;
case sizeof (BITMAPINFOHEADER): i= 1 ; szV=
TEXT ("i") ; break ;
case sizeof (BITMAPV4HEADER): i= 2 ; szV=
TEXT ("V4") ; break ;
case sizeof (BITMAPV5HEADER): i= 3 ; szV=
TEXT ("V5") ; break ;
default:
Printf (hwnd, TEXT ("Unknown header size of %u.\r\n\r\n"),
pbmih->bV5Size) ;
free (pFile) ;
return ;
}

Printf (hwnd, TEXT ("%s\r\n"), szInfoName[i]) ;
// Display the BITMAPCOREHEADER fields

```

```

    if (pbmih->bV5Size == sizeof (BITMAPCOREHEADER))
    {
        pbmch = (BITMAPCOREHEADER *) pbmih ;
        Printf(hwnd,TEXT("\t.bcSize = %u\r\n"), pbmch->bcSize) ;
        Printf(hwnd,TEXT("\t.bcWidth = %u\r\n"), pbmch->bcWidth) ;
        Printf(hwnd,TEXT("\t.bcHeight = %u\r\n"), pbmch->bcHeight) ;
        Printf(hwnd,TEXT("\t.bcPlanes = %u\r\n"), pbmch->bcPlanes) ;
        Printf(hwnd,TEXT("\t.bcBitCount = %u\r\n\r\n"), pbmch->bcBitCount) ;
        free (pFile) ;
        return ;
    }

    // Display the BITMAPINFOHEADER fields
    Printf(hwnd,TEXT("\t.b%sSize = %u\r\n"), szV, pbmih->bV5Size) ;
    Printf(hwnd,TEXT("\t.b%sWidth = %i\r\n"), szV, pbmih->bV5Width) ;
    Printf(hwnd,TEXT("\t.b%sHeight = %i\r\n"), szV, pbmih->bV5Height) ;
    Printf(hwnd,TEXT("\t.b%sPlanes = %u\r\n"), szV, pbmih->bV5Planes) ;
    Printf(hwnd,TEXT("\t.b%sBitCount=%u\r\n"),szV, pbmih->bV5BitCount) ;
    Printf(hwnd,TEXT("\t.b%sCompression = %s\r\n"), szV,
        szCompression [min (4, pbmih->bV5Compression)]) ;
    Printf(hwnd,TEXT("\t.b%sSizeImage= %u\r\n"),szV,
        pbmih->bV5SizeImage) ;
    Printf(hwnd,TEXT ("\t.b%sXPelsPerMeter = %i\r\n"), szV,
        pbmih->bV5XPelsPerMeter) ;
    Printf(hwnd,TEXT ("\t.b%sYPelsPerMeter = %i\r\n"), szV,
        pbmih->bV5YPelsPerMeter) ;
    Printf (hwnd, TEXT ("\t.b%sClrUsed = %i\r\n"), szV,
        pbmih->bV5ClrUsed) ;
    Printf (hwnd, TEXT ("\t.b%sClrImportant = %i\r\n\r\n"), szV,
        pbmih->bV5ClrImportant) ;

    if (pbmih->bV5Size == sizeof (BITMAPINFOHEADER))
    {
        if (pbmih->bV5Compression == BI_BITFIELDS)
        {
            Printf (hwnd,TEXT("Red Mask = %08X\r\n"), pbmih->bV5RedMask) ;
            Printf (hwnd,TEXT ("Green Mask = %08X\r\n"), pbmih->bV5GreenMask) ;
            Printf (hwnd,TEXT ("Blue Mask = %08X\r\n\r\n"), pbmih->bV5BlueMask) ;
        }

        free (pFile) ;
        return ;
    }

    // Display additional BITMAPV4HEADER fields
    Printf (hwnd, TEXT ("\t.b%sRedMask = %08X\r\n"), szV,
        pbmih->bV5RedMask) ;
    Printf (hwnd, TEXT ("\t.b%sGreenMask = %08X\r\n"), szV,
        pbmih->bV5GreenMask) ;

```

```

    Printf (hwnd,      TEXT ("\t.b%sBlueMask = %08X\r\n"), szV,
                                           pbmih->bV5BlueMask) ;
    Printf (hwnd,      TEXT ("\t.b%sAlphaMask = %08X\r\n"), szV,
                                           pbmih->bV5AlphaMask) ;
    Printf (hwnd,      TEXT ("\t.b%sCSType = %u\r\n"), szV,
                                           pbmih->bV5CSType) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzX = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzX) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzY = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzY) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzZ = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzZ) ;
    Printf (hwnd,      TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzX      =
%08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzX) ;
    Printf (hwnd,      TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzY      =
%08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzY) ;
    Printf (hwnd,      TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzZ      =
%08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzZ) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzX = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzX) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzY = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzY) ;
    Printf (hwnd,      TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzZ = %08X\r\n"),
                                           szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzZ) ;
    Printf (hwnd,      TEXT ("\t.b%sGammaRed = %08X\r\n"), szV,
                                           pbmih->bV5GammaRed) ;
    Printf (hwnd,      TEXT ("\t.b%sGammaGreen = %08X\r\n"), szV,
                                           pbmih->bV5GammaGreen) ;
    Printf (hwnd,      TEXT ("\t.b%sGammaBlue = %08X\r\n\r\n"), szV,
                                           pbmih->bV5GammaBlue) ;

    if (pbmih->bV5Size == sizeof (BITMAPV4HEADER))
    {
        free (pFile) ;
        return ;
    }

```

```

        // Display additional BITMAPV5HEADER fields
        Printf          (hwnd,          TEXT ("\t.b%sIntent = %u\r\n"), szV,
pbmih->bV5Intent) ;
        Printf          (hwnd,          TEXT ("\t.b%sProfileData = %u\r\n"), szV,
                                pbmih->bV5ProfileData) ;
        Printf          (hwnd,          TEXT ("\t.b%sProfileSize = %u\r\n"), szV,
                                pbmih->bV5ProfileSize) ;
        Printf          (hwnd,          TEXT ("\t.b%sReserved = %u\r\n\r\n"), szV,
                                pbmih->bV5Reserved) ;

        free (pFile) ;
        return ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND          hwndEdit ;
    static OPENFILENAME ofn ;
    static TCHAR          szFileName      [MAX_PATH],      szTitleName
[MAX_PATH] ;
    static TCHAR          szFilter[]=          TEXT("Bitmap      Files
(*.BMP)\0*.bmp\0")

                                TEXT("All Files (*.*)\0*.*\0\0") ;

    switch (message)
    {
    case WM_CREATE:
        hwndEdit = CreateWindow (TEXT ("edit"), NULL,
                                WS_CHILD | WS_VISIBLE | WS_BORDER |
                                WS_VSCROLL | WS_HSCROLL |
                                ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
                                0, 0, 0, 0, hwnd, (HMENU) 1,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL) ;

        ofn.lStructSize          = sizeof (OPENFILENAME) ;
        ofn.hwndOwner            = hwnd ;
        ofn.hInstance            = NULL ;
        ofn.lpstrFilter          = szFilter ;
        ofn.lpstrCustomFilter    = NULL ;
        ofn.nMaxCustFilter      = 0 ;
        ofn.nFilterIndex        = 0 ;
        ofn.lpstrFile            = szFileName ;
        ofn.nMaxFile            = MAX_PATH ;
        ofn.lpstrFileTitle      = szTitleName ;
        ofn.nMaxFileTitle        = MAX_PATH ;
        ofn.lpstrInitialDir      = NULL ;
        ofn.lpstrTitle          = NULL ;
    }
}

```

```

        ofn.Flags                = 0 ;
        ofn.nFileOffset          = 0 ;
        ofn.nFileExtension       = 0 ;
        ofn.lpstrDefExt           = TEXT ("bmp") ;
        ofn.lCustData             = 0 ;
        ofn.lpfnHook              = NULL ;
        ofn.lpTemplateName       = NULL ;
        return 0 ;

    case WM_SIZE:
        MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam),
TRUE) ;

        return 0 ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_FILE_OPEN:
                if (GetOpenFileName (&ofn))
                    DisplayDibHeaders (hwndEdit, szFileName) ;

                return 0 ;
        }
        break ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

DIBHEADS.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

```

////////////////////////////////////
// Accelerator
DIBHEADS ACCELERATORS DISCARDABLE
BEGIN
    "O",          IDM_FILE_OPEN,          VIRTKEY, CONTROL, NOINVERT
END

```

```

////////////////////////////////////
// Menu
DIBHEADS MENU DISCARDABLE
BEGIN
    POPUP "&File"

```

```

        BEGIN
            MENUITEM "&Open\tCtrl+O",    IDM_FILE_OPEN
        END
    END
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by DibHeads.rc

#define IDM_FILE_OPEN            40001

```

此程式有一个简短的 WndProc 函式，它建立了一个唯读的编辑视窗来填满它的显示区域，它也处理功能表上的「File Open」命令。它通过呼叫 GetOpenFileName 函式使用标准的「File Open」对话方块，然後呼叫 DisplayDibHeaders 函式。此函式把整个 DIB 档案读入记忆体并逐栏地显示所有的表头资讯。

显示和列印

点阵图是用来看的。在这一节中，我们看一看 Windows 在视讯显示器上或列印页面上支援显示 DIB 的两个函式。要得到更好的性能，您可以使用一种兜圈子的方法来显示点阵图，我会在本章的後面讨论该方法，但先研究这两个函式会好一些。

这两个函式称为 SetDIBitsToDevice(发音为「set dee eye bits to device」) 和 StretchDIBits (发音为「stretch dee eye bits」)。每个函式都使用储存在记忆体中的 DIB 并能显示整个 DIB 或它的矩形部分。当使用 SetDIBitsToDevice 时，以图素为单位所显示映射的大小与 DIB 的图素大小相同。例如，一个 640×480 的 DIB 会占据整个标准的 VGA 萤幕，但在 300dpi 的雷射印表机上它只有约 2.1×1.6 英寸。StretchDIBits 能延伸和缩小 DIB 尺寸的行和列从而在输出设备上显示一个特定的大小。

了解 DIB

当呼叫两个函式之一来显示 DIB 时，您需要几个关于图像的资讯。正如我前面说过的，DIB 档案包含下列部分：



DIB 档案能被载入记忆体。如果除了档案表头外，整个档案被储存在记忆体的连续区块中，指向该记忆体块开始处（也就是资讯表头的开头）的指标被称为指向 packed DIB 的指标（见下图）。



这是通过剪贴簿传输 DIB 时所用的格式，并且也是您从 DIB 建立画刷时所用的格式。因为整个 DIB 由单个指标（如 pPackedDib）引用，所以 packed DIB 是在记忆体中储存 DIB 的方便方法，您可以把指标定义为指向 BYTE 的指标。使用本章前面所示的结构定义，能得到所有储存在 DIB 内的资讯，包括色彩对照表和个别图素位元。

然而，要想得到这么多资讯，还需要一些程式码。例如，您不能通过以下叙述简单地取得 DIB 的图素宽度：

```
iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;
```

DIB 有可能是 OS/2 相容格式的。在那种格式中，packed DIB 以 BITMAPCOREHEADER 结构开始，并且 DIB 的图素宽度和高度以 16 位元 WORD，而不是 32 位元 LONG 储存。因此，首先必须检查 DIB 是否为旧的格式，然後进行相对应的操作：

```
if (((PBITMAPCOREHEADER) pPackedDib)->bcSize == sizeof (BITMAPCOREHEADER))
    iWidth = ((PBITMAPCOREHEADER) pPackedDib)->bcWidth ;
else
    iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;
```

当然，这不很糟，但它不如我们所喜好的清晰。

现在有一个很有趣的实验：给定一个指向 packed DIB 的指标，我们要找出

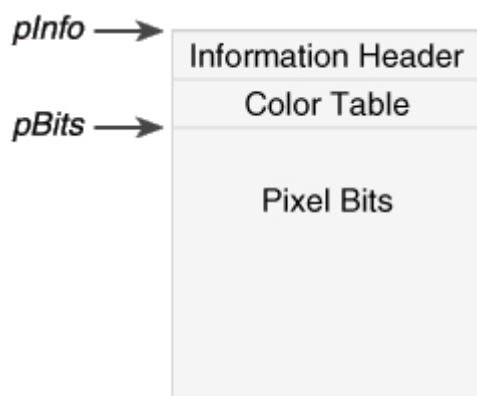
位於座标 (5, 27) 的图素值。即使假定 DIB 不是 OS/2 相容的格式, 您也需要了解 DIB 的宽度、高度和位元数。您需要计算每一列图素的位元组长度, 确定色彩对照表内的项目数, 以及色彩对照表是否包括三个 32 位元的颜色遮罩。您还需检查 DIB 是否被压缩, 在这种情况下图素是不能直接由位址得到的。

如果您需要直接存取所有的 DIB 图素 (就像许多图形处理工作一样), 这可能会增加一点处理时间。由於这个原因, 储存一个指向 packed DIB 的指标就很方便了, 不过这并不是一种有效率的解决方式。另一个漂亮的解决方法是为 DIB 定义一个包含足够成员资料的 C++ 类别, 从而允许快速随机地存取 DIB 图素。然而, 我曾经答应读者在本书内无需了解 C++, 我将在下一章说明一个 C 的解决方法。

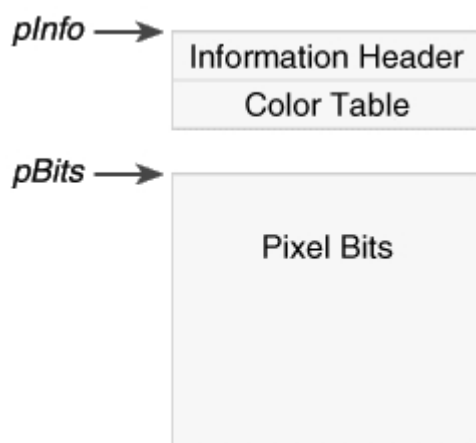
對於 SetDIBitsToDevice 和 StretchDIBits 函式, 需要的资讯包括一个指向 DIB 的 BITMAPINFO 结构的指标。您应回想起, BITMAPINFO 结构由 BITMAPINFOHEADER 结构和色彩对照表组成。因此这仅是一个指向 packed DIB 的指标。

函式也需要一个指向图素位元的指标。尽管程式码写得很不漂亮, 但这个指标还是可以从资讯表头内的资讯推出。注意, 当您存取 BITMAPFILEHEADER 结构的 bfOffBits 栏位时, 这个指标能很容易地计算出。bfOffBits 栏位指出了从 DIB 档案的开头到图素位元的偏移量。您可以简单地把此偏移量加到 BITMAPINFO 指标中, 然後减去 BITMAPFILEHEADER 结构的大小。然而, 当您从剪贴簿上得到指向 packed DIB 的指标时, 这并不起作用, 因为没有 BITMAPFILEHEADER 结构。

此图表显示了两个所需的指标:



SetDIBitsToDevice 和 StretchDIBits 函式需要两个指向 DIB 的指标, 因为这两个部分不在连续的记忆体块内。您可能有如下所示的两块记忆体:



确实，把 DIB 分成两个记忆体块是很有用的，只是我们更喜欢与整个 DIB 储存在单个记忆体块的 packed DIB 打交道。

除了这两个指标，SetDIBitsToDevice 和 StretchDIBits 函式通常也需要 DIB 的图素宽度和高度。如只想显示 DIB 的一部分，就不必明确地知道这些值，但它们会定义您在 DIB 图素位元阵列内定义的矩形的上限。

点对点图素显示

SetDIBitsToDevice 函式显示没有延伸和缩小的 DIB。DIB 的每个图素对应到输出设备的一个图素上，而且 DIB 中的图像一定会被正确显示出来——也就是说，图像的顶列在上方。任何会影响装置内容的座标转换都影响了显示 DIB 的开始位置，但不影响显示出来的图片大小和方向。该函式如下：

```

iLines = SetDIBitsToDevice (
    hdc,                // device context handle
    xDst,                // x destination coordinate
    yDst,                // y destination coordinate
    cxSrc,               // source rectangle width
    cySrc,               // source rectangle height
    xSrc,                // x source coordinate
    ySrc,                // y source coordinate
    yScan,               // first scan line to draw
    cyScans,             // number of scan lines to draw
    pBits,               // pointer to DIB pixel bits
    pInfo,               // pointer to DIB information
    fClrUse) ;           // color use flag
  
```

不要对参数的数量感到厌烦，在多数情况下，函式用起来比看起来要简单。不过在其他用途上来说，它的用法真的是乱七八糟，不过我们将学会怎么用它。

和 GDI 显示函式一样，SetDIBitsToDevice 的第一个参数是装置内容代号，它指出显示 DIB 的设备。下面两个参数 xDst 和 yDst，是输出设备的逻辑座标，并指出了显示 DIB 图像左上角的座标（「上端」指的是视觉上的上方，并不是 DIB 图素的第一行）。注意，这些都是逻辑座标，因此它们附属於实际上起作用

的任何坐标转换方式或——在 Windows NT 的情况下——设定的任何空间转换。在内定的 MM_TEXT 映射方式下，可以把这些参数设为 0，从显示平面上向左向上显示 DIB 图像。

您可以显示整个 DIB 图像或仅显示其中的一部分，这就是後四个参数的作用。但是 DIB 图素资料的由上而下的方向产生了许多误解，待会儿会谈谈到这些。现在应该清楚当显示整个 DIB 时，应把 xSrc 和 ySrc 设定为 0，并且 cxSrc 和 cySrc 应分别等於 DIB 的图素宽度和高度。注意，因为 BITMAPINFOHEADER 结构的 biHeight 栏位对於由上而下的 DIB 来说是负的，cySrc 应设定为 biHeight 栏位的绝对值。

此函式的文件（/Platform SDK/Graphics and Multimedia Services/GDI/Bitmaps/Bitmap Reference/Bitmap Functions/SetDIBitsToDevice）中说 xSrc、ySrc、cxSrc 和 cySrc 参数是逻辑单位。这是不正确的，它们是图素的座标和尺寸。对於 DIB 内的图素，拥有逻辑座标和单位是没有什么意义的。而且，不管是什么映射方式，在输出设备上显示的 DIB 始终是 cxSrc 图素宽和 cySrc 图素高。

现在先不详细讨论这两个参数 yScan 和 cyScan。这些参数在您从磁片档案或通过数据机读取资料时，透过每次显示 DIB 的一小部分减少对记忆体的需求。通常，yScan 设定为 0，cyScan 设定为 DIB 的高度。

pBits 参数是指向 DIB 图素位元的指标。pInfo 参数是指向 DIB 的 BITMAPINFO 结构的指标。虽然 BITMAPINFO 结构的位址与 BITMAPINFOHEADER 结构的位址相同，但是 SetDIBitsToDevice 结构被定义为使用 BITMAPINFO 结构，暗示著：对於 1 位元、4 位元和 8 位元 DIB，点阵图资讯表头後必须跟著色彩对照表。尽管 pInfo 参数被定义为指向 BITMAPINFO 结构的指标，它也是指向 BITMAPCOREINFO、BITMAPV4HEADER 或 BITMAPV5HEADER 结构的指标。

最後一个参数是 DIB_RGB_COLORS 或 DIB_PAL_COLORS，在 WINGDI.H 内分别定义为 0 和 1。如果您使用 DIB_RGB_COLORS，这意味著 DIB 包含了色彩对照表。DIB_PAL_COLORS 旗标指出，DIB 内的色彩对照表已经被指向在装置内容内选定并识别的调色盘的 16 位元索引代替。在下一章我们将学习这个选项。现在先使用 DIB_RGB_COLORS，或者是 0。

SetDIBitsToDevice 函式传回所显示的扫描行的数目。

因此，要呼叫 SetDIBitsToDevice 来显示整个 DIB 图像，您需要下列资讯：

- **hdc** 目的表面的装置内容代号
- **xDst** 和 **yDst** 图像左上角的目的座标
- **cxDib** 和 **cyDib** DIB 的图素宽度和高度，在这里，cyDib 是

BITMAPINFOHEADER 结构内 biHeight 栏位的绝对值。

- **pInfo** 和 **pBits** 指向点阵图资讯部分和图素位元的指标

然後用下列方法呼叫 SetDIBitsToDevice:

```
SetDIBitsToDevice (
    hdc,          // device context handle
    xDst,         // x destination coordinate
    yDst,         // y destination coordinate
    cxDib,        // source rectangle width
    cyDib,        // source rectangle height
    0,            // x source coordinate
    0,            // y source coordinate
    0,            // first scan line to draw
    cyDib,        // number of scan lines to draw
    pBits,        // pointer to DIB pixel bits
    pInfo,        // pointer to DIB information
    0) ;          // color use flag
```

因此，在 DIB 的 12 个参数中，四个设定为 0，一个是重复的。

程式 15-2 SHOWDIB1 通过使用 SetDIBitsToDevice 函式显示 DIB。

程式 15-2 SHOWDIB1

```
SHOWDIB1.C
/*-----
    SHOWDIB1.C --          Shows a DIB in the client area
                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG              msg ;
    WNDCLASS         wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
```

```

    wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName       = szAppName ;
    wndclass.lpszClassName      = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Show DIB #1"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    hAccel = LoadAccelerators (hInstance, szAppName) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static      BITMAPFILEHEADER * pbmfh ;
    static      BITMAPINFO        * pbmi ;
    static      BYTE               * pBits ;
    static      int                cxClient, cyClient, cxDib, cyDib ;
    static      TCHAR              szFileName [MAX_PATH], szTitleName
[MAX_PATH] ;
    BOOL        bSuccess ;
    HDC         hdc ;
    PAINTSTRUCT ps ;

    switch (message)
    {
    case WM_CREATE:
        DibFileInitialize (hwnd) ;

```

```

        return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_INITMENUPOPUP:
    EnableMenuItem ((HMENU) wParam, IDM_FILE_SAVE,
        pbmfh ? MF_ENABLED : MF_GRAYED) ;
    return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
        // Show the File Open dialog box

        if (!DibFileOpenDlg (hwnd, szFileName,
szTitleName))

            return 0 ;

        // If there's an existing DIB, free the memory

        if (pbmfh)
        {
            free (pbmfh) ;
        }
        pbmfh = NULL ;

        // Load the entire DIB into memory

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        pbmfh = DibLoadImage (szFileName) ;
        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        // Invalidate the client area for later update

        InvalidateRect (hwnd, NULL, TRUE) ;

        if (pbmfh == NULL)
        {
            MessageBox (    hwnd, TEXT ("Cannot load DIB file"),
                szAppName, 0) ;
            return 0 ;
        }
    }

```

```

        // Get pointers to the info structure & the bits

        pbmi = (BITMAPINFO *) (pbmfh + 1) ;
        pBits = (BYTE *) pbmfh +
pbmfh->bfOffBits ;

        // Get the DIB width and height

        if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        {
            cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
            cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
        }
        else
        {
            cxDib = pbmi->bmiHeader.biWidth ;
            cyDib = abs( pbmi->bmiHeader.biHeight) ;
        }
        return 0 ;

        case IDM_FILE_SAVE:
        // Show the File Save dialog box

            if (!DibFileSaveDlg (hwnd, szFileName,
szTitleName))

                return 0 ;

            // Save the DIB to memory

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            bSuccess = DibSaveImage (szFileName, pbmfh) ;
            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            if (!bSuccess)
                MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
                    szAppName, 0) ;
                return 0 ;
        }
        break ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        if (pbmfh)
            SetDIBitsToDevice (hdc,

```

```

        0,                // xDst
        0,                // yDst
        cxDib,            // cxSrc
        cyDib,            // cySrc
        0,                // xSrc
        0,                // ySrc
        0,                // first scan line
        cyDib,            // number of scan lines
        pBits,
        pbmi,
        DIB_RGB_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    if (pbmfh)
        free (pbmfh) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

DIBFILE.H
/*-----
    DIBFILE.H -- Header File for DIBFILE.C
    -----*/

void DibFileInitialize (HWND hwnd) ;
BOOL DibFileOpenDlg    (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;
BOOL DibFileSaveDlg    (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;

BITMAPFILEHEADER *    DibLoadImage    (PTSTR pstrFileName) ;
BOOL                  DibSaveImage(PTSTR pstrFileName, BITMAPFILEHEADER *) ;

DIBFILE.C
/*-----
-
    DIBFILE.C -- DIB File Functions
    -----
*/

#include <windows.h>
#include <commdlg.h>
#include "dibfile.h"

static OPENFILENAME ofn ;
void DibFileInitialize (HWND hwnd)
{
    static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0") \

```

```

        TEXT ("All Files (*.*)\0*\.*\0\0") ;
    ofn.lStructSize      = sizeof (OPENFILENAME) ;
    ofn.hwndOwner        = hwnd ;
    ofn.hInstance        = NULL ;
    ofn.lpstrFilter      = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter   = 0 ;
    ofn.nFilterIndex     = 0 ;
    ofn.lpstrFile        = NULL ; // Set in Open and Close functions
    ofn.nMaxFile         = MAX_PATH ;
    ofn.lpstrFileName    = NULL ; // Set in Open and Close functions
    ofn.nMaxFileTitle    = MAX_PATH ;
    ofn.lpstrInitialDir  = NULL ;
    ofn.lpstrTitle       = NULL ;
    ofn.Flags            = 0 ;      // Set in Open and Close functions
    ofn.nFileOffset      = 0 ;
    ofn.nFileExtension   = 0 ;
    ofn.lpstrDefExt      = TEXT ("bmp") ;
    ofn.lCustData        = 0 ;
    ofn.lpfnHook         = NULL ;
    ofn.lpTemplateName   = NULL ;
}

BOOL DibFileOpenDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
    ofn.hwndOwner        = hwnd ;
    ofn.lpstrFile        = pstrFileName ;
    ofn.lpstrFileName    = pstrTitleName ;
    ofn.Flags            = 0 ;

    return GetOpenFileName (&ofn) ;
}

BOOL DibFileSaveDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
    ofn.hwndOwner        = hwnd ;
    ofn.lpstrFile        = pstrFileName ;
    ofn.lpstrFileName    = pstrTitleName ;
    ofn.Flags            = OFN_OVERWRITEPROMPT ;

    return GetSaveFileName (&ofn) ;
}

BITMAPFILEHEADER * DibLoadImage (PTSTR pstrFileName)
{
    BOOL                bSuccess ;
    DWORD               dwFileSize, dwHighSize, dwBytesRead ;
    HANDLE              hFile ;
    BITMAPFILEHEADER * pbmfh ;

```



```
hFile = CreateFile ( pstrFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;

if ( hFile == INVALID_HANDLE_VALUE)
    return NULL ;

dwFileSize = GetFileSize (hFile, &dwHighSize) ;

if (dwHighSize)
{
    CloseHandle (hFile) ;
    return NULL ;
}

pbmfh = malloc (dwFileSize) ;
if (!pbmfh)
{
    CloseHandle (hFile) ;
    return NULL ;
}

bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
CloseHandle (hFile) ;

if (!bSuccess || (dwBytesRead != dwFileSize)
    || (pbmfh->bfType != * (WORD *) "BM")
    || (pbmfh->bfSize != dwFileSize))
{
    free (pbmfh) ;
    return NULL ;
}
return pbmfh ;
}

BOOL DibSaveImage (PTSTR pstrFileName, BITMAPFILEHEADER * pbmfh)
{
    BOOL bSuccess ;
    DWORD dwBytesWritten ;
    HANDLE hFile ;

    hFile = CreateFile ( pstrFileName, GENERIC_WRITE, 0, NULL,
        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;

    if (hFile == INVALID_HANDLE_VALUE)
        return FALSE ;
    bSuccess = WriteFile (hFile, pbmfh, pbmfh->bfSize, &dwBytesWritten, NULL) ;
    CloseHandle (hFile) ;
}
```

```

    if (!bSuccess || (dwBytesWritten != pbmfh->bfSize))
    {
        DeleteFile (pstrFileName) ;
        return FALSE ;
    }
    return TRUE ;
}

SHOWDIB1.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB1 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open...", IDM_FILE_OPEN
        MENUITEM "&Save...", IDM_FILE_SAVE
    END
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib1.rc

#define IDM_FILE_OPEN        40001
#define IDM_FILE_SAVE        40002

```

DIBFILE.C 档案包含了显示「File Open」和「File Save」对话方块的常式，以及把整个 DIB 档案（拥有 BITMAPFILEHEADER 结构）载入单个记忆体块的常式。程式也会将这样一个记忆体区写出到档案。

当在 SHOWDIB1.C 内执行「File Open」命令载入 DIB 档案後，程式计算记忆体块中 BITMAPINFOHEADER 结构和图素位元的偏移量，程式也获得 DIB 的图素宽度和高度。所有资讯都储存在静态变数中。在处理 WM_PAINT 讯息处理期间，程式通过呼叫 SetDIBitsToDevice 显示 DIB。

当然，SHOWDIB1 还缺少一些功能。例如，如果 DIB 对显示区域来说太大，则没有卷动列可用来移动查看。在下一章的末尾将修改这些缺陷。

DIB 的颠倒世界

我们将得到一个重要的教训，它不仅在生活中重要，而且在作业系统的应用程式介面的设计中也重要。这个教训是：覆水难收。

回到 OS/2 Presentation Manager 那由下而上的 DIB 图素位元的定义处, 这样的定义是有点道理的, 因为 PM 内的任何坐标系都有一个内定的左下角原点。例如: 在 PM 视窗内, 内定的 (0,0) 原点是视窗的左下角。(如果您觉得这很古怪, 很多人和您的感觉一样。如果您不觉得古怪, 那您可能是位数学家。) 点阵图的绘制函式也根据左下角指定目的地。

因此, 在 OS/2 内如果给点阵图指定了目的座标 (0,0), 则图像将从视窗的左下角向上向右显示, 如图 15-1 所示。

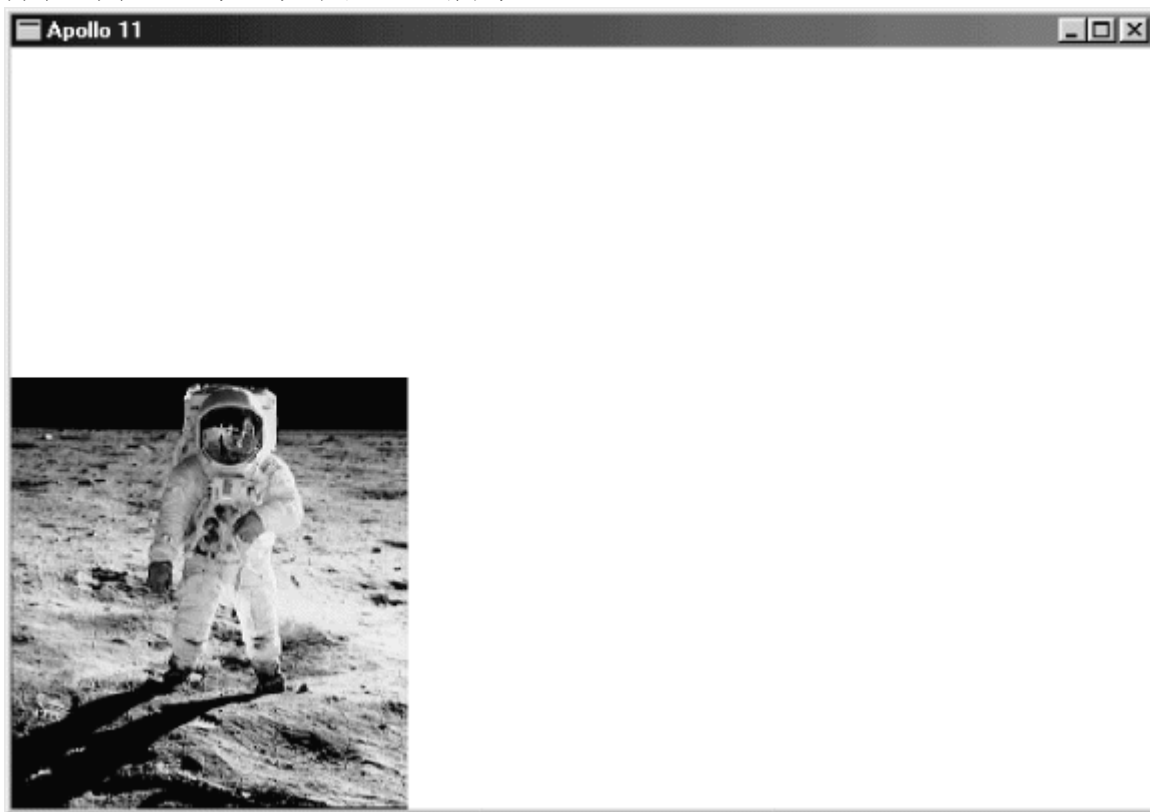


图 15-1 在 OS/2 中以 (0,0) 为目的点显示的点阵图

在够慢的机器上, 您能实际看到电脑由下而上绘制点阵图。

尽管 OS/2 座标系统显得很古怪, 但它的优点是高度的一致。点阵图的 (0,0) 原点是点阵图档案中第一行的第一个图素, 并且此图素被映射到在点阵图绘制函式中指定的目的座标上。

Windows 存在的问题是不能保持内部的一致性。当您只要显示整个 DIB 图像中的一小块矩形时, 就要使用参数 xSrc、ySrc、cxSrc 和 cySrc。这些来源座标和大小与 DIB 资料的第一行 (图像的最後一行) 相关。这方面与 OS/2 相似, 与 OS/2 不同的是, Windows 在目的座标上显示图像的顶列。因此, 如果显示整个 DIB 图像, 显示在 (xDst, yDst) 的图素是位於座标 (0, cyDib - 1) 处的图素。DIB 资料的最後一列就是图形的顶列。如果仅显示图像的一部分, 则在 (xDst, yDst) 显示的图素是位於座标 (xSrc, ySrc + cySrc - 1) 处的 DIB 图素。

图 15-2 显示的图表将帮助您理解这方面的内容。您可以把下面显示的 DIB 当成是储存在记忆体中的——就是说，上下颠倒。座标的原点与 DIB 图素资料的第一个位元是一致的。SetDIBitsToDevice 的 xSrc 参数是以 DIB 的左边为基准，并且 cxSrc 是 xSrc 右边的图像宽度，这很直观。ySrc 参数以 DIB 资料的首列（也就是图像的底部）为基准，并且 cySrc 是从 ySrc 到资料的末列（图像的顶端）的图像高度。

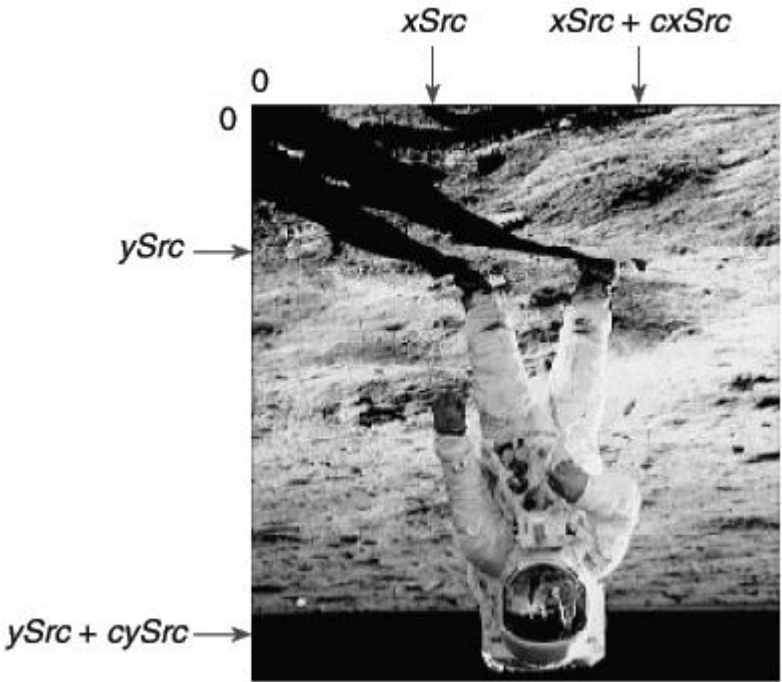


图 15-2 正常 DIB（由下而上）的座标

如果目的装置内容具有使用 MM_TEXT 映射方式的内定图素座标，来源矩形和目的矩形角落座标之间的关系显示在表 15-3 中。

表 15-3

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cySrc - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxSrc - 1, yDst + cySrc - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxSrc - 1, yDst)

(xSrc, ySrc) 不映射到 (xDst, yDst)，使得表格显得很混乱。在其他映射方式中，点 (xSrc, ySrc + cySrc - 1) 总是映射到逻辑点 (xDst, yDst)，图像也与 MM_TEXT 所显示的一样。

到目前为止，我们讨论了当 BITMAPINFOHEADER 结构的 biHeight 栏位是正值时的正常情况。如果 biHeight 栏位是负值，则 DIB 资料会以合理的由上而下的方式排列。您可能会认为这样将解决所有问题，如果您真地这样认为，那您就错了。

很明显地，有人会认为如果把 DIB 上下倒置，旋转每一行，然後给 biHeight 设定一个正值，它将像正常的由下而上的 DIB 一样操作，所有与 DIB 矩形相关的现存程式码就不必修改。我认为这是一个合理的目的，但它忘记了一个事实，程式需要修改以处理由上而下的 DIB，这样就不会使用一个负高度。

而且，此决定的结果意味著由上而下的 DIB 的来源座标在 DIB 资料的最後一列有一个原点，它也是图像的底列。这与我们遇到的情况完全不同。位於 (0, 0) 原点的 DIB 图素不再是 pBits 指标引用的第一个图素，也不是 DIB 档案的最後一个图素，它位於两者之间。

图 15-3 显示的图表说明了在由上而下的 DIB 中指定矩形的方法，也是它储存在档案或记忆体中的样子。

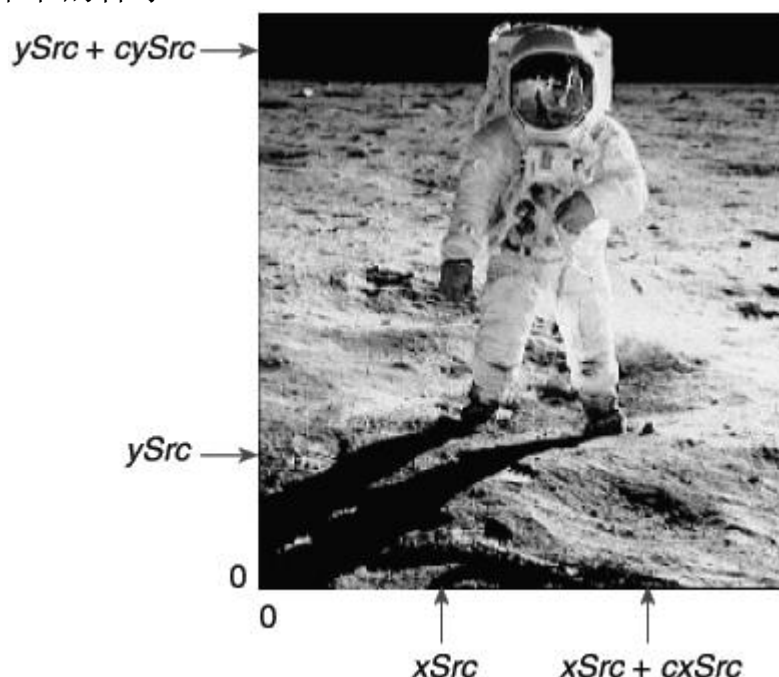


图 15-3 指定由上而下的 DIB 的座标

无论如何，这个方案的实际优点是 SetDIBitsToDevice 函式的参数与 DIB 资料的方向无关。如果有显示了同一图像的两个 DIB（一个由下而上，另一个由上而下。表示在两个 DIB 档案内的列顺序相反），您可以使用相同的参数呼叫 SetDIBitsToDevice 来选择显示图像的相同部分。

如程式 15-3 APOLL011 中所示。

程式 15-3 APOLL011

```
APOLL011.C
/*-----
    APOLL011.C --      Program for screen captures
                      (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
```

```

#include "dibfile.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Apollo11") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLAS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }
    hwnd = CreateWindow (szAppName, TEXT ("Apollo 11"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{

```

```

static BITMAPFILEHEADER    * pbmfh [2] ;
static BITMAPINFO          * pbmi  [2] ;
static BYTE                * pBits [2] ;
static int                 cxClient, cyClient, cxDib[2], cyDib[2] ;
HDC                        hdc ;
PAINTSTRUCT                ps ;

switch (message)
{
case WM_CREATE:
    pbmfh[0] = DibLoadImage (TEXT ("Apollo11.bmp")) ;
    pbmfh[1] = DibLoadImage (TEXT ("ApolloTD.bmp")) ;

    if (pbmfh[0] == NULL || pbmfh[1] == NULL)
    {
        MessageBox (hwnd, TEXT ("Cannot load DIB file"),
            szAppName, 0) ;
        return 0 ;
    }

    // Get pointers to the info structure & the
bits
    pbmi [0] = (BITMAPINFO *) (pbmfh[0] + 1) ;
    pbmi [1] = (BITMAPINFO *) (pbmfh[1] + 1) ;
    pBits [0] = (BYTE *) pbmfh[0] + pbmfh[0]->bfOffBits ;
    pBits [1] = (BYTE *) pbmfh[1] + pbmfh[1]->bfOffBits ;

    // Get the DIB width and height (assume BITMAPINFOHEADER)
    // Note that cyDib is the absolute value of the header
value!!!

    cxDib [0] = pbmi[0]->bmiHeader.biWidth ;
    cxDib [1] = pbmi[1]->bmiHeader.biWidth ;

    cyDib [0] = abs (pbmi[0]->bmiHeader.biHeight) ;
    cyDib [1] = abs (pbmi[1]->bmiHeader.biHeight) ;
    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Bottom-up DIB full size

```

```
        SetDIBitsToDevice (hdc,
0,                                // xDst
cyClient / 4,                    // yDst
cxDib[0],                        // cxSrc
cyDib[0],                        // cySrc
0,                                // xSrc
0,                                // ySrc
0,                                // first scan line
cyDib[0],                        // number of scan lines
pBits[0],
pbmi[0],
DIB_RGB_COLORS) ;

        // Bottom-up DIB partial

        SetDIBitsToDevice (hdc,
240,                                // xDst
cyClient / 4,                    // yDst
80,                                // cxSrc
166,                                // cySrc
80,                                // xSrc
60,                                // ySrc
0,                                // first scan line
cyDib[0],                        // number of scan lines
pBits[0],
pbmi[0],
DIB_RGB_COLORS) ;

        // Top-down DIB full size

        SetDIBitsToDevice      (hdc,
340,                                // xDst
cyClient / 4,                    // yDst
cxDib[0],                        // cxSrc
cyDib[0],                        // cySrc
0,                                // xSrc
0,                                // ySrc
0,                                // first scan line
cyDib[0],                        // number of scan lines
pBits[0],
pbmi[0],
DIB_RGB_COLORS) ;

        // Top-down DIB partial

        SetDIBitsToDevice      (hdc,
580,                                // xDst
cyClient / 4,                    // yDst
```



```
        80,                // cxSrc
        166,              // cySrc
        80,                // xSrc
        60,                // ySrc
        0,                // first scan line
        cyDib[1],          // number of scan lines
        pBits[1],
        pbmi[1],
        DIB_RGB_COLORS) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY:
        if (pbmfh[0])
                free (pbmfh[0]) ;
        if (pbmfh[1])
                free (pbmfh[1]) ;

        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

程式载入了名为 APOLL011.BMP (由下而上版本) 和 APOLL0TD.BMP (由上而下版本) 的两个 DIB。它们都是 220 图素宽和 240 图素高。注意, 在程式从表头资讯结构中确定 DIB 的宽度和高度时, 它使用 abs 函式得到 biHeight 栏位的绝对值。当以全部大小或范围显示 DIB 时, 不管显示点阵图的种类, xSrc、ySrc、cxSrc 和 cySrc 座标都是相同的。结果如图 15-4 所示。

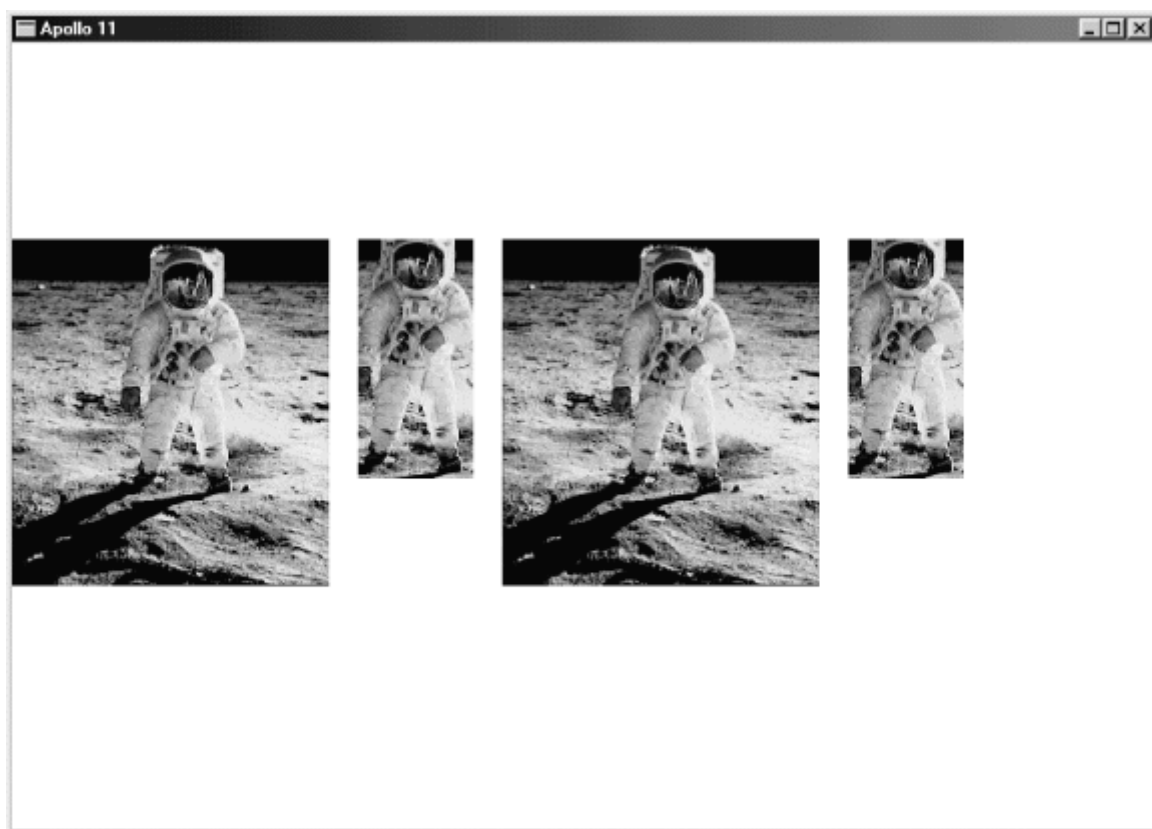


图 15-4 APOLL011 的萤幕显示

注意，「第一条扫描线」和「扫描线数目」参数保持不变，我将在以后简短说明。pBits 参数也不变，不要只为了使它指向您需要显示的区域而试图更改 pBits。

我在这个问题上花了这么多时间，并不是因为要让那些试图跟 API 定义中有问题的部分妥协的 Windows 程式写作者难堪，而是想让您不至於因为这个令人混淆的问题而紧张起来。这个问题之所以令人困惑，是因为它本身早就被搞混了。

我也想让您留意 Windows 文件中的某些叙述，例如对 SetDIBitsToDevice，文件说：「由下而上 DIB 的原点是点阵图的左下角；由上而下 DIB 的原点是左上角」。这不仅模糊，而且是错误的。我可以用更好的方式来讲述：由下而上 DIB 的原点是点阵图图像的左下角，它是点阵图资料的第一列的第一个图素。由上而下 DIB 的原点也是点阵图图像的左下角，但在这种情况下，左下角是点阵图资料的最後一列的第一个图素。

如果要撰写存取 DIB 个别位元的函式，问题会变的更糟。这应该与您为显示部分 DIB 映射而指定的座标一致，我的解决方法是（我将在第十六章的 DIB 程式库中使用）以统一的手法参考 DIB 图素和座标，就像在图像被正确显示时（0,0）原点所指的是 DIB 图像顶行的最左边的图素一样。

循序显示

拥有大量记忆体能确保程式更容易地执行。要显示磁片档案内的 DIB, 可以分为两个独立的工作: 将 DIB 载入记忆体, 然後显示它。

然而, 您也可能在不把整个档案载入记忆体的情况下显示 DIB。即使有足够的实体记忆体提供给 DIB, 把 DIB 移入记忆体也会迫使 Windows 的虚拟记忆体系统把记忆体中别的资料和程式码移到磁片上。如果 DIB 仅用於显示并立即从记忆体中消除, 这就非常讨厌。

还有另一个问题: 假设 DIB 位於例如软碟的慢速储存媒体上, 或由数据机传输过来, 或者来自扫描器或视频截取程式取得图素资料的转换常式。您是否得等到整个 DIB 被载入记忆体後才显示它? 还是从磁片或电话线或扫描器上得到 DIB 时, 就开始显示它?

解决这些问题是 SetDIBitsToDevice 函式中 yScan 和 cyScans 参数的目的。要使用这个功能, 需要多次呼叫 SetDIBitsToDevice, 大多数情况下使用同样的参数。然而對於每次呼叫, pBits 参数指向点阵图图素总体排列的不同部分。yScans 参数指出了 pBits 指向图素资料的行, cyScans 参数是被 pBits 引用的行数。这大量地减少了记忆体需求。您仅需要为储存 DIB 的资讯部分 (BITMAPINFOHEADER 结构和色彩对照表) 和至少一行图素资料配置足够的记忆体。

例如, 假设 DIB 有 23 行图素, 您希望每次最多 5 行的分段显示这个 DIB。您可能想配置一个由变数 pInfo 引用的记忆体块来储存 DIB 的 BITMAPINFO 部分, 然後从档案中读取该 DIB。在检查完此结构的栏位後, 能够计算出一行的位元组长度。乘以 5 并配置该大小的另一个记忆体块 (pBits)。现在读取前 5 行, 呼叫您正常使用的函式, 把 yScan 设定为 0, 把 cyScans 设定为 5。现在从档案中读取下 5 行, 这一次将 yScan 设定为 5, 继续将 yScan 设定为 10, 然後为 15。最後, 将最後 3 行读入 pBits 指向的记忆体块, 并将 yScan 设定为 20, 将 cyScans 设定为 3, 以呼叫 SetDIBitsToDevice。

现在有一个不好的讯息。首先, 使用 SetDIBitsToDevice 的这个功能要求程式的资料取得和资料显示元素之间结合得相当紧密。这通常是不理想的, 因为您必须在获得资料和显示资料之间切换。首先, 您将延缓整个程序; 第二, SetDIBitsToDevice 是唯一具有这个功能的点阵图显示函式。StretchDIBits 函式不包括这个功能, 因此您不能使用它以不同图素大小显示发表的 DIB。您必须呼叫 StretchDIBits 多次, 每次更改 BITMAPINFOHEADER 结构中的资讯, 并在萤幕的不同区域显示结果。

程式 15-4 SEQDISP 展示了这个功能的使用方法。

程式 15-4 SEQDISP

```

SEQDISP.C
/*-----
--
        SEQDISP.C --      Sequential Display of DIBs
                           (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("SeqDisp") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS         wndclass ;

    wndclass.style           = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName   = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("DIB Sequential Display"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;

```

```

UpdateWindow (hwnd) ;

hAccel = LoadAccelerators (hInstance, szAppName) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static BITMAPINFO      *    pbmi ;
    static BYTE            *    pBits ;
    static int              cxDib, cyDib, cBits ;
    static OPENFILENAME ofn ;
    static TCHAR            szFileName [MAX_PATH], szTitleName
[MAX_PATH] ;
    static TCHAR            szFilter[]= TEXT ("Bitmap Files
(*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
    BITMAPFILEHEADER        bmfh ;
    BOOL                    bSuccess, bTopDown ;
    DWORD                   dwBytesRead ;
    HANDLE                  hFile ;
    HDC                      hdc ;
    HMENU                   hMenu ;
    int                     iInfoSize, iBitsSize, iRowLength, y ;
    PAINTSTRUCT              ps ;

    switch (message)
    {
    case WM_CREATE:
        ofn.lStructSize          = sizeof (OPENFILENAME) ;
        ofn.hwndOwner            = hwnd ;
        ofn.hInstance            = NULL ;
        ofn.lpstrFilter          = szFilter ;
        ofn.lpstrCustomFilter    = NULL ;
        ofn.nMaxCustFilter       = 0 ;
        ofn.nFilterIndex         = 0 ;
        ofn.lpstrFile            = szFileName ;
        ofn.nMaxFile             = MAX_PATH ;
        ofn.lpstrFileTitle       = szTitleName ;
        ofn.nMaxFileTitle        = MAX_PATH ;

```

```

        ofn.lpstrInitialDir      = NULL ;
        ofn.lpstrTitle          = NULL ;
        ofn.Flags                = 0 ;
        ofn.nFileOffset         = 0 ;
        ofn.nFileExtension      = 0 ;
        ofn.lpstrDefExt          = TEXT ("bmp") ;
        ofn.lCustData            = 0 ;
        ofn.lpfHook              = NULL ;
        ofn.lpTemplateName      = NULL ;
        return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
        // Display File Open dialog
        if (!GetOpenFileName (&ofn))
            return 0 ;

        // Get rid of old DIB

        if (pbmi)
        {
            free (pbmi) ;
            pbmi = NULL ;
        }

        if (pBits)
        {
            free (pBits) ;
            pBits = NULL ;
        }

        // Generate WM_PAINT message to erase background

        InvalidateRect (hwnd, NULL, TRUE) ;
        UpdateWindow (hwnd) ;

        // Open the file

        hFile = CreateFile (szFileName,
GENERIC_READ,
        FILE_SHARE_READ, NULL, OPEN_EXISTING,
        FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;

        if (hFile == INVALID_HANDLE_VALUE)

```

```
        {
            MessageBox ( hwnd, TEXT ("Cannot open file."),
                        szAppName, MB_ICONWARNING | MB_OK) ;
            return 0 ;
        }

        // Read in the BITMAPFILEHEADER

        bSuccess      =      ReadFile      (hFile,&bmfh,      sizeof
(BITMAPFILEHEADER),
            &dwBytesRead, NULL) ;

        if (!bSuccess || dwBytesRead!=sizeof (BITMAPFILEHEADER))
        {
            MessageBox (hwnd, TEXT ("Cannot read file."),
                        szAppName, MB_ICONWARNING | MB_OK) ;
            CloseHandle (hFile) ;
            return 0 ;
        }

        // Check that it's a bitmap

        if (bmfh.bfType != * (WORD *) "BM")
        {
            MessageBox (hwnd, TEXT ("File is not a bitmap."),
                        szAppName, MB_ICONWARNING | MB_OK) ;
            CloseHandle (hFile) ;
            return 0 ;
        }

        // Allocate memory for header and bits

        iInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
        iBitsSize = bmfh.bfSize - bmfh.bfOffBits ;

        pbmi = malloc (iInfoSize) ;
        pBits      = malloc (iBitsSize) ;

        if (pbmi == NULL || pBits == NULL)
        {
            MessageBox (hwnd, TEXT ("Cannot allocate memory."),
                        szAppName, MB_ICONWARNING | MB_OK) ;
            if (pbmi)
                free (pbmi) ;
            if (pBits)
                free (pBits) ;
            CloseHandle (hFile) ;
            return 0 ;
        }
```

```

    }

    // Read in the Information Header

    bSuccess = ReadFile (hFile, pbmi, iInfoSize,    &dwBytesRead,
NULL) ;

        if (!bSuccess || (int) dwBytesRead != iInfoSize)
        {
            MessageBox (hwnd, TEXT ("Cannot read file."),
                szAppName, MB_ICONWARNING | MB_OK) ;
            if (pbmi)
                free (pbmi) ;
            if (pBits)
                free (pBits) ;
            CloseHandle (hFile) ;
            return 0 ;
        }

    // Get the DIB width and height

    bTopDown = FALSE ;

    if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    {
        cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
        cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
        cBits = ((BITMAPCOREHEADER *) pbmi)->bcBitCount ;
    }
    else
    {
        if (pbmi->bmiHeader.biHeight < 0)
            bTopDown = TRUE ;

        cxDib =          pbmi->bmiHeader.biWidth ;
        cyDib = abs      (pbmi->bmiHeader.biHeight) ;
        cBits =          pbmi->bmiHeader.biBitCount ;

        if (pbmi->bmiHeader.biCompression != BI_RGB &&
            pbmi->bmiHeader.biCompression != BI_BITFIELDS)
        {
            MessageBox (hwnd, TEXT ("File is compressed."),
                szAppName, MB_ICONWARNING | MB_OK) ;
            if (pbmi)
                free (pbmi) ;
            if (pBits)
                free (pBits) ;
            CloseHandle (hFile) ;

```



```

        return 0 ;
    }

    // Get the row length

    iRowLength = ((cxDib * cBits + 31) & ~31) >> 3 ;

    // Read and display
    SetCursor (LoadCursor (NULL,
IDC_WAIT)) ;

    ShowCursor (TRUE) ;

    hdc = GetDC (hwnd) ;

    for (y = 0 ; y < cyDib ; y++)
    {
        ReadFile (hFile, pBits + y * iRowLength, iRowLength, &dwBytesRead,
NULL) ;
        SetDIBitsToDevice (hdc,
            0, // xDst
            0, // yDst
            cxDib, // cxSrc
            cyDib, // cySrc
            0, // xSrc
            0, // ySrc
            bTopDown ? cyDib - y - 1 : y,
            // first scan line
            1, // number of scan lines
            pBits + y * iRowLength,
            pbmi,
            DIB_RGB_COLORS) ;
    }
    ReleaseDC (hwnd, hdc) ;
    CloseHandle (hFile) ;
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
    return 0 ;
}
break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (pbmi && pBits)
        SetDIBitsToDevice (hdc,
            0, // xDst
            0, // yDst

```

```

        cxDib,                // cxSrc
        cyDib,                // cySrc
        0,                    // xSrc
        0,                    // ySrc
        0,                    // first scan line
        cyDib,                // number of scan lines
        pBits,
    DIB_RGB_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    if (pbmi)
        free (pbmi) ;

    if (pBits)
        free (pBits) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

SEQDISP.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

```

////////////////////////////////////
/
// Accelerator
SEQDISP ACCELERATORS DISCARDABLE
BEGIN
    "O",    IDM_FILE_OPEN,    VIRTKEY, CONTROL, NOINVERT
END

```

```

////////////////////////////////////
/
// Menu
SEQDISP MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open...\tCtrl+O",  IDM_FILE_OPEN
    END
END

```

RESOURCE.H (摘录)

// Microsoft Developer Studio generated include file.

```
// Used by SeqDisp.rc
```

```
#define IDM_FILE_OPEN          40001
```

在处理「File Open」功能表命令期间，在 SEQDISP.C 内的所有档案 I/O 都会发生。在处理 WM_COMMAND 的最後，程式进入读取单行图素并用 SetDIBitsToDevice 显示该行图素的回圈。整个 DIB 储存在记忆体中以便在处理 WM_PAINT 期间也能显示它。

缩放到合适尺寸

SetDIBitsToDevice 完成了将 DIB 的图素对点送入输出设备的显示程序。这對於列印 DIB 用处不大。印表机的解析度越高，得到的图像就越小，您最终会得到如邮票大小的图像。

要通过缩小或放大 DIB，在输出设备上以特定的大小显示它，可以使用 StretchDIBits:

```
iLines = StretchDIBits (
    hdc,                // device context handle
    xDst,                // x destination coordinate
    yDst,                // y destination coordinate
    cxDst,               // destination rectangle width
    cyDst,               // destination rectangle height
    xSrc,                // x source coordinate
    ySrc,                // y source coordinate
    cxSrc,               // source rectangle width
    cySrc,               // source rectangle height
    pBits,               // pointer to DIB pixel bits
    pInfo,               // pointer to DIB information
    fClrUse,             // color use flag
    dwRop) ;            // raster operation
```

- 函式参数除了下列三个方面，均与 SetDIBitsToDevice 相同。
- 目的座标包括逻辑宽度(cxDst)和高度(cyDst)，以及开始点。
- 不能通过持续显示 DIB 来减少记忆体需求。

最後一个参数是位元映射操作方式，它指出了 DIB 图素与输出设备图素结合的方式，在最後一章将学到这些内容。现在我們为此参数设定为 SRCCOPY。

还有另一个更细微的差别。如果查看 SetDIBitsToDevice 的宣告，您会发现 cxSrc 和 cySrc 是 DWORD，这是 32 位元无正负号长整数型态。在 StretchDIBits 中，cxSrc 和 cySrc（以及 cxDst 和 cyDst）定义为带正负号的整数型态，这意味著它们可以为负数，实际上等一下就会看到，它们确实能为负数。如果您已经开始检查是否别的参数也可以为负数，就让我声明一下：在两个函式中，xSrc 和 ySrc 均定义为 int，但这是错的，这些值始终是非负数。

DIB 内的来源矩形被映射到目的矩形的座标显示如表 15-4 所示。

表 15-4

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cyDst - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxDst - 1, yDst + cyDst - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxDst - 1, yDst)

右列中的-1 项是不精确的，因为放大的程度（以及映射方式和其他变换）能产生略微不同的结果。

例如，考虑一个 2×2 的 DIB，这里 StretchDIBits 的 xSrc 和 ySrc 参数均为 0，cxSrc 和 cySrc 均为 2。假定我们显示到的装置内容具有 MM_TEXT 映射方式并且不进行变换。如果 xDst 和 yDst 均为 0，cxDst 和 cyDst 均为 4，那么我们将以倍数 2 放大 DIB。每个来源图素 (x, y) 将映射到下面所示的四个目的图素上：

```
(0,0) --> (0,2) and (1,2) and (0,3) and (1,3)
(1,0) --> (2,2) and (3,2) and (2,3) and (3,3)
(0,1) --> (0,0) and (1,0) and (0,1) and (1,1)
(1,1) --> (2,0) and (3,0) and (2,1) and (3,1)
```

上表正确地指出了目的的角，(0, 3)、(3, 3)、(0, 0)和(3, 0)。在其他情况下，座标可能是个大概值。

目的装置内容的映射方式对 SetDIBitsToDevice 的影响仅是由於 xDst 和 yDst 是逻辑座标。StretchDIBits 完全受映射方式的影响。例如，如果您设定了 y 值向上递增的一种度量映射方式，DIB 就会颠倒显示。

您可以通过把 cyDst 设定为负数来弥补这种情况。实际上，您可以将任何参数的宽度和高度变为负值来水平或垂直 • 转 DIB。在 MM_TEXT 映射方式下，如果 cySrc 和 cyDst 符号相反，DIB 会沿著水平轴 • 转并颠倒显示。如果 cxSrc 和 cxDst 符号相反，DIB 会沿著垂直轴 • 转并显示它的镜面图像。

下面是总结这些内容的运算式，xMM 和 yMM 指出映射方式的方向，如果 x 值向右增长，则 xMM 值为 1；如果 x 值向左增长，则值为-1。同样，如果 y 值向下增长，则 yMM 值为 1；如果 y 值向上增长，则值为-1。Sign 函式对於正值传回 TRUE，对於负值传回 FALSE。

```
if (!Sign (xMM × cxSrc × cxDst))
    DIB is flipped on its vertical axis (mirror image)
if (!Sign (yMM × cySrc × cyDst))
    DIB is flipped on its horizontal axis (upside down)
```

若有疑问，请查阅表 15-4。

程式 15-5 SHOWDIB 以实际尺寸显示 DIB、放大至显示区域视窗的大小、列印 DIB 以及把 DIB 传输到剪贴簿。

程式 15-5 SHOWDIB

```

SHOWDIB2.C
/*-----
    SHOWDIB2.C --      Shows a DIB in the client area
                        (c) Charles Petzold, 1998
    -----*/
/

#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS         wndclass ;
    wndclass.style   = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra   = 0 ;
    wndclass.cbWndExtra   = 0 ;
    wndclass.hInstance    = hInstance ;
    wndclass.hIcon        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName  = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Show DIB #2"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

```

```
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

hAccel = LoadAccelerators (hInstance, szAppName) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return msg.wParam ;
}

int ShowDib (HDC hdc, BITMAPINFO * pbmi, BYTE * pBits, int cxDib, int cyDib,
            int cxClient, int cyClient, WORD wShow)
{
    switch (wShow)
    {
    case IDM_SHOW_NORMAL:
        return SetDIBitsToDevice (hdc, 0, 0, cxDib, cyDib, 0, 0,
                                0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

    case IDM_SHOW_CENTER:
        return SetDIBitsToDevice (hdc, (cxClient - cxDib) / 2,
                                (cyClient - cyDib) / 2,
                                cxDib, cyDib, 0, 0, 0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

    case IDM_SHOW_STRETCH:
        SetStretchBltMode (hdc, COLORONCOLOR) ;
        return StretchDIBits(hdc, 0, 0, cxClient, cyClient, 0, 0, cxDib, cyDib,
                            pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;

    case IDM_SHOW_ISOSTRETCH:
        SetStretchBltMode (hdc, COLORONCOLOR) ;
        SetMapMode (hdc, MM_ISOTROPIC) ;
        SetWindowExtEx (hdc, cxDib, cyDib, NULL) ;
        SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
        SetWindowOrgEx (hdc, cxDib / 2, cyDib / 2, NULL) ;
        SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

        return StretchDIBits(hdc, 0, 0, cxDib, cyDib, 0, 0, cxDib, cyDib,
                            pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;
    }
}
```

```

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static BITMAPFILEHEADER    *    pbmfh ;
    static BITMAPINFO           *    pbmi ;
    static BYTE                 *    pBits ;
    static DOCINFO              di = {sizeof (DOCINFO),
                                     TEXT ("ShowDib2: Printing") } ;

    static      int             cxClient, cyClient, cxDib, cyDib ;
    static      PRINTDLG        printdlg = { sizeof (PRINTDLG) } ;
    static      TCHAR           szFileName [MAX_PATH], szTitleName
[MAX_PATH] ;
    static      WORD            wShow = IDM_SHOW_NORMAL ;
    BOOL         bSuccess ;
    HDC          hdc, hdcPrn ;
    HGLOBAL      hGlobal ;
    HMENU        hMenu ;
    int          cxPage, cyPage, iEnable ;
    PAINTSTRUCT  ps ;
    BYTE         * pGlobal ;

    switch (message)
    {
    case WM_CREATE:
        DibFileInitialize (hwnd) ;
        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_INITMENUPOPUP:
        hMenu = GetMenu (hwnd) ;

        if (pbmfh)
            iEnable = MF_ENABLED ;
        else
            iEnable = MF_GRAYED ;

        EnableMenuItem (hMenu, IDM_FILE_SAVE, iEnable) ;
        EnableMenuItem (hMenu, IDM_FILE_PRINT, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_CUT, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_COPY, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_DELETE, iEnable) ;
    }
}

```

```

        return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
        // Show the File Open dialog box

        if (!DibFileOpenDlg (hwnd, szFileName,
szTitleName))

            return 0 ;

        // If there's an existing DIB, free the memory
        if (pbmfh)
        {
            free (pbmfh) ;
            pbmfh = NULL ;
        }
        // Load the entire DIB into memory

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        pbmfh = DibLoadImage (szFileName) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        // Invalidate the client area for later update

        InvalidateRect (hwnd, NULL, TRUE) ;

        if (pbmfh == NULL)
        {
            MessageBox (hwnd, TEXT ("Cannot load DIB file"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;
        }
        // Get pointers to the info structure & the bits

        pbmi = (BITMAPINFO *) (pbmfh + 1) ;
        pBits = (BYTE *) pbmfh + pbmfh->bfOffBits ;

        // Get the DIB width and height

        if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))

```



```

        {
            cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
            cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
        }
        else
        {
            cxDib = pbmi->bmiHeader.biWidth ;
            cyDib = abs (pbmi->bmiHeader.biHeight) ;
        }
        return 0 ;

case IDM_FILE_SAVE:
    // Show the File Save dialog box

    if (!DibFileSaveDlg (hwnd, szFileName,
szTitleName))
        return 0 ;
    // Save the DIB to a disk file

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    bSuccess = DibSaveImage (szFileName, pbmfh) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (!bSuccess)
        MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
                    szAppName, MB_ICONEXCLAMATION |
MB_OK) ;

    return 0 ;

case IDM_FILE_PRINT:
    if (!pbmfh)
        return 0 ;

    // Get printer DC

    printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

    if (!PrintDlg (&printdlg))
        return 0 ;

    if (NULL == (hdcPrn = printdlg.hDC))
    {
        MessageBox ( hwnd, TEXT ("Cannot obtain Printer DC"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    }

```

```

        return 0 ;
    }

    // Check whether the printer can print bitmaps

    if  (!(RC_BITBLT & GetDeviceCaps (hdcPrn, RASTERCAPS)))
    {
        DeleteDC (hdcPrn) ;
        MessageBox (hwnd, TEXT ("Printer cannot print bitmaps"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;
    }

    // Get size of printable area of page

    cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
    cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

    bSuccess = FALSE ;
    // Send the DIB to the printer

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
    {
        ShowDib (    hdcPrn, pbmi, pBits, cxDib, cyDib,
            cxPage, cyPage, wShow) ;

        if (EndPage (hdcPrn) > 0)
        {
            bSuccess = TRUE ;
            EndDoc (hdcPrn) ;
        }
    }

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    DeleteDC (hdcPrn) ;

    if (!bSuccess)
        MessageBox (hwnd, TEXT ("Could not print bitmap"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

case  IDM_EDIT_COPY:
case  IDM_EDIT_CUT:
    if (!pbmfh)
        return 0 ;

```

```

// Make a copy of the packed DIB

hGlobal = GlobalAlloc (GHND | GMEM_SHARE, pbmfh->bfSize -
    sizeof (BITMAPFILEHEADER)) ;

pGlobal = GlobalLock (hGlobal) ;

CopyMemory ( pGlobal, (BYTE *) pbmfh + sizeof (BITMAPFILEHEADER),
    pbmfh->bfSize - sizeof (BITMAPFILEHEADER)) ;

GlobalUnlock (hGlobal) ;

// Transfer it to the clipboard

OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (CF_DIB, hGlobal) ;
CloseClipboard () ;

if (LOWORD (wParam) == IDM_EDIT_COPY)
    return 0 ;
// fall through if IDM_EDIT_CUT
case IDM_EDIT_DELETE:
    if (pbmfh)
    {
        free (pbmfh) ;
        pbmfh = NULL ;
        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

case IDM_SHOW_NORMAL:
case IDM_SHOW_CENTER:
case IDM_SHOW_STRETCH:
case IDM_SHOW_ISOSTRETCH:
    CheckMenuItem (hMenu, wShow, MF_UNCHECKED) ;
    wShow = LOWORD (wParam) ;
    CheckMenuItem (hMenu, wShow, MF_CHECKED) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
}
break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (pbmfh)

```

```

        ShowDib (   hdc, pbmi, pBits, cxDib, cyDib,
                    cxClient, cyClient, wShow) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        if (pbmfh)
            free (pbmfh) ;

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

SHOWDIB2.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB2 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open...\tCtrl+O",IDM_FILE_OPEN
        MENUITEM "&Save...\tCtrl+S",  IDM_FILE_SAVE
        MENUITEM SEPARATOR
        MENUITEM "&Print\tCtrl+P",    IDM_FILE_PRINT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM          "Cu&t\tCtrl+X", IDM_EDIT_CUT
        MENUITEM          "&Copy\tCtrl+C", IDM_EDIT_COPY
        MENUITEM          "&Delete\tDelete", IDM_EDIT_DELETE
    END
    POPUP "&Show"
    BEGIN
        MENUITEM          "&Actual Size",  IDM_SHOW_NORMAL, CHECKED
        MENUITEM          "&Center",        IDM_SHOW_CENTER
        MENUITEM          "&Stretch to Window", IDM_SHOW_STRETCH
        MENUITEM          "Stretch &Isotropically", IDM_SHOW_ISOSTRETCH
    END
END

////////////////////////////////////

```

```

/
// Accelerator
SHOWDIB2 ACCELERATORS DISCARDABLE
BEGIN
    "C",    IDM_EDIT_COPY,          VIRTKEY, CONTROL, NOINVERT
    "O",    IDM_FILE_OPEN,         VIRTKEY, CONTROL, NOINVERT
    "P",    IDM_FILE_PRINT,        VIRTKEY, CONTROL, NOINVERT
    "S",    IDM_FILE_SAVE,         VIRTKEY, CONTROL, NOINVERT
    VK_DELETE,    IDM_EDIT_DELETE,    VIRTKEY, NOINVERT
    "X",    IDM_EDIT_CUT,          VIRTKEY, CONTROL, NOINVERT
END

```

[RESOURCE.H \(摘录\)](#)

```

// Microsoft Developer Studio generated include file.
// Used by ShowDib2.rc

#define IDM_FILE_OPEN            40001
#define IDM_SHOW_NORMAL         40002
#define IDM_SHOW_CENTER         40003
#define IDM_SHOW_STRETCH        40004
#define IDM_SHOW_ISOSTRETCH     40005
#define IDM_FILE_PRINT          40006
#define IDM_EDIT_COPY           40007
#define IDM_EDIT_CUT            40008
#define IDM_EDIT_DELETE         40009
#define IDM_FILE_SAVE           40010

```

有意思的是 ShowDib 函式，它依赖于功能表选择以四种不同的方式之一在程式的显示区域显示 DIB。可以使用 SetDIBitsToDevice 从显示区域的左上角或在显示区域的中心显示 DIB。程式也有两个使用 StretchDIBits 的选项，DIB 能放大填充整个显示区域。在此情况下它可能会变形，或它能等比例显示，也就是说不会变形。

把 DIB 复制到剪贴簿包括：在整体共用记忆体中制作 packed DIB 记忆体块的副本。剪贴簿资料型态为 CF_DIB。程式没有列出从剪贴簿复制 DIB 的方法，因为在仅有指向 packed DIB 的指标的情况下这样做需要更多步骤来确定图素位元的偏移量。我将在下一章的末尾示范如何做到这点的办法。

您可能注意到了 SHOWDIB2 中的一些不足之处。如果您以 256 色显示模式执行 Windows，就会看到显示除了单色或 4 位元 DIB 以外的其他图形出现的问题，您看不到真正的颜色。存取那些颜色需要使用调色盘，在下一章会做这些工作。您也可能注意到速度问题，尤其在 Windows NT 下执行 SHOWDIB2 时。在下一章 packed DIB 和点阵图时，我会展示处理的方法。我也给 DIB 显示添加滚动列，这样也能以实际尺寸查看大於萤幕的 DIB。

色彩转换、调色盘和显示效能

记得在虎豹小霸王编剧 William Goldman 的另一出电影剧本《All the President's Men》中，Deep Throat 告诉 Bob Woodward 揭开水门秘密的关键是「跟著钱走」。那么在点阵图显示中获得高级性能的关键就是「跟著图素位元走」以及理解色彩转换发生的时机。DIB 是装置无关的格式，视讯显示器记忆体几乎总是与图素格式不同。在 SetDIBitsToDevice 或 StretchDIBits 函式呼叫期间，每个图素（可能有几百万个）必须从装置无关的格式转换成设备相关格式。

在许多情况下，这种转换是很繁琐的。例如，在 24 位元视讯显示器上显示 24 位元 DIB，显示驱动程式最多是切换红、绿、蓝的位元组顺序而已。在 24 位元设备上显示 16 位元 DIB 就需要位元的搬移和修剪了。在 24 位元设备上显示 4 位元或 8 位元 DIB 要求在 DIB 色彩对照表内查找 DIB 图素位元，然後对位元组重新排列。

但是要在 4 位元或 8 位元视讯显示器上显示 16 位元、24 位元或 32 位元 DIB 时，会发生什么事情呢？一种完全不一样的颜色转换发生了。对於 DIB 内的每个图素，装置驱动程式必须在图素和显示器上可用的颜色之间「找寻最接近的色彩」，这包括回圈和计算。（GDI 函式 GetNearestColor 进行「最接近色彩搜寻」。）

整个 RGB 色彩的三维阵列可用立方体表示。曲线内任意两点之间的距离是：

$$\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

在这里两个颜色是 R1G1B1 和 R2G2B2。执行最接近色彩搜寻包括从一种颜色到其他颜色集合中找寻最短距离。幸运的是，在 RGB 颜色立方体中「比较」距离时，并不需要计算平方根部分。但是需转换的每个图素必须与设备的所有颜色相比较以发现最接近的颜色。这是个工作量相当大的工作。（尽管在 8 位元设备上显示 8 位元 DIB 也得进行最接近色彩搜寻，但它不必对每个图素都进行，它仅需对 DIB 色彩对照表中的每种颜色进行寻找。）

正是由於以上原因，应该避免使用 SetDIBitsToDevice 或 StretchDIBits 在 8 位元视讯显示卡上显示 16 位元、24 位元或 32 位元 DIB。DIB 应转换为 8 位元 DIB，或者 8 位元 DDB，以求得更好的显示效能。实际上，您可以经由将 DIB 转换为 DDB 并使用 BitBlt 和 StretchBlt 显示图像，来加快显示任何 DIB 的速

度。

如果在 8 位元视讯显示器上执行 Windows (或仅仅切换到 8 位元模式来观察在显示 True-ColorDIB 时的效能变化)，您可能会注意到另一个问题：DIB 不会使用所有颜色来显示。任何在 8 位元视讯显示器上的 DIB 刚好限制在以 20 种颜色显示。如何获得多於 20 种颜色是「调色盘管理器」的任务，这将在下一章提到。

最後，如果在同一台机器上执行 Windows 98 和 Windows NT，您可能会注意到：对於同样的显示模式，Windows NT 显示大型 DIB 花费的时间较长。这是 Windows NT 的客户/伺服器体系结构的结果，它使大量资料在传输给 API 函式时耗费更多时间。解决方法是将 DIB 转换为 DDB。而我等一下将谈到的 CreateDIBSection 函式对这种情况特别有用。

DIB 和 DDB 的结合

您可以做许多事情去发掘 DIB 的格式，并呼叫两个 DIB 绘图函式：SetDIBitsToDevice 和 StretchDIBits。您可以直接存取 DIB 中的各个位元、位元组和图素，且一旦您有了一堆能让您以结构化的方式检查和更改资料的函式，您要怎么处理 DIB 就没人管了。

实际上，我们发现还是有一些限制。在上一章，我们了解了使用 GDI 函式在 DDB 上绘制图像的方法。到目前为止，还没有在 DIB 上绘图的方法。另一个问题是 SetDIBitsToDevice 和 StretchDIBits 没有 BitBlt 和 StretchBlt 速度快，尤其在 Windows NT 环境下以及执行许多最接近颜色搜寻（例如，在 8 位元视频卡上显示 24 位元 DIB）时。

因此，在 DIB 和 DDB 之间进行转换是有好处的。例如，如果我们有一个需要在萤幕上显示许多次的 DIB，那么把 DIB 转换为 DDB 就很有意义，这样我们就能够使用快速的 BitBlt 和 StretchBlt 函式来显示它了。

从 DIB 建立 DDB

从 DIB 中建立 GDI 点阵图物件可能吗？基本上我们已经知道了方法：如果有 DIB，您就能够使用 CreateCompatibleBitmap 来建立与 DIB 大小相同并与视讯显示器相容的 GDI 点阵图物件。然後将该点阵图物件选入记忆体装置内容并呼叫 SetDIBitsToDevice 在那个记忆体 DC 上绘图。结果就是 DDB 具有与 DIB 相同的图像，但具有与视讯显示器相容的颜色组织。

您也可以通过呼叫 CreateDIBitmap 用几个步骤完成上述工作。函式的语法为：

```

hBitmap = CreateDIBitmap (
    hdc,                      // device context handle
    pInfoHdr,                 // pointer to DIB
    information header
    fInit,                    // 0 or CBM_INIT
    pBits,                    // pointer to DIB pixel bits
    pInfo,                    // pointer to DIB information
    fClrUse) ;                // color use flag

```

请注意 pInfoHdr 和 pInfo 这两个参数，它们分别定义为指向 BITMAPINFOHEADER 结构和 BITMAPINFO 结构的指标。正如我们所知，BITMAPINFO 结构是後面紧跟色彩对照表的 BITMAPINFOHEADER 结构。我们一会儿会看到这种区别所起的作用。最後一个参数是 DIB_RGB_COLORS(等於 0)或 DIB_PAL_COLORS，它们在 SetDIBitsToDevice 函式中使用。下一章我将讨论更多这方面的内容。

理解 Windows 中点阵图函式的作用是很重要的。不要考虑 CreateDIBitmap 函式的名称，它不建立与「装置无关的点阵图」，它从装置无关的规格中建立「设备相关的点阵图」。注意该函式传回 GDI 点阵图物件的代号，CreateBitmap、CreateBitmapIndirect 和 CreateCompatibleBitmap 也与它一样。

呼叫 CreateDIBitmap 函式最简单的方法是：

```
hBitmap = CreateDIBitmap (NULL, pbmih, 0, NULL, NULL, 0) ;
```

唯一的参数是指向 BITMAPINFOHEADER 结构（不带色彩对照表）的指标。在这个形式中，函式建立单色 GDI 点阵图物件。第二种简单的方法是：

```
hBitmap = CreateDIBitmap (hdc, pbmih, 0, NULL, NULL, 0) ;
```

在这个形式中，函式建立了与装置内容相容的 DDB，该装置内容由 hdc 参数指出。到目前为止，我们都是透过 CreateBitmap（建立单色点阵图）或 CreateCompatibleBitmap（建立与视讯显示器相容的点阵图）来完成一些工作。

在 CreateDIBitmap 的这两个简化模式中，图素还未被初始化。如果 CreateDIBitmap 的第三个参数是 CBM_INIT，Windows 就会建立 DDB 并使用最後三个参数初始化点阵图位元。pInfo 参数是指向包括色彩对照表的 BITMAPINFO 结构的指标。pBits 参数是指向由 BITMAPINFO 结构指出的色彩格式中的位元阵列的指标，根据色彩对照表这些位元被转换为设备的颜色格式，这与 SetDIBitsToDevice 的情况相同。实际上，整个 CreateDIBitmap 函式可以用下列程式码来实作：

```

HBITMAP CreateDIBitmap (    HDC hdc, CONST BITMAPINFOHEADER * pbmih,
    DWORD fInit, CONST VOID * pBits,
    CONST BITMAPINFO * pbmi, UINT fUsage)
{
    HBITMAP    hBitmap ;
    HDC        hdc ;

```



```

int          cx, cy, iBitCount ;

if (pbmih->biSize == sizeof (BITMAPCOREHEADER))
{
    cx  = ((PBITMAPCOREHEADER) pbmih)->bcWidth ;
    cy  = ((PBITMAPCOREHEADER) pbmih)->bcHeight ;
    iBitCount  = ((PBITMAPCOREHEADER) pbmih)->bcBitCount ;
}
else
{
    cx = pbmih->biWidth ;
    cy = pbmih->biHeight ;
    iBitCount  = pbmih->biBitCount ;
}
if (hdc)
    hBitmap = CreateCompatibleBitmap (hdc, cx, cy) ;
else
    hBitmap = CreateBitmap (cx, cy, 1, 1, NULL) ;
if (fInit == CBM_INIT)
{
    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;
    SetDIBitsToDevice (    hdcMem, 0, 0, cx, cy, 0, 0, 0 cy,
pBits, pbmi, fUsage) ;
    DeleteDC (hdcMem) ;
}

return hBitmap ;
}

```

如果仅需显示 DIB 一次，并担心 SetDIBitsToDevice 显示太慢，则呼叫 CreateDIBitmap 并使用 BitBlt 或 StretchBlt 来显示 DDB 就没有什么意义。因为 SetDIBitsToDevice 和 CreateDIBitmap 都执行颜色转换，这两个工作会占用同样长的时间。只有在多次显示 DIB 时（例如在处理 WM_PAINT 讯息时）进行这种转换才有意义。

程式 15-6 DIBCONV 展示了利用 SetDIBitsToDevice 把 DIB 档案转换为 DDB 的方法。

程式 15-6 DIBCONV

```

DIBCONV.C
/*-----
    DIBCONV.C --      Converts a DIB to a DDB
                      (c) Charles Petzold, 1998
-----*/

#include <windows.h>

```

```
#include <commdlg.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("DibConv") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = szAppName ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("DIB to DDB Conversion"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}
```

```

}
HBITMAP CreateBitmapObjectFromDibFile (HDC hdc, PTSTR szFileName)
{
    BITMAPFILEHEADER *    pbmfh ;
    BOOL                  bSuccess ;
    DWORD                 dwFileSize, dwHighSize, dwBytesRead ;
    HANDLE                 hFile ;
    HBITMAP                hBitmap ;

    // Open the file: read access, prohibit write access

    hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
    if (hFile == INVALID_HANDLE_VALUE)
        return NULL ;

    // Read in the whole file

    dwFileSize = GetFileSize (hFile, &dwHighSize) ;

    if (dwHighSize)
    {
        CloseHandle (hFile) ;
        return NULL ;
    }

    pbmfh = malloc (dwFileSize) ;

    if (!pbmfh)
    {
        CloseHandle (hFile) ;
        return NULL ;
    }

    bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
    CloseHandle (hFile) ;

    // Verify the file
    if (!bSuccess || (dwBytesRead != dwFileSize)
        || (pbmfh->bfType != * (WORD *) "BM")
        || (pbmfh->bfSize != dwFileSize))
    {
        free (pbmfh) ;
        return NULL ;
    }

    // Create the DDB
    hBitmap = CreateDIBitmap (hdc,
        (BITMAPINFOHEADER *) (pbmfh + 1),

```

```

        CBM_INIT,
        (BYTE *) pbmfh + pbmfh->bfOffBits,
        (BITMAPINFO *) (pbmfh + 1),
        DIB_RGB_COLORS) ;

    free (pbmfh) ;
    return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP      hBitmap ;
    static int          cxClient, cyClient ;
    static OPENFILENAME  ofn ;
    static TCHAR  szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
    static TCHAR  szFilter[]=TEXT("Bitmap Files (*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*\.*\0\0") ;
    BITMAP        bitmap ;
    HDC            hdc, hdcMem ;
    PAINTSTRUCT    ps ;

    switch (message)
    {
    case  WM_CREATE:
        ofn.lStructSize      = sizeof (OPENFILENAME) ;
        ofn.hwndOwner        = hwnd ;
        ofn.hInstance        = NULL ;
        ofn.lpstrFilter      = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter   = 0 ;
        ofn.nFilterIndex     = 0 ;
        ofn.lpstrFile        = szFileName ;
        ofn.nMaxFile         = MAX_PATH ;
        ofn.lpstrFileTitle   = szTitleName ;
        ofn.nMaxFileTitle    = MAX_PATH ;
        ofn.lpstrInitialDir   = NULL ;
        ofn.lpstrTitle       = NULL ;
        ofn.Flags            = 0 ;
        ofn.nFileOffset      = 0 ;
        ofn.nFileExtension   = 0 ;
        ofn.lpstrDefExt       = TEXT ("bmp") ;
        ofn.lCustData         = 0 ;
        ofn.lpfnHook         = NULL ;
        ofn.lpTemplateName   = NULL ;

        return 0 ;

    case  WM_SIZE:
        cxClient = LOWORD (lParam) ;

```

```

        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

        // Show the File Open dialog box

        if (!GetOpenFileName (&ofn))
            return 0 ;

        // If there's an existing DIB, delete it

        if (hBitmap)
        {
            DeleteObject (hBitmap) ;
            hBitmap = NULL ;
        }

        // Create the DDB from the DIB

        SetCursor      (LoadCursor      (NULL,
IDC_WAIT)) ;

        ShowCursor (TRUE) ;

        hdc = GetDC (hwnd) ;
        hBitmap = CreateBitmapObjectFromDibFile (hdc,
szFileName) ;

        ReleaseDC (hwnd, hdc) ;

        ShowCursor (FALSE) ;
        SetCursor      (LoadCursor      (NULL,
IDC_ARROW)) ;

        // Invalidate the client area for later update

        InvalidateRect (hwnd, NULL, TRUE) ;
        if (hBitmap == NULL)
        {
            MessageBox (hwnd, TEXT ("Cannot load DIB file"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
        }
        return 0 ;
    }
    break ;

case WM_PAINT:

```

```

        hdc = BeginPaint (hwnd, &ps) ;

        if (hBitmap)
        {
            GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

            hdcMem = CreateCompatibleDC (hdc) ;
            SelectObject (hdcMem, hBitmap) ;

            BitBlt (hdc, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
                    hdcMem, 0, 0, SRCCOPY) ;

                                DeleteDC (hdcMem) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        if (hBitmap)
            DeleteObject (hBitmap) ;

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

DIBCONV.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////

/

// Menu

DIBCONV MENU DISCARDABLE

BEGIN

POPUP "&File"

BEGIN

MENUITEM "&Open", IDM_FILE_OPEN

END

END

RESOURCE.H (摘录)

// Microsoft Developer Studio generated include file.

// Used by DibConv.rc

#define IDM_FILE_OPEN 40001

DIBCONV.C 本身就是完整的，并不需要前面的档案。在对它仅有的功能表命

令 (「 File Open 」) 的回应中, WndProc 呼叫程式的 CreateBitmapObjectFromDibFile 函式。此函式将整个档案读入记忆体并将指向记忆体块的指标传递给 CreateDIBitmap 函式, 函式传回点阵图的代号, 然後包含 DIB 的记忆体块被释放。在 WM_PAINT 讯息处理期间, WndProc 将点阵图选入相容的记忆体装置内容并使用 BitBlt (不是 SetDIBitsToDevice) 在显示区域显示点阵图。它通过使用点阵图代号呼叫带有 BITMAP 结构的 GetObject 函式来取得点阵图的宽度和高度。

在从 CreateDIBitmap 建立点阵图时不必初始化 DDB 图素位元, 之後您可以呼叫 SetDIBits 初始化图素位元。该函式的语法如下:

```
iLines = SetDIBits (
    hdc,          // device context handle
    hBitmap,      // bitmap handle
    yScan,        // first scan line to convert
    cyScans,      // number of scan lines to convert
    pBits,        // pointer to pixel bits
    pInfo,        // pointer to DIB information
    fClrUse) ;    // color use flag
```

函式使用了 BITMAPINFO 结构中的色彩对照表把位元转换为设备相关的格式。只有在最後一个参数设定为 DIB_PAL_COLORS 时, 才需要装置内容代号。

从 DDB 到 DIB

与 SetDIBits 函式相似的函式是 GetDIBits, 您可以使用此函式把 DDB 转换为 DIB:

```
int WINAPI GetDIBits (
    hdc,          // device context handle
    hBitmap,      // bitmap handle
    yScan,        // first scan line to convert
    cyScans,      // number of scan lines to convert
    pBits,        // pointer to pixel bits (out)
    pInfo,        // pointer to DIB information (out)
    fClrUse) ;    // color use flag
```

然而, 此函式产生的恐怕不是 SetDIBits 的反运算结果。在一般情况下, 如果使用 CreateDIBitmap 和 SetDIBits 将 DIB 转换为 DDB, 然後使用 GetDIBits 把 DDB 转换回 DIB, 您就不会得到原来的图像。这是因为在 DIB 被转换为设备相关的格式时, 有一些资讯遗失了。遗失的资讯数量取决於进行转换时 Windows 所执行的显示模式。

您可能会发现没有使用 GetDIBits 的必要性。考虑一下: 在什么环境下您的程式发现自身带有点阵图代号, 但没有用於在起始的位置建立点阵图的资料? 剪贴簿? 但是剪贴簿为 DIB 提供了自动的转换。GetDIBits 函式的一个例子

是在捕捉萤幕显示内容的情况下，例如第十四章中 BLOWUP 程式所做的。我不示范这个函式，但在 Microsoft 网站的 Knowledge Base 文章 Q80080 中有一些资讯。

DIB 区块

我希望您已经对设备相关和装置无关点阵图的区别有了清晰的概念。DIB 能拥有几种色彩组织中的一种，DDB 必须是单色的或是与真实输出设备相同的格式。DIB 是一个档案或记忆体块；DDB 是 GDI 点阵图物件并由点阵图代号表示。DIB 能被显示或转换为 DDB 并转换回 DIB，但是这里包含了装置无关位元和设备相关位元之间的转换程序。

现在您将遇到一个函式，它打破了这些规则。该函式在 32 位元 Windows 版本中发表，称为 CreateDIBSection，语法为：

```
hBitmap = CreateDIBSection (
    hdc,          // device context handle
    pInfo,        // pointer to DIB information
    fClrUse,       // color use flag
    ppBits,       // pointer to pointer variable
    hSection,      // file-mapping object handle
    dwOffset) ;   // offset to bits in file-mapping object
```

CreateDIBSection 是 Windows API 中最重要的函式之一（至少在使用点阵图时），然而您会发现它很深奥并难以理解。

让我们从它的名称开始，我们知道 DIB 是什么，但「DIB section」到底是什么呢？当您第一次检查 CreateDIBSection 时，可能会寻找该函式与 DIB 区块工作的方式。这是正确的，CreateDIBSection 所做的就是建立了 DIB 的一部分（点阵图图素位元的记忆体块）。

现在我们看一下传回值，它是 GDI 点阵图物件的代号，这个传回值可能是该函式呼叫最会拐人的部分。传回值似乎暗示著 CreateDIBSection 在功能上与 CreateDIBitmap 相同。事实上，它只是相似但完全不同。实际上，从 CreateDIBSection 传回的点阵图代号与我们在本章和上一章遇到的所有点阵图建立函式传回的点阵图代号在本质上不同。

一旦理解了 CreateDIBSection 的真实特性，您可能觉得奇怪为什么不把传回值定义得有所区别。您也可能得出结论：CreateDIBSection 应该称之为 CreateDIBitmap，并且如同我前面所指出的 CreateDIBitmap 应该称之为 CreateDDBitmap。

首先让我们检查一下如何简化 CreateDIBSection，并正确地使用它。首先，把最後两个参数 hSection 和 dwOffset，分别设定为 NULL 和 0，我将在本章最

後讨论这些参数的用法。第二，仅在 fColorUse 参数设定为 DIB_ PAL_COLORS 时，才使用 hdc 参数，如果 fColorUse 为 DIB_RGB_COLORS (或 0)，hdc 将被忽略 (这与 CreateDIBitmap 不同，hdc 参数用於取得与 DDB 相容的设备的色彩格式)。

因此，CreateDIBSection 最简单的形式仅需要第二和第四个参数。第二个参数是指向 BITMAPINFO 结构的指标，我们以前曾使用过。我希望指向第四个参数的指标定义的指标不会使您困惑，它实际上很简单。

假设要建立每图素 24 位元的 384×256 位元 DIB，24 位元格式不需要色彩对照表，因此它是最简单的，所以我们可以为 BITMAPINFO 参数使用 BITMAPINFOHEADER 结构。

您需要定义三个变数：BITMAPINFOHEADER 结构、BYTE 指标和点阵图代号：

```
BITMAPINFOHEADER bmih ;
BYTE              * pBits ;
HBITMAP           hBitmap ;
```

现在初始化 BITMAPINFOHEADER 结构的栏位：

```
bmih->biSize      = sizeof (BITMAPINFOHEADER) ;
bmih->biWidth      = 384 ;
bmih->biHeight     = 256 ;
bmih->biPlanes     = 1 ;
bmih->biBitCount   = 24 ;
bmih->biCompression = BI_RGB ;
bmih->biSizeImage  = 0 ;
bmih->biXPelsPerMeter = 0 ;
bmih->biYPelsPerMeter = 0 ;
bmih->biClrUsed    = 0 ;
bmih->biClrImportant = 0 ;
```

在基本准备後，我们呼叫该函式：

```
hBitmap = CreateDIBSection (NULL, (BITMAPINFO *) &bmih, 0, &pBits, NULL, 0) ;
```

注意，我们为第二个参数赋予 BITMAPINFOHEADER 结构的位址。这是常见的，但一个 BYTE 指标 pBits 的位址，就不常见了。这样，第四个参数是函式需要的指向指标的指标。

这是函式呼叫所做的：CreateDIBSection 检查 BITMAPINFOHEADER 结构并配置足够的记忆体块来载入 DIB 图素位元。(在这个例子里，记忆体块的大小为 384×256×3 位元组。)它在您提供的 pBits 参数中储存了指向此记忆体块的指标。函式传回点阵图代号，正如我说的，它与 CreateDIBitmap 和其他点阵图建立函式传回的代号不一样。

然而，我们还没有做完，点阵图图素是未初始化的。如果正在读取 DIB 档案，可以简单地把 pBits 参数传递给 ReadFile 函式并读取它们。或者可以使用

一些程式码「人工」设定。

程式 15-7 DIBSECT 除了呼叫 CreateDIBSection 而不是 CreateDIBitmap 之外, 与 DIBCONV 程式相似。

程式 15-7 DIBSECT

```
DIBSECT.C
/*-----
--
--      DIBSECT.C --      Displays a DIB Section in the client area
--                        (c) Charles Petzold, 1998
--
--*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName[] = TEXT ("DIBsect") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = szAppName ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("DIB Section Display"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
```

```

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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}
HBITMAP CreateDIBSectionFromDibFile (PTSTR szFileName)
{
    BITMAPFILEHEADER    bmfh ;
    BITMAPINFO           *    pbmi ;
    BYTE                *    pBits ;
    BOOL                bSuccess ;
    DWORD                dwInfoSize, dwBytesRead ;
    HANDLE               hFile ;
    HBITMAP              hBitmap ;

    // Open the file: read access, prohibit write access

    hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ,
        NULL, OPEN_EXISTING, 0, NULL) ;
    if (hFile == INVALID_HANDLE_VALUE)
        return NULL ;
    // Read in the BITMAPFILEHEADER
    bSuccess = ReadFile (hFile, &bmfh, sizeof (BITMAPFILEHEADER),
        &dwBytesRead, NULL) ;

    if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
        || (bmfh.bfType != * (WORD *) "BM"))
    {
        CloseHandle (hFile) ;
        return NULL ;
    }

    // Allocate memory for the BITMAPINFO structure & read
    it in
    dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
    pbmi = malloc (dwInfoSize) ;
    bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;
    if (!bSuccess || (dwBytesRead != dwInfoSize))
    {

```

```

        free (pbmi) ;
        CloseHandle (hFile) ;
        return NULL ;
    }

    // Create the DIB Section
    hBitmap = CreateDIBSection (NULL, pbmi, DIB_RGB_COLORS, &pBits, NULL, 0) ;
    if (hBitmap == NULL)
    {
        free (pbmi) ;
        CloseHandle (hFile) ;
        return NULL ;
    }

    // Read in the bitmap bits
    ReadFile (hFile, pBits, bmfh.bfSize - bmfh.bfOffBits, &dwBytesRead, NULL) ;
    free (pbmi) ;
    CloseHandle (hFile) ;

    return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP          hBitmap ;
    static int              cxClient, cyClient ;
    static OPENFILENAME ofn ;
    static TCHAR            szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
    static TCHAR            szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*.*\0\0") ;
    BITMAP                  bitmap ;
    HDC                     hdc, hdcMem ;
    PAINTSTRUCT             ps ;

    switch (message)
    {
    case WM_CREATE:
        ofn.lStructSize          = sizeof (OPENFILENAME) ;
        ofn.hwndOwner            = hwnd ;
        ofn.hInstance            = NULL ;
        ofn.lpstrFilter           = szFilter ;
        ofn.lpstrCustomFilter     = NULL ;
        ofn.nMaxCustFilter       = 0 ;
        ofn.nFilterIndex         = 0 ;
        ofn.lpstrFile             = szFileName ;
        ofn.nMaxFile              = MAX_PATH ;
        ofn.lpstrFileTitle        = szTitleName ;
        ofn.nMaxFileTitle         = MAX_PATH ;
        ofn.lpstrInitialDir       = NULL ;

```

```

        ofn.lpstrTitle           = NULL ;
        ofn.Flags                = 0 ;
        ofn.nFileOffset         = 0 ;
        ofn.nFileExtension      = 0 ;
        ofn.lpstrDefExt          = TEXT ("bmp") ;
        ofn.lCustData            = 0 ;
        ofn.lpfnHook             = NULL ;
        ofn.lpTemplateName      = NULL ;

        return 0 ;

case WM_SIZE:

    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_COMMAND:

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

        // Show the File Open dialog box

                                if (!GetOpenFileName (&ofn))
                                    return 0 ;

        // If there's an existing bitmap, delete it

                                if (hBitmap)
                                {
                                    DeleteObject (hBitmap) ;
                                    hBitmap = NULL ;
                                }

        // Create the DIB Section from the DIB file

                                SetCursor (LoadCursor (NULL,
IDC_WAIT)) ;

                                ShowCursor (TRUE) ;

                                hBitmap = CreateDIBSectionFromDibFile
(szFileName) ;

                                ShowCursor (FALSE) ;
                                SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        // Invalidate the client area for later update

                                InvalidateRect (hwnd, NULL, TRUE) ;

```

```

        if (hBitmap == NULL)
        {
            MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
        }
        return 0 ;
    }
    break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hBitmap)
    {
        GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;

        BitBlt (    hdc, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
            hdcMem, 0, 0, SRCCOPY) ;

        DeleteDC (hdcMem) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    if (hBitmap)
        DeleteObject (hBitmap) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

DIBSECT.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

DIBSECT MENU DISCARDABLE

BEGIN

POPUP "&File"

```

        BEGIN
            MENUITEM "&Open",          IDM_FILE_OPEN
        END
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by DIBsect.rc

#define IDM_FILE_OPEN          40001

```

注意 DIBCONV 中的 CreateBitmapObjectFromDibFile 函式和 DIBSECT 中的 CreateDibsectionFromDibFile 函式之间的区别。DIBCONV 读入整个档案，然後把指向 DIB 记忆体块的指标传递给 CreateDIBitmap 函式。DIBSECT 首先读取 BITMAPFILEHEADER 结构中的资讯，然後确定 BITMAPINFO 结构的大小，为此配置记忆体，并在第二个 ReadFile 呼叫中将它读入记忆体。然後，函式把指向 BITMAPINFO 结构和指标变数 pBits 的指标传递给 CreateDIBSection。函式传回点阵图代号并设定 pBits 指向函式将要读取 DIB 图素位元的记忆体块。

pBits 指向的记忆体块归系统所有。当通过呼叫 DeleteObject 删除点阵图时，记忆体会被自动释放。然而，程式能利用该指标直接改变 DIB 位元。当应用程式透过 API 传递大量记忆体块时，只要系统拥有这些记忆体块，在 WINDOWS NT 下就不会影响速度。

我之前曾说过，当在视讯显示器上显示 DIB 时，某些时候必须进行从装置无关图素到设备相关图素的转换，有时这些格式转换可能相当费时。来看看三种用於显示 DIB 的方法：

- 当使用 SetDIBitsToDevice 或 StretchDIBits 来把 DIB 直接显示在萤幕上，格式转换在 SetDIBitsToDevice 或 StretchDIBits 呼叫期间发生。
- 当使用 CreateDIBitmap 和（可能是）SetDIBits 把 DIB 转换为 DDB，然後使用 BitBlt 或 StretchBlt 来显示它时，如果设定了 CBM_INIT 旗标，格式转换在 CreateDIBitmap 或 SetDIBits 期间发生。
- 当使用 CreateDIBSection 建立 DIB 区块，然後使用 BitBlt 或 StretchBlt 显示它时，格式转换在 BitBlt 对 StretchBlt 的呼叫期间发生。

再读一下上面这些叙述，确定您不会误解它的意思。这是从 CreateDIBSection 传回的点阵图代号不同於我们所遇到的其他点阵图代号的一个地方。此点阵图代号实际上指向储存在记忆体中由系统维护但应用程式能存取的 DIB。在需要的时候，DIB 会转化为特定的色彩格式，通常是在用 BitBlt 或 StretchBlt 显示点阵图时。

您也可以将点阵图代号选入记忆体装置内容并使用 GDI 函式来绘制。在 pBits 变数指向的 DIB 图素内将反映出结果。因为 Windows NT 下的 GDI 函式分

批呼叫，在记忆体设备背景上绘制之後和「人为」的存取位元之前会呼叫 GdiFlush。

在 DIBSECT，我们清除 pBits 变数，因为程式不再需要这个变数了。您会使用 CreateDIBSection 的主要原因在於您有需要直接更改位元值。在 CreateDIBSection 呼叫之後似乎就没有别的方法来取得位元指标了。

DIB 区块的其他区别

从 CreateDIBitmap 传回的点阵图代号与函式的 hdc 参数引用的设备有相同的平面和图素位元组织。您能通过具有 BITMAP 结构的 GetObject 呼叫来检验这一点。

CreateDIBSection 就不同了。如果以该函式传回的点阵图代号的 BITMAP 结构呼叫 GetObject，您会发现点阵图具有的色彩组织与 BITMAPINFOHEADER 结构的栏位指出的色彩组织相同。您能将这个代号选入与视讯显示器相容的记忆体装置内容。这与上一章关于 DDB 的内容相矛盾，但这也就是我说此 DIB 区块点阵图代号不同的原因。

另一个奇妙之处是：您可能还记得，DIB 中图素资料行的位元组长度始终是 4 的倍数。GDI 点阵图物件中行的位元组长度，就是使用 GetObject 从 BITMAP 结构的 bmWidthBytes 栏位中得到的长度，始终是 2 的倍数。如果用每图素 24 位元和宽度 2 图素设定 BITMAPINFOHEADER 结构并随后呼叫 GetObject，您就会发现 bmWidthBytes 栏位是 8 而不是 6。

使用从 CreateDIBSection 传回的点阵图代号，也可以使用 DIBSECTION 结构呼叫 GetObject：

```
GetObject (hBitmap, sizeof (DIBSECTION), &dibsection) ;
```

此函式不能处理其他点阵图建立函式传回的点阵图代号。DIBSECTION 结构定义如下：

```
typedef struct tagDIBSECTION // ds
{
    BITMAP                dsBm ;                // BITMAP structure
    BITMAPINFOHEADER dsBmih ;                // DIB information header
    DWORD                dsBitFields [3] ; // color masks
    HANDLE                dshSection ;                // file-mapping object
handle
    DWORD                dsOffset ;                // offset to bitmap bits
}
DIBSECTION, * PDIBSECTION ;
```

此结构包含 BITMAP 结构和 BITMAPINFOHEADER 结构。最後两个栏位是传递给 CreateDIBSection 的最後两个参数，等一下将会讨论它们。

DIBSECTION 结构中包含除了色彩对照表以外有关点阵图的许多内容。当把 DIB 区块点阵图代号选入记忆体装置内容时，可以通过呼叫 GetDIBColorTable 来得到色彩对照表：

```
hdcMem = CreateCompatibleDC (NULL) ;
SelectObject (hdcMem, hBitmap) ;
GetDIBColorTable (hdcMem, uFirstIndex, uNumEntries, &rgb) ;
DeleteDC (hdcMem) ;
```

同样，您可以通过呼叫 SetDIBColorTable 来设定色彩对照表中的项目。

档案映射选项

我们还没有讨论 CreateDIBSection 的最后两个参数，它们是档案映射物件的代号和档案中点阵图位元开始的偏移量。档案映射物件使您能够像档案位于记忆体中一样处理档案。也就是说，可以通过使用记忆体指标来存取档案，但档案不需要整个载入记忆体中。

在大型 DIB 的情况下，此技术对于减少记忆体需求是很有帮助的。DIB 图素位元能够储存在磁片上，但仍然可以当作位于记忆体中一样进行存取，虽然会影响程式执行效能。问题是，当图素位元实际上储存在磁片上时，它们不可能是实际 DIB 档案的一部分。它们必须位于其他的档案内。

为了展示这个程序，下面显示的函式除了不把图素位元读入记忆体以外，与 DIBSECT 中建立 DIB 区块的函式很相似。然而，它提供了档案映射物件和传递给 CreateDIBSection 函式的偏移量：

```
HBITMAP CreateDIBSectionMappingFromFile (PTSTR szFileName)
{
    BITMAPFILEHEADER    bmfh ;
    BITMAPINFO           *    pbmi ;
    BYTE                *    pBits ;
    BOOL                bSuccess ;
    DWORD               dwInfoSize, dwBytesRead ;
    HANDLE              hFile, hFileMap ;
    HBITMAP              hBitmap ;

    hFile = CreateFile (szFileName, GENERIC_READ | GENERIC_WRITE,
                        0, // No sharing!
                        NULL, OPEN_EXISTING, 0, NULL) ;

    if (hFile == INVALID_HANDLE_VALUE)
        return NULL ;
    bSuccess = ReadFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
                        &dwBytesRead, NULL) ;

    if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER)))
```

```
        || (bmfh.bfType != * (WORD *) "BM"))
    {
        CloseHandle (hFile) ;
        return NULL ;
    }
    dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
    pbmi = malloc (dwInfoSize) ;
    bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;

    if (!bSuccess || (dwBytesRead != dwInfoSize))
    {
        free (pbmi) ;
        CloseHandle (hFile) ;
        return NULL ;
    }
    hFileMap = CreateFileMapping (hFile, NULL, PAGE_READWRITE, 0, 0, NULL) ;
    hBitmap = CreateDIBSection ( NULL, pbmi, DIB_RGB_COLORS, &pBits, hFileMap,
bmfh.bfOffBits) ;
    free (pbmi) ;
    return hBitmap ;
}
```

啊哈！这个程式不会动。CreateDIBSection 的文件指出「dwOffset [函式的最後一个参数]必须是 DWORD 大小的倍数」。尽管资讯表头的大小始终是 4 的倍数并且色彩对照表的大小也始终是 4 的倍数，但点阵图档案表头却不是，它是 14 位元组。因此 bmfh.bfOffBits 永远不会是 4 的倍数。

总结

如果您有小型的 DIB 并且需要频繁地操作图素位元，您可以使用 SetDIBitsToDevice 和 StretchDIBits 来显示它们。然而，对于大型的 DIB，此技术会遇到显示效能的问题，尤其在 8 位元视讯显示器上和 Windows NT 环境下。

您可以使用 CreateDIBitmap 和 SetDIBits 把 DIB 转化为 DDB。现在，显示点阵图可以使用快速的 BitBlt 和 StretchBlt 函式来进行了。然而，您不能直接存取这些与装置无关的图素位元。

CreateDIBSection 是一个很好的折衷方案。在 Windows NT 下通过 BitBlt 和 StretchBlt 使用点阵图代号比使用 SetDIBitsToDevice 和 StretchDIBits(但没有 DDB 的缺陷)会得到更好的效能。您仍然可以存取 DIB 图素位元。

下一章，在讨论「Windows 调色盘管理器」之后会进入点阵图的探索。

第十六章 调色盘管理器

如果硬件允许，本章就没有存在的必要。尽管许多现代的显示卡提供 24 位元颜色（也称「true color」或「数百万色」）或 16 位元颜色（「增强色」或「数万种颜色」），一些显示卡——尤其是在携带型电脑上或高解析度模式中——每个图素只允许 8 位元。这意味著仅有 256 种颜色。

我们用 256 种颜色能做什么呢？很明显，要显示真实世界的图像，仅 16 种颜色是不够的，至少要使用数千或数百万种颜色，256 种颜色位於中间状态。是的，用 256 种颜色来显示真实世界的图像足够了，但需要根据特定的图像来指定这些颜色。这意味著作业系统不能简单地选择「标准」系列的 256 种颜色，就希望它们对每个应用程式都是理想的颜色。

这就是 Windows 调色盘管理器所要涉及的全部内容。它用於指定程式在 8 位元显示模式下执行时所需要的颜色。如果知道程式肯定不会在 8 位元显示模式下执行，那么您也不需要使用调色盘管理器。不过，由於补充了点阵图的一些细节，所以本章还是包含重要资讯的。

使用调色盘

传统上讲，调色盘是画家用来混合颜色的板子。这个词也可以指画家在绘画过程中使用的所有颜色。在电脑图形中，调色盘是在图形输出设备（例如视讯显示器）上可用的颜色范围。这个名词也可以指支援 256 色模式的显示卡上的对照表。

视频硬件

显示卡上的调色盘对照表运作过程如下图所示：



在 8 位元显示模式中，每个图素占 8 位元。图素值查询包含 256RGB 值的对照表的位址。这些 RGB 值可以正好 24 位元宽，或者小一点，通常是 18 位元宽（即主要的红、绿和蓝各 6 位元）。每种颜色的值都输入到数位类比转换器，以得到发送给监视器的红、绿和蓝三个类比信号。

通常，软体可以用任意值来载入调色盘对照表，但这对装置无关的视窗介面，例如 Microsoft Windows，会有一些干扰。首先，Windows 必须提供软体介面，以便在不直接干扰硬体的情况下，應用程式就可以存取调色盘管理器。第二个问题更严重：因为所有的應用程式都共用同一个视讯显示器，而且同时执行，所以一个應用程式使用了调色盘对照表可能会影响其他程式的使用。

这时就需要使用 Windows 调色盘管理器(在 Windows 3.0 中提出)了。Windows 保留了 256 种颜色中的 20 种，而允许應用程式修改其余的 236 种。（在某些情况下，應用程式最多可以改变 256 种颜色中的 254 种——只有黑色和白色除外——但这有一点麻烦）。Windows 为系统保留的 20 种颜色（有时称为 20 种「静态」颜色）如表 16-1 所示。

表 16-1 256 种颜色显示模式中的 20 种保留的颜色

图素位元	RGB 值	颜色名称	图素位元	RGB 值	颜色名称
00000000	00 00 00	黑	11111111	FF FF FF	白
00000001	80 00 00	暗红	11111110	00 FF FF	青
00000010	00 80 00	暗绿	11111101	FF 00 FF	洋红
00000011	80 80 00	暗黄	11111100	00 00 FF	蓝
00000100	00 00 80	暗蓝	11111011	FF FF 00	黄
00000101	80 00 80	暗洋红	11111010	00 FF 00	绿
00000110	00 80 80	暗青	11111001	FF 00 00	红
00000111	C0 C0 C0	亮灰	11111000	80 80 80	暗灰
00001000	C0 DC C0	美元绿	11110111	A0 A0 A4	中性灰
00001001	A6 CA F0	天蓝	11110110	FF FB F0	乳白色

在 256 种颜色显示模式下执行时，由 Windows 维护系统调色盘，此调色盘与显示卡上的硬体调色盘对照表相同。内定的系统调色盘如表 16-1 所示。應用程式可以通过指定「逻辑调色盘 (logical palettes)」来修改其余 236 种颜色。如果有多个應用程式使用逻辑调色盘，那么 Windows 就给活动视窗最高优先权（我们知道，活动视窗有高亮显示标题列，并且显示在其他所有视窗的前面）。我们将用一些简单的范例程式来检查它是如何工作的。

要执行本章其他部分的程式，您可能需要将显示卡切换到 256 色模式。在桌面上单擎滑鼠右键，从功能表中选择「属性」，然後选择「设定」页面标签。

显示灰阶

程式 16-1 所示的 GRAYS1 程式没有使用 Windows 调色盘管理器，而尝试用正常显示的 65 级种阶作为从黑到白的多种彩色的「来源」。

程式 16-1 GRAYS1

```

GRAYS1.C
/*-----
-
    GRAYS1.C --      Gray Shades
                                (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Grays1") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon                (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Shades of Gray #1"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

```

```

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static int          cxClient, cyClient ;
    HBRUSH              hBrush ;
    HDC                 hdc ;
    int                 i ;
    PAINTSTRUCT          ps ;
    RECT                rect ;

    switch (message)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

                                // Draw the fountain of grays

        for (i = 0 ; i < 65 ; i++)
        {
            rect.left           = i * cxClient
/ 65 ;

            rect.top            = 0 ;
            rect.right          = (i + 1) *
cxClient / 65 ;

            rect.bottom         = cyClient ;

            hBrush = CreateSolidBrush (RGB(min (255, 4 * i),
                                min (255, 4 * i)),
                                min (255, 4 * i)) ;
            FillRect (hdc, &rect, hBrush) ;
            DeleteObject (hBrush) ;
        }
        EndPaint (hwnd, &ps) ;
    }
}

```

```

        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

在 WM_PAINT 讯息处理期间，程式呼叫了 65 次 FillRect 函式，每次都使用不同灰阶建立的画刷。灰阶值是 RGB 值 (0, 0, 0)、(4, 4, 4)、(8, 8, 8) 等等，直到最後一个值 (255, 255, 255)。最後一个值来自 CreateSolidBrush 函式中的 min 巨集。

如果在 256 色显示模式下执行该程式，您将看到从黑到白的 65 种灰阶，而且它们几乎都用混色著色。纯颜色只有黑色、暗灰色 (128, 128, 128)、亮灰色 (192, 192, 192) 和白色。其他颜色是混合了这些纯颜色的多位元模式。如果我们在显示行或文字，而不是用这 65 种灰阶填充区域，Windows 将不使用混色而只使用这四种纯色。如果我们正在显示点阵图，则图像将用 20 种标准 Windows 颜色近似。这时正如同您在执行最後一章中的程式的同时又载入了彩色或灰阶 DIB 所见到的一样。通常，Windows 在点阵图中不使用混色。

程式 16-2 所示的 GRAYS2 程式用较少的外部程式码验证了调色盘管理器中最重要的函式和讯息。

程式 16-2 GRAYS2

```

GRAYS2.C
/*-----
-
    GRAYS2.C --      Gray Shades Using Palette Manager
                        (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Grays2") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
    wndclass.lpfnWndProc
        = CS_HREDRAW | CS_VREDRAW ;
        = WndProc ;

```

```

    wndclass.cbClsExtra           = 0 ;
    wndclass.cbWndExtra           = 0 ;
    wndclass.hInstance           = hInstance ;
    wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName         = NULL ;
    wndclass.lpszClassName        = szAppName ;

    if (! RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                    szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Shades of Gray #2"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HPALETTE          hPalette ;
    static int               cxClient, cyClient ;
    HBRUSH                   hBrush ;
    HDC                      hdc ;
    int                      i ;
    LOGPALETTE               *    plp ;
    PAINTSTRUCT              ps ;
    RECT                     rect ;

    switch (message)
    {
    case WM_CREATE:

```



```

// Set up a LOGPALETTE structure and create
a palette

plp = malloc (sizeof (LOGPALETTE) + 64 * sizeof
(PALETTEENTRY)) ;

plp->palVersion = 0x0300 ;
plp->palNumEntries = 65 ;

for (i = 0 ; i < 65 ; i++)
{
    plp->palPalEntry[i].peRed = (BYTE) min (255, 4 * i) ;
    plp->palPalEntry[i].peGreen = (BYTE) min (255, 4 * i) ;
    plp->palPalEntry[i].peBlue = (BYTE) min (255, 4 * i) ;
    plp->palPalEntry[i].peFlags = 0 ;
}
hPalette = CreatePalette (plp) ;
free (plp) ;
return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Select and realize the palette in the device
context

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    // Draw the fountain of grays

    for (i = 0 ; i < 65 ; i++)
    {
        rect.left = i * cxClient / 64 ;
        rect.top = 0 ;
        rect.right = (i + 1) * cxClient
/ 64 ;
        rect.bottom = cyClient ;

        hBrush = CreateSolidBrush (PALETTE_RGB( min
(255, 4 * i),
                                min (255, 4 * i),
                                min (255, 4 * i))) ;
        FillRect (hdc, &rect, hBrush) ;
    }

```

```

        DeleteObject (hBrush) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    DeleteObject (hPalette) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

通常，使用调色盘管理器的第一步就是呼叫 CreatePalette 函式来建立逻辑调色盘。逻辑调色盘包含程式所需要的全部颜色——即 236 种颜色。GRAYS1 程式在 WM_CREATE 讯息处理期间处理此作业。它初始化 LOGPALETTE (「logical palette: 逻辑调色盘」) 结构的栏位，并将这个结构的指标传递给 CreatePalette 函式。CreatePalette 传回逻辑调色盘的代号，并将此代号储存在静态变数 hPalette 中。

LOGPALETTE 结构定义如下：

```

typedef struct
{

```

```

        WORD            palVersion ;
        WORD            palNumEntries ;
        PALETTEENTRY    palPalEntry[1] ;
    }
LOGPALETTE, * PLOGPALETTE ;

```

第一个栏位通常设为 0x0300，表示相容 Windows 3.0。第二个栏位设定为调色盘表中的项目数。LOGPALETTE 结构中的第三个栏位是一个 PALETTEENTRY 结构的阵列，此结构也是一个调色盘项目。PALETTEENTRY 结构定义如下：

```

typedef struct
{
    BYTE peRed ;
    BYTE peGreen ;
    BYTE peBlue ;
    BYTE peFlags ;
}
PALETTEENTRY, * PPALETTEENTRY ;

```

每个 PALETTEENTRY 结构都定义了一个我们要在调色盘中使用的 RGB 颜色值。

注意，LOGPALETTE 中只能定义一个 PALETTEENTRY 结构的阵列。您需要为 LOGPALETTE 结构和附加的 PALETTEENTRY 结构配置足够大的记忆体空间。GRAYS2 需要 65 种灰阶，因此它为 LOGPALETTE 结构和 64 个附加的 PALETTEENTRY 结构配置了足够大的记忆体空间。GRAYS2 将 palNumEntries 栏位设定为 65，然後从 0 到 64 回圈，计算灰阶等级（一般是回圈索引的 4 倍，但不超过 255），将结构中的 peRed、peGreen 和 peBlue 栏位设定为此灰阶等级。peFlags 栏位设为 0。程式将指向这个记忆体块的指标传递给 CreatePalette，在一个静态变数中储存该调色盘代号，然後释放记忆体。

逻辑调色盘是 GDI 物件。程式应该删除它们建立的所有逻辑调色盘。WndProc 透过在 WM_DESTROY 讯息处理期间呼叫 DeleteObject，仔细地删除了逻辑调色盘。

注意逻辑调色盘是独立的装置内容。在真正使用之前，必须确保将其选进装置内容。在 WM_PAINT 讯息处理期间，SelectPalette 将逻辑调色盘选进装置内容。除了含有第三个参数以外，此函式与 SelectObject 函式相似。通常，第三个参数设为 FALSE。如果 SelectPalette 的第三个参数设为 TRUE，那么调色盘将始终是「背景调色盘」，这意味著当其他所有程式都显现了各自的调色盘之後，该调色盘才可以获得仍位於系统调色盘中的一个未使用项目。

在任何时候都只有一个逻辑调色盘能选进装置内容。函式将传回前一个选进装置内容的逻辑调色盘代号。如果您希望将此逻辑调色盘重新选进装置内容，则可以储存此代号。

通过将颜色映射到系统调色盘，RealizePalette 函式使 Windows 在装置内

容中「显现」逻辑调色盘，而系统调色盘是与显示卡实际的调色盘相对应。实际工作在此函数呼叫期间进行。Windows 必须决定呼叫函数的视窗是活动的还是非活动的，并尽可能将系统调色盘已改变通知给其他视窗（我们将简要说明一下通知的程序）。

回忆一下 GRAYS1，它用 RGB 巨集来指定纯色画刷的颜色。RGB 巨集建构一个 32 位元长整数（记作 COLORREF 值），其中高位元组是 0，3 个低位元组是红、绿和蓝的亮度。

使用 Windows 调色盘管理器的程式可以继续使用 RGB 颜色值来指定颜色。不过，这些 RGB 颜色值将不能存取逻辑调色盘中的附加颜色。它们的作用与没有使用调色盘管理器相同。要在逻辑调色盘中使用附加的颜色，就要用到 PALETTE_RGB 巨集。除了 COLORREF 值的高位元组设为 2 而不是 0 以外，「调色盘 RGB」颜色与 RGB 颜色很相似。

下面是重要的规则：

- 为了使用逻辑调色盘中的颜色，请用调色盘 RGB 值或调色盘索引来指定（我将简要讨论调色盘索引）。不要使用常规的 RGB 值。如果使用了常规的 RGB 值，您将得到一种标准颜色，而不是逻辑调色盘中的颜色。
- 没有将调色盘选进装置内容时，不要使用调色盘 RGB 值或调色盘索引。
- 尽管可以使用调色盘 RGB 值来指定逻辑调色盘中没有的颜色，但您还是要从逻辑调色盘获得颜色。

例如，在 GRAYS2 中处理 WM_PAINT 期间，当您选择并显现了逻辑调色盘之後，如果试图显示红色，则将显示灰阶。您必须用 RGB 颜色值来选择不在逻辑调色盘中的颜色。

注意，GRAYS2 从不检查视讯显示驱动程式是否支援调色盘管理程式。在不支援调色盘管理程式的显示模式（即所有非 256 种颜色的显示模式）下执行 GRAYS2 时，GRAYS2 的功能与 GRAYS1 相同。

调色盘资讯

如果程式在逻辑调色盘中指定一种颜色，该颜色又是 20 种保留颜色之一，那么 Windows 将把逻辑调色盘项目映射给该颜色。另外，如果两个或多个应用程式都在它们的逻辑调色盘中指定了同一种颜色，那么这些应用程式将共用系统调色盘项目。程式可以通过将 PALETTEENTRY 结构的 peFlags 栏位指定为常数 PC_NOCOLLAPSE 来忽略该内定状态（其余两个可能的标记是 PC_EXPLICIT（用於显示系统调色盘）和 PC_RESERVED（用於调色盘动画），我将在本章的後面展示这两个标记）。

要帮助组织系统调色盘，Windows 调色盘管理器含有两个发送给主视窗的讯息。

第一个是 `WM_QUERYNEWPALETTE`。当主视窗活动时，该讯息发送给主视窗。如果程式在您的视窗上绘画时使用了调色盘管理器，则它必须处理该讯息。`GRAYS2` 展示具体的作法。程式获得装置内容代号，并选进调色盘，呼叫 `RealizePalette`，然後使视窗失效以产生 `WM_PAINT` 讯息。如果显现了逻辑调色盘，则视窗讯息处理程式从该讯息传回 `TRUE`，否则传回 `FALSE`。

当系统调色盘改成与 `WM_QUERYNEWPALETTE` 讯息的结果相同时，Windows 将 `WM_PALETTECHANGED` 讯息发送给由目前活动的视窗来启动并终止处理视窗链的所有主视窗。这允许前台视窗有优先权。传递给视窗讯息处理程式的 `wParam` 值是活动视窗的代号。只有当 `wParam` 不等於程式的视窗代号时，使用调色盘管理器的程式才会处理该讯息。

通常，在处理 `WM_PALETTECHANGED` 时，使用自订调色盘的任何程式都呼叫 `SelectPalette` 和 `RealizePalette`。後续的视窗在讯息处理期间呼叫 `RealizePalette` 时，Windows 首先检查逻辑调色盘中的 RGB 颜色是否与已载入到系统调色盘中的 RGB 颜色相匹配。如果两个程式需要相同的颜色，那么这两个程式就共同使用一个系统调色盘项目。接下来，Windows 检查未使用的系统调色盘项目。如果都已使用，则逻辑调色盘中的颜色从 20 种保留项目映射到最近的颜色。

如果不关心程式非活动时显示区域的外观，那么您不必处理 `WM_PALETTECHANGED` 讯息。否则，您有两个选择。`GRAYS2` 显示其中之一：在处理 `WM_QUERYNEWPALETTE` 讯息时，它获得装置内容，选进调色盘，然後呼叫 `RealizePalette`。这时就可以在处理 `WM_QUERYNEWPALETTE` 时呼叫 `InvalidateRect` 了。相反地，`GRAYS2` 呼叫 `UpdateColors`。这个函式通常比重新绘制视窗更有效，同时它改变视窗中图素的值来帮助保护以前的颜色。

使用调色盘管理器的许多程式都将让 `WM_QUERYNEWPALETTE` 和 `WM_PALETTECHANGED` 讯息用 `GRAYS2` 所显示的方法来处理。

调色盘索引方法

程式 16-3 所示的 `GRAYS3` 程式与 `GRAYS2` 非常相似，只是在处理 `WM_PAINT` 期间使用了呼叫 `PALETTEINDEX` 的巨集，而不是 `PALETTEINDEX`。

程式 16-3 GRAYS3

```
GRAYS3.C
```

```
/*-----  
-
```

```

GRAYS3.C --          Gray Shades Using Palette Manager
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("Grays3") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;
    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Shades of Gray #3"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

```

```

        return msg.wParam ;
    }

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static HPALETTE          hPalette ;
    static int               cxClient, cyClient ;
    HBRUSH                   hBrush ;
    HDC                      hdc ;
    int                      i ;
    LOGPALETTE               *      plp ;
    PAINTSTRUCT              ps ;
    RECT                    rect ;
    switch (message)
    {
        case WM_CREATE:
            //      Set up a LOGPALETTE structure and create a palette

            plp=malloc (sizeof (LOGPALETTE) + 64 * sizeof (PALETTEENTRY)) ;

            plp->palVersion      = 0x0300 ;
            plp->palNumEntries   = 65 ;

            for (i = 0 ; i < 65 ; i++)
            {
                plp->palPalEntry[i].peRed      = (BYTE) min (255, 4 * i) ;
                plp->palPalEntry[i].peGreen    = (BYTE) min (255, 4 * i) ;
                plp->palPalEntry[i].peBlue     = (BYTE) min (255, 4 * i) ;
                plp->palPalEntry[i].peFlags    = 0 ;
            }
            hPalette = CreatePalette (plp) ;
            free (plp) ;
            return 0 ;

        case WM_SIZE:
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;
            return 0 ;

        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps) ;

            // Select and realize the palette in the device context

            SelectPalette (hdc, hPalette, FALSE) ;
            RealizePalette (hdc) ;
    }
}

```

```
        // Draw the fountain of grays

        for (i = 0 ; i < 65 ; i++)
        {
            rect.left      = i * cxClient / 64 ;
            rect.top       = 0 ;
            rect.right     = (i + 1) * cxClient / 64 ;
            rect.bottom    = cyClient ;

            hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;

            FillRect (hdc, &rect, hBrush) ;
            DeleteObject (hBrush) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_QUERYNEWPALETTE:
        if (!hPalette)
            return FALSE ;

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        InvalidateRect (hwnd, NULL, FALSE) ;

        ReleaseDC (hwnd, hdc) ;
        return TRUE ;

    case WM_PALETTECHANGED:
        if (!hPalette || (HWND) wParam == hwnd)
            break ;

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        UpdateColors (hdc) ;

        ReleaseDC (hwnd, hdc) ;
        break ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```


「调色盘」索引的颜色不同於调色盘 RGB 颜色，其高位元组是 1，而低位元组的值是目前在装置内容中选择的、逻辑调色盘中的索引。在 GRAYS3 中，逻辑调色盘有 65 个项目，用於这些项目的索引从 0 到 64。值

PALETTEINDEX (0)

指黑色，

PALETTEINDEX (32)

指灰色，而

PALETTEINDEX (64)

指白色。

因为 Windows 不需要执行最近颜色的搜索，所以使用调色盘索引比使用 RGB 值更有效。

查询调色盘支援

您可以容易地验证：当 Windows 在 16 位元或 24 位元显示模式下执行时，GRAYS2 和 GRAYS3 程式执行良好。但是在某些情况下，要使用调色盘管理器的 Windows 應用程式可能要先确定装置驱动程式是否支援它。这时，您可以呼叫 `GetDeviceCaps`，并以视讯显示的装置内容代号和 `RASTERCAPS` 作为参数。函式将传回由一系列旗标组成的整数。通过在传回值和常数 `RC_PALETTE` 之间执行位元操作来检验支援的调色盘：

```
RC_PALETTE & GetDeviceCaps (hdc, RASTERCAPS)
```

如果此值非零，则视讯显示器装置驱动程式将支援调色盘操作。在这种情况下，来自 `GetDeviceCaps` 的其他三个重要项目也是可用的。函式呼叫

```
GetDeviceCaps (hdc, SIZEPALETTE)
```

将传回在显示卡上调色盘表的总尺寸。这与同时显示的颜色总数相同。因为调色盘管理器只用於每图素 8 位元的视讯显示模式，所以此值将是 256。

函式呼叫

```
GetDeviceCaps (hdc, NUMRESERVED)
```

传回在调色盘表中的颜色数，该表是装置驱动程式为系统保留的，此值是 20。不呼叫调色盘管理器，这些只是 Windows 應用程式在 256 色显示模式下使用的纯色。要使用其余的 236 种颜色，程式必须使用调色盘管理器函式。

一个附加项目也可用：

```
GetDeviceCaps (hdc, COLORRES)
```

此值告诉您载入到硬体调色盘表的 RGB 颜色值解析度（以位元计）。这些是进入数位类比转换器的位元。某些视讯显示卡只使用 6 位元 ADC，所以该值是 18。其余使用 8 位元的 ADC，所以值是 24。

Windows 程式注意颜色解析度并因此采取一些动作是很有用的。例如，如果该颜色解析度是 18，那么程式将不可能要求到 128 种灰阶，因为只有 64 个离散的灰阶可用。要求到 128 种灰阶就不必用多余的项目来填充硬体调色盘表。

系统调色盘

我在前面提过，Windows 系统调色盘直接与显示卡上的硬体调色盘查询表相符（然而，硬体调色盘查询表可能比系统调色盘的颜色解析度低）。程式可以通过呼叫下面的函式来获得系统调色盘中的某些或全部的 RGB 项目：

```
GetSystemPaletteEntries (hdc, uStart, uNum, &pe) ;
```

只有显示卡模式支援调色盘操作时，该函式才能执行。第二个和第三个参数是无正负号整数，显示第一个调色盘项目的索引和调色盘项目数。最后一个参数是指向 PALETTEENTRY 型态的指标。

您可以在几种情况下使用该函式。程式可以定义 PALETTEENTRY 结构如下：

```
PALETTEENTRY pe ;
```

然後可按下面的方法多次呼叫 GetSystemPaletteEntries：

```
GetSystemPaletteEntries (hdc, i, 1, &pe) ;
```

其中的 i 从 0 到某个值，该值小於从 GetDeviceCaps（带有 SIZEPALETTE 索引 255）传回的值。或者，程式要获得所有的系统调色盘项目，可以通过定义指向 PALETTEENTRY 结构的指标，然後重新配置足够的记忆体块，以储存与调色盘大小指定同样多的 PALETTEENTRY 结构。

GetSystemPaletteEntries 函式确实允许您检验硬体调色盘表。系统调色盘中的项目按图素值增加的顺序排列，这些值用於表示视讯显示缓冲区中的颜色。我将简单地讨论一下具体作法。

其他调色盘函式

我们在前面看过，Windows 程式能够改变系统调色盘，但只是间接改变：第一步建立逻辑调色盘，它基本上是程式要使用的 RGB 颜色值阵列。CreatePalette 函式不会导致系统调色盘或者显示卡调色盘表的任何变化。逻辑调色盘必须在任何事情发生之前就选进装置内容并显现。

程式可以通过呼叫

```
GetPaletteEntries (hPalette, uStart, uNum, &pe) ;
```

来查询逻辑调色盘中的 RGB 颜色值。您可以按使用 GetSystemPaletteEntries 的方法来使用此函式。但是要注意，第一个参数是逻辑调色盘的代号，而不是装置内容的代号。

建立逻辑调色盘以後，让您改变其中的值的相应函式是：

```
SetPaletteEntries (hPalette, uStart, uNum, &pe) ;
```

另外，记住呼叫此函式不引起系统调色盘的任何变化——即使目前调色盘选进了装置内容。此函式也不改变逻辑调色盘的尺寸。要改变逻辑调色盘的尺寸，请使用 `ResizePalette`。

下面的函式接受 RGB 颜色引用值作为最後的参数，并将索引传回给逻辑调色盘，该逻辑调色盘与和它最接近的 RGB 颜色值相对应：

```
iIndex = GetNearestPaletteIndex (hPalette, cr) ;
```

第二个参数是 `COLORREF` 值。如果希望的话，呼叫 `GetPaletteEntries` 就可以获得逻辑调色盘中实际的 RGB 颜色值。

如果程式在 8 位元显示模式下需要多於 236 种自订颜色，则可以呼叫 `GetSystemPaletteUse`。这允许程式设定 254 种自订颜色；系统仅保留黑色和白色。不过，程式仅在最大化充满全萤幕时才允许这样，而且它还将一些系统颜色设为黑色和白色，以便标题列和功能表等仍然可见。

位元映射操作问题

从第五章可以了解到，GDI 允许使用不同的「绘画模式」或「位元映射操作」来画线并填充区域。用 `SetROP2` 设定绘画模式，其中的「2」表示两个物件之间的二元 (binary) 位元映射操作。三元位元映射操作用於处理 `BitBlt` 和类似功能。这些位元映射操作决定了正在画的物件图素与表面图素的结合方式。例如，您可以画一条直线，以便线上的图素与显示的图素按位元异或的方式相结合。

位元映射操作就是在图素位元上照著各个位元的顺序进行操作。改变调色盘会影响到这些位元映射操作。位元映射操作的操作物件是图素位元，而这些图素位元可能与实际颜色没有关联。

透过执行 `GRAYS2` 或 `GRAYS3` 程式，您自己就可以得出这个结论。调整尺寸时，拖动顶部或底部的边界穿过视窗，Windows 利用反转背景图素位元的位元映射操作来显示拖动尺寸的边界，其目的是使拖动尺寸边界总是可见的。但在 `GRAYS2` 和 `GRAYS3` 程式中，您将看到各种随机变换的颜色，这些颜色恰好与对应於调色盘表中未使用的项目，那是反转显示图素位元的结果。可视颜色没有反转——只有图素位元反转了。

正如您在表 16-1 中所看到的一样，20 种标准保留颜色位於系统调色盘的顶部和底部，以便位元映射操作的结果仍然正常。然而，一旦您开始修改调色盘——尤其是替换了保留颜色——那么颜色物件的位元映射操作就变得没有意义了。

唯一保证的是位元映射操作将用黑色和白色运作。黑色是系统调色盘中的第一个项目（所有的图素位元都设为 0），而白色是最後的项目（所有的图素位

元都设为 1)。这两个项目不能改变。如果需要预知在颜色物件上进行位元映射操作的结果，则可以先获得系统调色盘表，然後查看不同图素位元值的 RGB 颜色值。

查看系统调色盘

在 Windows 下执行的程式将处理逻辑调色盘，为使逻辑调色盘更好地服务於所有使用逻辑调色盘的程式，Windows 将在系统调色盘中设定颜色。该系统调色盘复制了显示卡的硬体对照表内容。这样，查看系统调色盘有助於调适调色盘应用程式。

因为对於这个问题有三种截然不同的处理方式，所以我将向您展示三个程式，以显示系统调色盘的内容。

SYSPAL1 程式，如程式 16-4 所示，使用了前面所讲的 GetSystemPaletteEntries 函式。

程式 16-4 SYSPAL1

```
SYSPAL1.C
/*-----
-
    SYSPAL1.C --    Displays system palette
                                (c) Charles Petzold, 1998
-----
*/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("SysPal1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;
    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
```

```
wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName       = NULL ;
wndclass.lpszClassName      = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),
                szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("System Palette #1"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

if (!hwnd)
    return 0 ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

BOOL CheckDisplay (HWND hwnd)
{
    HDC    hdc ;
    int    iPalSize ;

    hdc = GetDC (hwnd) ;
    iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
    ReleaseDC (hwnd, hdc) ;

    if (iPalSize != 256)
    {
        MessageBox (hwnd, TEXT ("This program requires that the video ")
                    TEXT ("display mode have a 256-color palette."),
                    szAppName, MB_ICONERROR) ;

        return FALSE ;
    }
    return TRUE ;
}
```

```

}

LRESULT CALLBACK WndProc (   HWND  hwnd,   UINT  message,   WPARAM  wParam,LPARAM
lParam)
{
    static int           cxClient, cyClient ;
    static SIZE          sizeChar ;
    HDC                 hdc ;
    HPALETTE             hPalette ;
    int                 i, x, y ;
    PAINTSTRUCT          ps ;
    PALETTEENTRY         pe [256] ;
    TCHAR                                                         szBuffer [16] ;

    switch (message)
    {
    case WM_CREATE:
        if (!CheckDisplay (hwnd))
            return -1 ;
        hdc = GetDC (hwnd) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
        GetTextExtentPoint32 (hdc, TEXT ("FF-FF-FF"), 10,
&sizeChar) ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

    case WM_DISPLAYCHANGE:
        if (!CheckDisplay (hwnd))
            DestroyWindow (hwnd) ;

        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

        GetSystemPaletteEntries (hdc, 0, 256, pe) ;

        for (i = 0, x = 0, y = 0 ; i < 256 ; i++)
        {
            wsprintf ( szBuffer, TEXT ("%02X-%02X-%02X"),
pe[i].peRed, pe[i].peGreen, pe[i].peBlue) ;

```

```

        TextOut (hdc, x, y, szBuffer, lstrlen (szBuffer)) ;

        if (( x += sizeChar.cx) + sizeChar.cx > cxClient)
        {
            x = 0 ;

            if (( y += sizeChar.cy) > cyClient)
                break ;
        }
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_PALETTECHANGED:
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

与 SYSPAL 系列中的其他程式一样，除非带有 SIZEPALETTE 参数的 GetDeviceCaps 传回值为 256，否则 SYSPAL1 不会执行。

注意无论 SYSPAL1 的显示区域什么时候收到 WM_PALETTECHANGED 讯息，它都是无效的。在合并 WM_PAINT 讯息处理期间，SYSPAL1 呼叫 GetSystemPaletteEntries，并用一个含 256 个 PALETTEENTRY 结构的阵列作为参数。RGB 值作为文字字串显示在显示区域。程式执行时，注意 20 种保留颜色是 RGB 值列表中的前 10 个和後 10 个，这与表 16-1 所示相同。

当 SYSPAL1 显示有用的资讯时，它与实际看到的 256 种颜色不同。那就是 SYSPAL2 的作业，如程式 16-5 所示。

程式 16-5 SYSPAL2

```

SYSPAL2.C
/*-----
-
SYSPAL2.C --   Displays system palette
                                   (c) Charles Petzold, 1998
-----
*/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

```

```

TCHAR szAppName [] = TEXT ("SysPal2") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("System Palette #2"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    if (!hwnd)
        return 0 ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

```



```

BOOL CheckDisplay (HWND hwnd)
{
    HDC hdc ;
    int iPalSize ;

    hdc = GetDC (hwnd) ;
    iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
    ReleaseDC (hwnd, hdc) ;

    if (iPalSize != 256)
    {
        MessageBox (hwnd, TEXT ("This program requires that the video ")
            TEXT ("display mode have a 256-color palette."),
            szAppName, MB_ICONERROR) ;
        return FALSE ;
    }
    return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static HPALETTE          hPalette ;
    static int               cxClient, cyClient ;
    HBRUSH                   hBrush ;
    HDC                      hdc ;
    int                      i, x, y ;
    LOGPALETTE               *      plp ;
    PAINTSTRUCT              ps ;
    RECT                     rect ;

    switch (message)
    {
    case WM_CREATE:
        if (!CheckDisplay (hwnd))
            return -1 ;

        plp=malloc (sizeof (LOGPALETTE) + 255 * sizeof (PALETTEENTRY)) ;

        plp->palVersion          = 0x0300 ;
        plp->palNumEntries       = 256 ;

        for (i = 0 ; i < 256 ; i++)
        {
            plp->palPalEntry[i].peRed          = i ;
            plp->palPalEntry[i].peGreen        = 0 ;
            plp->palPalEntry[i].peBlue         = 0 ;
        }
    }
}

```

```

                                plp->palPalEntry[i].peFlags =
PC_EXPLICIT ;
                                }

                                hPalette = CreatePalette (plp) ;
                                free (plp) ;
                                return 0 ;

case WM_DISPLAYCHANGE:
    if (!CheckDisplay (hwnd))
        DestroyWindow (hwnd) ;

    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    for (y = 0 ; y < 16 ; y++)
    for (x = 0 ; x < 16 ; x++)
    {
        hBrush = CreateSolidBrush (PALETTEINDEX (16
* y + x)) ;
        SetRect (&rect, x      * cxClient / 16, y*
cyClient / 16,
        (x + 1) *      cxClient      /      16, (y+1) *
cyClient / 16);
        FillRect (hdc, &rect, hBrush) ;
        DeleteObject (hBrush) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_PALETTECHANGED:
    if ((HWND) wParam != hwnd)
        InvalidateRect (hwnd, NULL, FALSE) ;

    return 0 ;

case WM_DESTROY:
    DeleteObject (hPalette) ;

```

```

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

SYSPAL2 在 WM_CREATE 讯息处理期间建立了逻辑调色盘。但是请注意：逻辑调色盘中所有的 256 个值都是从 0 到 255 的调色盘索引，并且 peFlags 栏位是 PC_EXPLICIT。该旗标是这样定义的：「逻辑调色盘项目的较低字组指定了一个硬体调色盘索引。此旗标允许應用程式显示硬体调色盘的内容。」该旗标就是专为我们要做的这件事情而设计的。

在 WM_PAINT 讯息处理期间，SYSPAL2 将该调色盘选进装置内容并显现它。这不会引起系统调色盘的任何重组，而是允许程式使用 PALETTEINDEX 巨集来指定系统调色盘中的颜色。按此方法，SYSPAL2 显示了 256 个矩形。另外，当您执行该程式时，注意顶行和底行的前 10 种和後 10 种颜色是 20 种保留颜色，如表 16-1 所示。当您执行使用自己逻辑调色盘的程式时，显示就改变了。

如果您既喜欢看 SYSPAL2 中的颜色，又喜欢 RGB 的值，那么请与第八章的 WHATCLR 程式同时执行。

SYSPAL 系列中的第三版使用的技术对我来说是最近才出现的——从我开始研究 Windows 调色盘管理器七年多後，才出现了那些技术。

事实上，所有的 GDI 函式都直接或间接地指定颜色作为 RGB 值。在 GDI 内部，这将转换成与那个颜色相关的图素位元。在某些显示模式中（例如，16 位元或 24 位元颜色模式），这些转换是相当直接的。在其他显示模式中（4 位元或 8 位元颜色），这可能涉及最接近颜色的搜索。

然而，有两个 GDI 函式让您直接指定图素位元中的颜色。当然在这种方式中使用的这两个函式都与设备高度相关。它们太依赖设备了，以至於它们可以直接显示视讯显示卡上实际的调色盘对照表。这两个函式是 BitBlt 和 StretchBlt。

程式 16-6 所示的 SYSPAL3 程式显示了使用 StretchBlt 显示系统调色盘中颜色的方法。

程式 16-6 SYSPAL3

```

SYSPAL3.C
/*-----
-
SYSPAL3.C --      Displays system palette
(c) Charles Petzold, 1998
-----
*/

```

```

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("SysPal3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
        MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("System Palette #3"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    if (!hwnd)
        return 0 ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

```

```
BOOL CheckDisplay (HWND hwnd)
{
    HDC hdc ;
    int iPalSize ;

    hdc = GetDC (hwnd) ;
    iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
    ReleaseDC (hwnd, hdc) ;

    if (iPalSize != 256)
    {
        MessageBox (hwnd,TEXT("This program requires that the video ")
                    TEXT("display mode have a 256-color palette."),
                    szAppName, MB_ICONERROR) ;
        return FALSE ;
    }
    return TRUE ;
}

LRESULT CALLBACK WndProc (  HWND hwnd,  UINT message,  WPARAM wParam,LPARAM
lParam)
{
    static HBITMAP hBitmap ;
    static int      cxClient, cyClient ;
    BYTE            bits [256] ;
    HDC              hdc, hdcMem ;
    int              i ;
    PAINTSTRUCT      ps ;

    switch (message)
    {
    case WM_CREATE:
        if (! CheckDisplay (hwnd))
            return -1 ;

        for ( i = 0 ; i < 256 ; i++)
            bits [i] = i ;

        hBitmap = CreateBitmap (16, 16, 1, 8, &bits) ;
        return 0 ;

    case WM_DISPLAYCHANGE:
        if (!CheckDisplay (hwnd))
            DestroyWindow (hwnd) ;

        return 0 ;
    }
```

```
case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;

    StretchBlt (hdc, 0, 0, cxClient, cyClient,
                hdcMem, 0, 0, 16, 16, SRCCOPY) ;

    DeleteDC (hdcMem) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在 WM_CREATE 讯息处理期间, SYSPAL3 使用 CreateBitmap 来建立 16 16 的每图素 8 位元的点阵图。该函式的最後一个参数是包括数值 0 到 255 的 256 位元组阵列。这些是 256 种可能的图素位元值。在处理 WM_PAINT 讯息的程序中, 程式将这个点阵图选进记忆体装置内容, 用 StretchBlt 来显示并填充该显示区域。Windows 仅将点阵图中的图素位元传输到视讯显示器硬体, 从而允许这些图素位元存取调色盘对照表中的 256 个项目。程式的显示区域甚至不必使接收 WM_PALETTECHANGED 讯息无效——对於对照表的任何修改都会立即影响到 SYSPAL3 的显示。

调色盘动画

在本节的标题中看到「动画」一词, 并开始考虑萤幕周围执行的「电脑宠物」时, 您的眼前可能会为之一亮。是的, 您可以使用 Windows 调色盘管理器作一些动画, 而且是有一定专业水平的动画。

通常, Windows 下的动画就是快速连续地显示一系列点阵图。调色盘动画与这种方法有很大的区别。您透过在萤幕上绘制您所需要的每件东西开始, 然後您处理调色盘来改变这些物件的颜色, 可能是画一些相对於萤幕背景来说是不可见的图像。您用这种方法就可以获得动画效果, 而不必重画任何东西。调色

盘动画的速度是相当快的。

对于调色盘动画，最初的建立工作与我们前面看见的有些不同：对于动画期间要修改的每种 RGB 颜色值，PALETTEENTRY 结构的 peFlags 栏位必须设定为 PC_RESERVED。

通常，就像我们所看到的一样，在建立逻辑调色盘时，您将 peFlags 标记设为 0。这允许 GDI 将多个逻辑调色盘中同样的颜色映射到相同的系统调色盘项目。例如，假设两个 Windows 程式都建立了包含 RGB 项目 10-10-10 的逻辑调色盘，那么在系统调色盘表中，Windows 只需要一个 10-10-10 项目。但如果这两个程式中的一个使用调色盘动画，那您就不要再让 GDI 使用调色盘了。调色盘动画意味著速度非常快——而且如果不重画，它也只可能提高速度。当使用调色盘动画的程式修改调色盘时，它不会影响其他程式，或者迫使 GDI 重组系统调色盘表。PC_RESERVED 的 peFlags 值为单个逻辑调色盘储存系统调色盘项目。

使用调色盘动画时，通常您可以在 WM_PAINT 讯息处理期间呼叫 SelectPalette 和 RealizePalette，使用 PALETTEINDEX 巨集来指定颜色。该巨集将一个索引带进逻辑调色盘表。

对于动画，您可能要通过改变调色盘来回应 WM_TIMER 讯息。要改变逻辑调色盘中的 RGB 颜色值，请使用一个 PALETTEENTRY 结构的阵列来呼叫函式 AnimatePalette。此函式速度很快，因为它只需要改变系统调色盘以及显示卡硬体调色盘表中的项目。

跳动的球

程式 16-7 显示了 BOUNCE 程式的元件，但还有一个程式可显示跳动的球。为了简单起见，根据显示区域的大小将球画成了椭圆形。因为本章有几个调色盘动画程式，所以 PALANIM.C (「调色盘动画」) 档案包含一些通用内容。

程式 16-7 BOUNCE

```
PALANIM.C
/*-----
    PALANIM.C --      Palette Animation Shell Program
                        s(c) Charles Petzold, 1998
    -----*/

#include <windows.h>
extern HPALETTE CreateRoutine      (HWND) ;
extern void      PaintRoutine      (HDC, int, int) ;
extern void      TimerRoutine      (HDC, HPALETTE) ;
extern void      DestroyRoutine    (HWND, HPALETTE) ;
```

```

LRESULT      CALLBA CK   WndProc   (HWND, UINT, WPARAM, LPARAM) ;

extern TCHAR szAppName [] ;
extern TCHAR szTitle [] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, szTitle,
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    if (!hwnd)
        return 0 ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

```



```
        return msg.wParam ;
    }

BOOL CheckDisplay (HWND hwnd)
{
    HDC hdc ;
    int iPalSize ;

    hdc = GetDC (hwnd) ;
    iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
    ReleaseDC (hwnd, hdc) ;

    if (iPalSize != 256)
    {
        MessageBox (hwnd, TEXT ("This program requires that the video ")
                    TEXT ("display mode have a 256-color palette."),
                    szAppName, MB_ICONERROR) ;
        return FALSE ;
    }
    return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static HPALETTE hPalette ;
    static int      cxClient, cyClient ;
    HDC             hdc ;
    PAINTSTRUCT     ps ;

    switch (message)
    {
    case WM_CREATE:
        if (!CheckDisplay (hwnd))
            return -1 ;

        hPalette = CreateRoutine (hwnd) ;
        return 0 ;

    case WM_DISPLAYCHANGE:
        if (!CheckDisplay (hwnd))
            DestroyWindow (hwnd) ;

        return 0 ;

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
    }
```

```
        return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    PaintRoutine (hdc, cxClient, cyClient) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
case WM_TIMER:
    hdc = GetDC (hwnd) ;

    SelectPalette (hdc, hPalette, FALSE) ;

    TimerRoutine (hdc, hPalette) ;

    ReleaseDC (hwnd, hdc) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    DestroyRoutine (hwnd, hPalette) ;
```

```

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
BOUNCE.C
/*-----
BOUNCE.C --      Palette Animation Demo
                                     (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Bounce") ;
TCHAR szTitle [] = TEXT ("Bounce: Palette Animation Demo") ;

static LOGPALETTE * plp ;
HPALETTE CreateRoutine (HWND hwnd)
{
    HPALETTE      hPalette ;
    int           i ;

    plp = malloc (sizeof (LOGPALETTE) + 33 * sizeof (PALETTEENTRY)) ;
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries    = 34 ;

    for (i = 0 ; i < 34 ; i++)
    {
        plp->palPalEntry[i].peRed      = 255 ;
        plp->palPalEntry[i].peGreen    = (i == 0 ? 0 : 255) ;
        plp->palPalEntry[i].peBlue     = (i == 0 ? 0 : 255) ;
        plp->palPalEntry[i].peFlags    = (i == 33 ? 0 : PC_RESERVED) ;
    }
    hPalette = CreatePalette (plp) ;
    SetTimer (hwnd, ID_TIMER, 50, NULL) ;
    return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
    HBRUSH      hBrush ;
    int         i, x1, x2, y1, y2 ;
    RECT        rect ;

    // Draw window background using palette index 33

    SetRect (&rect, 0, 0, cxClient, cyClient) ;
    hBrush = CreateSolidBrush (PALETTEINDEX (33)) ;

```

```

FillRect (hdc, &rect, hBrush) ;
DeleteObject (hBrush) ;

        // Draw the 33 balls
SelectObject (hdc, GetStockObject (NULL_PEN)) ;
for (i = 0 ; i < 33 ; i++)
{
    x1 = i * cxClient / 33 ;
    x2 = (i + 1)* cxClient / 33 ;

    if (i < 9)
    {
        y1 = i * cyClient / 9 ;
        y2 = (i + 1) * cyClient / 9 ;
    }
    else if (i < 17)
    {
        y1 = (16 - i) * cyClient / 9 ;
        y2 = (17 - i) * cyClient / 9 ;
    }
    else if (i < 25)
    {
        y1 = (i - 16) * cyClient / 9 ;
        y2 = (i - 15) * cyClient / 9 ;
    }
    else
    {
        y1 = (32 - i) * cyClient / 9 ;
        y2 = (33 - i) * cyClient / 9 ;
    }

    hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;
    SelectObject (hdc, hBrush) ;
    Ellipse (hdc, x1, y1, x2, y2) ;
    DeleteObject (SelectObject (hdc, GetStockObject
(WHITE_BRUSH))) ;
}
return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
    static BOOL bLeftToRight = TRUE ;
    static int iBall ;

    // Set old ball to white
    plp->palPalEntry[iBall].peGreen = 255 ;
    plp->palPalEntry[iBall].peBlue = 255 ;

```

```

    iBall += (bLeftToRight ? 1 : -1) ;
    if ( iBall == (bLeftToRight ? 33 : -1))
    {
        iBall = (bLeftToRight ? 31 : 1) ;
        bLeftToRight ^= TRUE ;
    }

    // Set new ball to red

    plp->palPalEntry[iBall].peGreen = 0 ;
    plp->palPalEntry[iBall].peBlue = 0 ;

    // Animate the palette

    AnimatePalette (hPalette, 0, 33, plp->palPalEntry) ;
    return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hPalette) ;
    free (plp) ;
    return ;
}

```

除非 Windows 处于支援调色盘的显示模式下，否则调色盘动画将不能工作。因此，PALANIM.C 通过呼叫 CheckDisplay 函式（与 SYSPAL 程式中的函式相同）来开始处理 WM_CREATE。

PALANIM.C 呼叫 BOUNCE.C 中的四个函式：在 WM_CREATE 讯息处理期间呼叫 CreateRoutine（在 BOUNCE 中用于建立逻辑调色盘）；在 WM_PAINT 讯息处理期间呼叫 PaintRoutine；在 WM_TIMER 讯息处理期间呼叫 TimerRoutine；在 WM_DESTROY 讯息处理期间呼叫 DestroyRoutine（在 BOUNCE 中用于清除）。在呼叫 PaintRoutine 和 TimerRoutine 之前，PALANIM.C 获得装置内容，并将其选进逻辑调色盘。在呼叫 PaintRoutine 之前，它也显现调色盘。PALANIM.C 期望 TimerRoutine 呼叫 AnimatePalette。尽管 AnimatePalette 需要从装置内容中选择调色盘，但它不需要呼叫 RealizePalette。

BOUNCE 中的球按「W」路线在显示区域中来回跳动。显示区域背景是白色，球是红色。任何时候，都可以在 33 个不重叠的位置之一看见球。这需要 34 个调色盘项目：一个用于背景，其他 33 个用于不同位置的球。在 CreateRoutine 中，BOUNCE 初始化 PALETTEENTRY 结构的一个阵列，将第一个调色盘项目（与球在左上角的位置对应）设定为红色，其他的设定为白色。注意，对于除背景以

外的所有项目，peFlags 栏位都设定为 PC_RESERVED（背景是最後的一个调色盘项目）。BOUNCE 通过将 Windows 计时器的间隔设定为 50 毫秒来终止 CreateRoutine。

BOUNCE 在 PaintRoutine 完成所有的绘画工作。视窗背景用一个实心画刷和调色盘索引 33 所指定的颜色来绘制。33 个球的颜色是依据从 0 到 32 的调色盘索引的颜色。当 BOUNCE 第一次在显示区域内绘画时，0 的调色盘索引映射成红色，其他调色盘索引映射到白色。这导致球出现在左上角。

当 WndProc 处理 WM_TIMER 讯息并呼叫 TimerRoutine 时，动画就发生了。TimerRoutine 通过呼叫 AnimatePalette 来结束，语法如下：

```
AnimatePalette (hPalette, uStart, uNum, &pe) ;
```

其中，第一个参数是调色盘代号，最後一个参数是指向阵列的指标，该阵列由一个或多个 PALETTEENTRY 结构组成。该函式改变逻辑调色盘中从 uStart 项目到 uNum 项目之间的若干项目。逻辑调色盘中新的 uStart 项目是 PALETTEENTRY 结构中的第一个成员。当心！uStart 参数是进入原始逻辑调色盘表的索引，而不是进入 PALETTEENTRY 阵列的索引。

为了方便起见，BOUNCE 使用 PALETTEENTRY 结构的阵列，该结构是建立逻辑调色盘时使用的 LOGPALETTE 结构的一部分。球的目前位置（从 0 到 32）储存在静态变数 iBall 中。在 TimerRoutine 期间，BOUNCE 将 PALETTEENTRY 成员设为白色。然後计算球的下一个位置，并将该元素设为红色。用下面的呼叫来改变调色盘：

```
AnimatePalette (hPalette, 0, 33, plp->palPalEntry) ;
```

GDI 改变 33 逻辑调色盘项目中的第一个（尽管实际上只改变了两个），使它与系统调色盘表中的变化相对应，然後修改显示卡上的硬体调色盘表。这样，不用重画球就开始移动了。

BOUNCE 执行时，您会发现同时执行 SYSPAL2 或 SYSPAL3 效果会更好。

尽管 AnimatePalette 执行得非常快，但是当只有一两个项目改变时，您还应该尽量避免改变所有的逻辑调色盘项目。这在 BOUNCE 中有点复杂，因为球要来回地跳——iBall 要先增加，然後再减少。一种方法是使用两个变数：分别称为 iBallOld（设定球的目前位置）和 iBallMin（iBall 和 iBallOld 中较小的）。然後您就可以像下面这样呼叫 AnimatePalette 来改变两个项目了：

```
iBallMin = min (iBall, iBallOld) ;  
AnimatePalette (hPal, iBallMin, 2, plp->palPalEntry + iBallMin) ;
```

还有另一种方法：我们先假定您定义了一个 PALETTEENTRY 结构：

```
PALETTEENTRY pe ;
```

在 TimerRoutine 期间，您将 PALETTEENTRY 栏位设为白色，并呼叫 AnimatePalette 来改变逻辑调色盘中 iBall 位置的一个项目：

```

pe.peRed      = 255 ;
    pe.peGreen    = 255 ;
    pe.peBlue     = 255 ;
    pe.peFlags    = PC_RESERVED ;
    AnimatePalette (hPalette, iBall, 1, &pe) ;

```

然後计算显示在 BOUNCE 中的 iBall 的新值，将 PALETTEENTRY 结构的栏位定义为红色，然後再次呼叫 AnimatePalette：

```

pe.peRed      = 255 ;
    pe.peGreen    = 0 ;
    pe.peBlue     = 0 ;
    pe.peFlags    = PC_RESERVED ;
    AnimatePalette (hPalette, iBall, 1, &pe) ;

```

尽管跳动的球是对动画的一个传统的简单说明，但它实际上并不适合调色盘动画，因为必须先画出球的所有可能位置。调色盘动画更适合於显示运动的重复图案。

一个项目的调色盘动画

调色盘动画中一个更有趣的方面就是，可以只使用一个调色盘项目来完成一些有趣的技术。例如程式 16-8 所示的 FADER 程式。这个程式也需要前面的 PALANIM.C 档案。

程式 16-8 FADER

```

FADER.C
/*-----
    FADER.C -- Palette Animation Demo
                                     (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Fader") ;
TCHAR szTitle [] = TEXT ("Fader: Palette Animation Demo") ;

static LOGPALETTE lp ;
HPALETTE CreateRoutine (HWND hwnd)
{
    HPALETTE hPalette ;
    lp.palVersion          = 0x0300 ;
    lp.palNumEntries       = 1 ;
    lp.palPalEntry[0].peRed      = 255 ;
    lp.palPalEntry[0].peGreen    = 255 ;
    lp.palPalEntry[0].peBlue     = 255 ;
    lp.palPalEntry[0].peFlags    = PC_RESERVED ;

```

```
    hPalette = CreatePalette (&lp) ;
    SetTimer (hwnd, ID_TIMER, 50, NULL) ;
    return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
    static TCHAR          szText [] = TEXT (" Fade In and Out ") ;
    int                   x, y ;
    SIZE                   sizeText ;

    SetTextColor (hdc, PALETTEINDEX (0)) ;
    GetTextExtentPoint32 (hdc, szText, lstrlen (szText), &sizeText) ;

    for (x = 0 ; x < cxClient ; x += sizeText.cx)
        for (y = 0 ; y < cyClient ; y += sizeText.cy)
        {
            TextOut (hdc, x, y, szText, lstrlen (szText)) ;
        }

    return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
    static BOOL bFadeIn = TRUE ;
    if (bFadeIn)
    {
        lp.palPalEntry[0].peRed  -= 4 ;
        lp.palPalEntry[0].peGreen -= 4 ;

        if ( lp.palPalEntry[0].peRed == 3)
            bFadeIn = FALSE ;
    }
    else
    {
        lp.palPalEntry[0].peRed  += 4 ;
        lp.palPalEntry[0].peGreen += 4 ;

        if (lp.palPalEntry[0].peRed == 255)
            bFadeIn = TRUE ;
    }

    AnimatePalette (hPalette, 0, 1, lp.palPalEntry) ;
    return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
```



```

{
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hPalette) ;
    return ;
}

```

FADER 在显示区域上显示满了文字字串「Fade In And Out」。文字首先显示为白色，这對於白色背景的视窗来说是看不出来的。通过使用调色盘动画，FADER 慢慢地将文字的颜色改为蓝色，然後再改回白色，这样一遍一遍地重复。文字就有渐现渐隐的显示效果了。

FADER 用 CreateRoutine 函式建立了逻辑调色盘，它只需要一个调色盘项目，并将颜色初始化为白色——红色、绿色和蓝色值都设为 255。在 PaintRoutine 中（您可能想起，当逻辑调色盘选进装置内容并显现以後，PALANIM 呼叫过此函式），FADER 呼叫 SetTextColor 将文字颜色设定为 PALETTEINDEX(0)。这意味著文字颜色设定为调色盘表格中的第一个项目，此项目初始为白色。然後 FADER 用「Fade In And Out」文字字串填充显示区域。这时，视窗背景是白色，文字也是白色，所以文字不可见。

在 TimerRoutine 函式中，FADER 通过改变 PALETTEENTRY 结构并将其传递给 AnimatePalette 来完成调色盘动画。最初，对每一个 WM_TIMER 讯息，程式都将红色和绿色值减 4，直到等於 3；然後将这些值加 4，直到等於 255。这将使文字颜色逐渐从白色变到蓝色，然後又回到白色。

程式 16-9 所示的 ALLCOLOR 程式只用了逻辑调色盘的一个项目来显示显示卡可以著色的所有颜色。当然，程式不是同时显示这些颜色，而是连续显示。如果显示卡有 18 位元的解析度（这时能有 262144 种不同的颜色），那么在两种颜色间隔 55 毫秒的速度下，只需要 4 小时就可以在萤幕上看到所有的颜色。

程式 16-9 ALLCOLOR

```

ALLCOLOR.C
/*-----
-
    ALLCOLOR.C --          Palette Animation Demo
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#define ID_TIMER    1

TCHAR szAppName    [] = TEXT ("AllColor") ;
TCHAR szTitle      [] = TEXT ("AllColor: Palette Animation Demo") ;

static      int                                iIncr ;

```

```

static      PALETTEENTRY      pe ;

HPALETTE CreateRoutine (HWND hwnd)
{
    HDC          hdc ;
    HPALETTE      hPalette ;
    LOGPALETTE    lp ;

    // Determine the color resolution and set iIncr
    hdc = GetDC (hwnd) ;
    iIncr = 1 << (8 - GetDeviceCaps (hdc, COLORRES) / 3) ;
    ReleaseDC (hwnd, hdc) ;

    // Create the logical palette
    lp.palVersion          = 0x0300 ;
    lp.palNumEntries        = 1 ;
    lp.palPalEntry[0].peRed    = 0 ;
    lp.palPalEntry[0].peGreen  = 0 ;
    lp.palPalEntry[0].peBlue   = 0 ;
    lp.palPalEntry[0].peFlags  = PC_RESERVED ;

    hPalette = CreatePalette (&lp) ;
    // Save global for less typing
    pe = lp.palPalEntry[0] ;
    SetTimer (hwnd, ID_TIMER, 10, NULL) ;
    return hPalette ;
}

void DisplayRGB (HDC hdc, PALETTEENTRY * ppe)
{
    TCHAR szBuffer [16] ;
    wsprintf (szBuffer, TEXT (" %02X-%02X-%02X "),
                ppe->peRed,                ppe->peGreen,
ppe->peBlue) ;
    TextOut (hdc, 0, 0, szBuffer, lstrlen (szBuffer)) ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
    HBRUSH      hBrush ;
    RECT        rect ;

    // Draw Palette Index 0 on entire window

    hBrush = CreateSolidBrush (PALETTEINDEX (0)) ;
    SetRect (&rect, 0, 0, cxClient, cyClient) ;
    FillRect (hdc, &rect, hBrush) ;
    DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
}

```

```

        // Display the RGB value
        DisplayRGB (hdc, &pe) ;
        return ;
    }

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
    static BOOL bRedUp = TRUE, bGreenUp = TRUE, bBlueUp = TRUE ;
        // Define new color value
    pe.peBlue += (bBlueUp ? iIncr : -iIncr) ;
    if ( pe.peBlue == (BYTE) (bBlueUp ? 0 : 256 - iIncr))
    {
        pe.peBlue = (bBlueUp ? 256 - iIncr : 0) ;
        bBlueUp ^= TRUE ;
        pe.peGreen += (bGreenUp ? iIncr : -iIncr) ;

        if ( pe.peGreen == (BYTE) (bGreenUp ? 0 : 256 - iIncr))
        {
            pe.peGreen = (bGreenUp ? 256 - iIncr : 0) ;
            bGreenUp ^= TRUE ;
            pe.peRed += (bRedUp ? iIncr : -iIncr) ;

            if ( pe.peRed == (BYTE) (bRedUp ? 0 :
256 - iIncr))
            {
                pe.peRed = (bRedUp ? 256 - iIncr : 0) ;
                bRedUp ^= TRUE ;
            }
        }
    }

    // Animate the palette
    AnimatePalette (hPalette, 0, 1, &pe) ;
    DisplayRGB (hdc, &pe) ;
    return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hPalette) ;
    return ;
}

```

在结构上, ALLCOLOR 与 FADER 非常相似。在 CreateRoutine 中, ALLCOLOR 只用一个设为黑色的调色盘项目 (PALETTEENTRY 结构的 red、green 和 blue 栏位设为 0) 来建立调色盘。在 PaintRoutine 中, ALLCOLOR 用 PALETTEINDEX(0)

建立实心画刷，并呼叫 FillRect 来用此画刷为整个显示区域著色。

在 TimerRoutine 中，ALLCOLOR 通过改变 PALETTEENTRY 颜色并呼叫 AnimatePalette 来启动调色盘。我编写 ALLCOLOR 程式，以便颜色变化顺畅。首先，蓝色值渐渐增加。达到最大时，绿色值增加，而蓝色值渐渐减少。红色、绿色和蓝色值的增加和减少取决于 iIncr 变数。在 CreateRoutine 期间，这将根据用 COLORRES 参数从 GetDeviceCaps 传回的值来计算。例如，如果 GetDeviceCaps 传回 18，那么 iIncr 设为 4——获得所有颜色所需要的最小值。

ALLCOLOR 还在显示区域的左上角显示目前的 RGB 颜色值。我最初添加这个程式码是出于测试目的，但是现在证明它是有用的，所以我保留了它。

工程应用程式

在工程应用程式中，动画对于显示机械或电的作用过程很有用。在电脑萤幕上显示内燃引擎虽然简单，但是动画可以使它变得更加生动，且更清楚地显示其工作程序。

使用调色盘动画的一个好范例就是显示流体通过管子的过程。这是一个例子，图像不必十分精确——实际上，如果图像很精确（就像看透明的管子），则很难说明管子里的流体是如何运动的。这时用符号会更好一些。程式 16-10 所示的 PIPES 程式是此技术的简单示范：在显示区域有两个水平的管子，流体在上面的管子里从左向右流动，而在下面的管子里从右向左移动。

程式 16-10 PIPES

PIPES.C

```
/*-----
    PIPES.C --          Palette Animation Demo
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Pipes") ;
TCHAR szTitle   [] = TEXT ("Pipes: Palette Animation Demo") ;

static LOGPALETTE * plp ;

HPALETTE CreateRoutine (HWND hwnd)
{
    HPALETTE      hPalette ;
    int           i ;
```

```

    plp = malloc (sizeof (LOGPALETTE) + 32 * sizeof (PALETTEENTRY)) ;
                                // Initialize the fields of the LOGPALETTE structure
    plp->palVersion      = 0x300 ;
    plp->palNumEntries   = 16 ;

    for (i = 0 ; i <= 8 ; i++)
    {
        plp->palPalEntry[i].peRed      = (BYTE) min (255, 0x20 * i) ;
        plp->palPalEntry[i].peGreen   = 0 ;
        plp->palPalEntry[i].peBlue    = (BYTE) min (255, 0x20 * i) ;
        plp->palPalEntry[i].peFlags   = PC_RESERVED ;

        plp->palPalEntry[16 - i]      = plp->palPalEntry[i] ;
        plp->palPalEntry[16 + i]      = plp->palPalEntry[i] ;
        plp->palPalEntry[32 - i]      = plp->palPalEntry[i] ;
    }

    hPalette = CreatePalette (plp) ;
    SetTimer (hwnd, ID_TIMER, 100, NULL) ;
    return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
    HBRUSH      hBrush ;
    int          i ;
    RECT         rect ;

                                // Draw window background

    SetRect (&rect, 0, 0, cxClient, cyClient) ;
    hBrush = SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;
    FillRect (hdc, &rect, hBrush) ;

                                // Draw the interiors of the pipes
    for (i = 0 ; i < 128 ; i++)
    {
        hBrush = CreateSolidBrush (PALETTEINDEX (i % 16)) ;
        SelectObject (hdc, hBrush) ;

        rect.left      = (127 - i) * cxClient / 128 ;
        zrect.right    = (128 - i) * cxClient / 128 ;
        rect.top       = 4 * cyClient / 14 ;
        rect.bottom    = 5 * cyClient / 14 ;

        FillRect (hdc, &rect, hBrush) ;

        rect.left      = i * cxClient / 128 ;

```

```

        rect.right          = ( i + 1 )      * cxClient /
128 ;

        rect.top            =      9      * cyClient / 14 ;
        rect.bottom         =      10     * cyClient / 14 ;

        FillRect (hdc, &rect, hBrush) ;

        DeleteObject      (SelectObject      (hdc,      GetStockObject
(WHITE_BRUSH))) ;
    }

        // Draw the edges of the pipes
    MoveToEx      (hdc, 0,  4 * cyClient / 14, NULL) ;
    LineTo        (hdc, cxClient,  4      * cyClient / 14) ;

    MoveToEx      (hdc, 0,  5 * cyClient / 14, NULL) ;
    LineTo        (hdc, cxClient,  5      * cyClient / 14) ;

    MoveToEx      (hdc, 0,  9 * cyClient / 14, NULL) ;
    LineTo        (hdc, cxClient,  9 * cyClient / 14) ;

    MoveToEx      (hdc, 0, 10 * cyClient / 14, NULL) ;
    LineTo        (hdc, cxClient,      10 * cyClient / 14) ;
    return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
    static int iIndex ;
    AnimatePalette (hPalette, 0, 16, plp->palPalEntry + iIndex) ;
    iIndex = (iIndex + 1) % 16 ;

    return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hPalette) ;
    free (plp) ;
    return ;
}

```

PIPES 为动画使用了 16 个调色盘项目，而您可能会使用更少的项目。最小化时，真正需要的是有足够的项目来显示流动的方向。用三个调色盘项目要比用一个静态箭头好。

程式 16-11 所示的 TUNNEL 程式是这组程式中最贪心的程式，它为动画使用

了 128 个调色盘项目，但是从效果来看，值得这样做。

程式 16-11 TUNNEL

```
TUNNEL.C
/*-----
-
TUNNEL.C --      Palette Animation Demo
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Tunnel") ;
TCHAR szTitle [] = TEXT ("Tunnel: Palette Animation Demo") ;

static LOGPALETTE * plp ;
HPALETTE CreateRoutine (HWND hwnd)
{
    BYTE          byGrayLevel ;
    HPALETTE       hPalette ;
    int            i ;

    plp = malloc (sizeof (LOGPALETTE) + 255 * sizeof (PALETTEENTRY)) ;
                // Initialize the fields of the LOGPALETTE structure
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries   = 128 ;

    for (i = 0 ; i < 128 ; i++)
    {
        if (i < 64)
            byGrayLevel = (BYTE) (4 * i) ;
        else
            byGrayLevel = (BYTE) min (255, 4 * (128
- i)) ;

        plp->palPalEntry[i].peRed      = byGrayLevel ;
        plp->palPalEntry[i].peGreen   = byGrayLevel ;
        plp->palPalEntry[i].peBlue    = byGrayLevel ;
        plp->palPalEntry[i].peFlags   = PC_RESERVED ;

        plp->palPalEntry[i + 128].peRed      = byGrayLevel ;
        plp->palPalEntry[i + 128].peGreen   = byGrayLevel ;
        plp->palPalEntry[i + 128].peBlue    = byGrayLevel ;
        plp->palPalEntry[i + 128].peFlags   = PC_RESERVED ;
    }

    hPalette = CreatePalette (plp) ;
    SetTimer (hwnd, ID_TIMER, 50, NULL) ;
}
```

```

        return hPalette ;
    }

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
    HBRUSH      hBrush ;
    int          i ;
    RECT         rect ;

    for (i = 0 ; i < 127 ; i++)
    {
        // Use a RECT structure for each of 128
rectangles
        rect.left  =  i * cxClient / 255 ;
        rect.top   =  i * cyClient / 255 ;
        rect.right = cxClient - i * cxClient / 255 ;
        rect.bottom = cyClient - i * cyClient / 255 ;

        hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;
        // Fill the rectangle and delete the brush

        FillRect (hdc, &rect, hBrush) ;
        DeleteObject (hBrush) ;
    }
    return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
    static int iLevel ;
    iLevel = (iLevel + 1) % 128 ;
    AnimatePalette (hPalette, 0, 128, plp->palPalEntry + iLevel) ;
    return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hPalette) ;
    free (plp) ;
    return ;
}

```

TUNNEL 在 128 个调色盘项目中使用 64 种移动的灰阶——从黑到白, 再从白到黑——表现在隧道旅行的效果。

调色盘和真实世界图像

当然，尽管我们已经完成了许多有趣的事：连续显示色彩的网底、做了调色盘动画，但调色盘管理器的真正目的是允许在 8 位元显示模式下显示真实世界中的图像。对于本章的其余部分，我们正好研究一下。正如您所期望的，在使用 packed DIB、GDI 点阵图物件和 DIB 区块时，必须按照不同的方法来使用调色盘。下面的六个程式阐明了用调色盘来处理点阵图的各种技术。

调色盘和 packed DIB

下面三个程式，有助于我们建立处理 packed DIB 记忆体块的一系列函式。这些函式都在程式 16-12 所示的 PACKEDIB 档案中。

程式 16-12 PACKEDIB 档案

```
PACKEDIB.H
/*-----
   PACKEDIB.H -- Header file for PACKEDIB.C
                                           (c) Charles Petzold, 1998
   -----*/

#include <windows.h>

BITMAPINFO * PackedDibLoad (PTSTR szFileName) ;
int PackedDibGetWidth (BITMAPINFO * pPackedDib) ;
int PackedDibGetHeight (BITMAPINFO * pPackedDib) ;
int PackedDibGetBitCount (BITMAPINFO * pPackedDib) ;
int PackedDibGetRowLength (BITMAPINFO * pPackedDib) ;
int PackedDibGetInfoHeaderSize (BITMAPINFO * pPackedDib) ;
int PackedDibGetColorsUsed (BITMAPINFO * pPackedDib) ;
int PackedDibGetNumColors (BITMAPINFO * pPackedDib) ;
int PackedDibGetColorTableSize (BITMAPINFO * pPackedDib) ;
RGBQUAD * PackedDibGetColorTablePtr (BITMAPINFO * pPackedDib) ;
RGBQUAD * PackedDibGetColorTableEntry (BITMAPINFO * pPackedDib, int i) ;
BYTE * PackedDibGetBitsPtr (BITMAPINFO * pPackedDib) ;
int PackedDibGetBitsSize (BITMAPINFO * pPackedDib) ;
HPALETTE PackedDibCreatePalette (BITMAPINFO * pPackedDib) ;
PACKEDIB.C
/*-----
   PACKEDIB.C -- Routines for using packed DIBs
                                           (c) Charles Petzold, 1998
   -----*/

#include <windows.h>

/*-----
```

```

-
    PackedDibLoad: Load DIB File as Packed-Dib Memory Block
-----
*/

BITMAPINFO * PackedDibLoad (PTSTR szFileName)
{
    BITMAPFILEHEADER    bmfh ;
    BITMAPINFO           *    pbmi ;
    BOOL                bSuccess ;
    DWORD               dwPackedDibSize, dwBytesRead ;
    HANDLE              hFile ;

    // Open the file: read access, prohibit write access

    hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
    if (hFile == INVALID_HANDLE_VALUE)
        return NULL ;
    // Read in the BITMAPFILEHEADER
    bSuccess = ReadFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
        &dwBytesRead, NULL) ;

    if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
        || (bmfh.bfType != * (WORD *) "BM"))
    {
        CloseHandle (hFile) ;
        return NULL ;
    }

    // Allocate memory for the packed DIB & read it in
    dwPackedDibSize = bmfh.bfSize - sizeof (BITMAPFILEHEADER) ;
    pbmi = malloc (dwPackedDibSize) ;
    bSuccess = ReadFile (hFile, pbmi, dwPackedDibSize, &dwBytesRead, NULL) ;
    CloseHandle (hFile) ;

    if (!bSuccess || (dwBytesRead != dwPackedDibSize))
    {
        free (pbmi) ;
        return NULL ;
    }

    return pbmi ;
}

/*-----
    Functions to get information from packed DIB
-----
*/

```

```

int PackedDibGetWidth (BITMAPINFO * pPackedDib)
{
    if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return
((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcWidth ;
    else
        return pPackedDib->bmiHeader.biWidth ;
}

int PackedDibGetHeight (BITMAPINFO * pPackedDib)
{
    if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return
((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcHeight ;
    else
        return abs (pPackedDib->bmiHeader.biHeight) ;
}

int PackedDibGetBitCount (BITMAPINFO * pPackedDib)
{
    if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return
((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcBitCount ;
    else
        return pPackedDib->bmiHeader.biBitCount ;
}

int PackedDibGetRowLength (BITMAPINFO * pPackedDib)
{
    return (( PackedDibGetWidth (pPackedDib) *
                PackedDibGetBitCount (pPackedDib) + 31)
& ~31) >> 3 ;
}

/*-----
-
    PackedDibGetInfoHeaderSize includes possible color masks!
-----
*/

int PackedDibGetInfoHeaderSize (BITMAPINFO * pPackedDib)
{
    if ( pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return
((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcSize ;

    else if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPINFOHEADER))

```

```

        return pPackedDib->bmiHeader.biSize +
            (pPackedDib->bmiHeader.biCompression ==
             BI_BITFIELDS ? 12 : 0) ;
    else return pPackedDib->bmiHeader.biSize ;
}

/*-----
    PackedDibGetColorsUsed returns value in information header;
    could be 0 to indicate non-truncated color table!
-----*/

int PackedDibGetColorsUsed (BITMAPINFO * pPackedDib)
{
    if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return 0 ;
    else
        return pPackedDib->bmiHeader.biClrUsed ;
}

/*-----
    PackedDibGetNumColors is actual number of entries in color table
-----*/

int PackedDibGetNumColors (BITMAPINFO * pPackedDib)
{
    int iNumColors ;
    iNumColors = PackedDibGetColorsUsed (pPackedDib) ;
    if ( iNumColors == 0 && PackedDibGetBitCount (pPackedDib) < 16)
        iNumColors = 1 << PackedDibGetBitCount (pPackedDib) ;

    return iNumColors ;
}

int PackedDibGetColorTableSize (BITMAPINFO * pPackedDib)
{
    if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return PackedDibGetNumColors (pPackedDib) * sizeof
(RGBTRIPLE) ;
    else
        return PackedDibGetNumColors (pPackedDib) * sizeof
(RGBQUAD) ;
}

RGBQUAD * PackedDibGetColorTablePtr (BITMAPINFO * pPackedDib)
{

```

```

        if (PackedDibGetNumColors (pPackedDib) == 0)
            return 0 ;
        return (RGBQUAD *) (((BYTE *)      pPackedDib) +
                               PackedDibGetInfoHeaderSize (pPackedDib)) ;
    }

RGBQUAD * PackedDibGetColorTableEntry (BITMAPINFO * pPackedDib, int i)
{
    if      (      PackedDibGetNumColors (pPackedDib) == 0)
        return 0 ;

    if      (      pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
        return (RGBQUAD *)
                (((RGBTRIPLE *)  PackedDibGetColorTablePtr
(pPackedDib)) + i) ;
    else
        return PackedDibGetColorTablePtr (pPackedDib) + i ;
}

/*-----
   PackedDibGetBitsPtr finally!
-----*/

BYTE * PackedDibGetBitsPtr (BITMAPINFO * pPackedDib)
{
    return ((BYTE *) pPackedDib)+      PackedDibGetInfoHeaderSize
(pPackedDib) +
                PackedDibGetColorTableSize (pPackedDib) ;
}

/*-----
   ---
   PackedDibGetBitsSize can be calculated from the height and row length
   if it's not explicitly in the biSizeImage field
-----*/

int PackedDibGetBitsSize (BITMAPINFO * pPackedDib)
{
    if ((pPackedDib->bmiHeader.biSize != sizeof (BITMAPCOREHEADER)) &&
        (pPackedDib->bmiHeader.biSizeImage != 0))
        return pPackedDib->bmiHeader.biSizeImage ;

    return      PackedDibGetHeight (pPackedDib) *
                PackedDibGetRowLength (pPackedDib) ;
}

```

```

/*-----
-
PackedDibCreatePalette creates logical palette from PackedDib
-----
*/
HPALETTE PackedDibCreatePalette (BITMAPINFO * pPackedDib)
{
    HPALETTE    hPalette ;
    int         i, iNumColors ;
    LOGPALETTE *   plp ;
    RGBQUAD     *   prgb ;

    if (0 == ( iNumColors = PackedDibGetNumColors (pPackedDib)))
        return NULL ;
    plp = malloc (sizeof (LOGPALETTE) *
        (iNumColors - 1) * sizeof (PALETTEENTRY)) ;

    plp->palVersion    = 0x0300 ;
    plp->palNumEntries = iNumColors ;
    for (i = 0 ; i < iNumColors ; i++)
    {
        prgb = PackedDibGetColorTableEntry (pPackedDib, i) ;
        plp->palPalEntry[i].peRed    = prgb->rgbRed ;
        plp->palPalEntry[i].peGreen  = prgb->rgbGreen ;
        plp->palPalEntry[i].peBlue   = prgb->rgbBlue ;
        plp->palPalEntry[i].peFlags  = 0 ;
    }

    hPalette = CreatePalette (plp) ;
    free (plp) ;

    return hPalette ;
}

```

第一个函式是 PackedDibLoad，它将唯一的参数作为档案名，并传回指向记忆体中 packed DIB 的指标。其他所有函式都将这个 packed DIB 指标作为它们的第一个参数并传回有关 DIB 的资讯。这些函式按「由下而上」顺序排列到档案中。每个函式都使用从前面函式获得的资讯。

我不倾向於说这是在处理 packed DIB 时有用的「完整」函式集。而且，我也不想汇编一个真正的扩展集，因为我不认为这是处理 packed DIB 的一个好方法。在写类似下面的函式时，您会很明显地发现这一点：

```
dwPixel = PackedDibGetPixel (pPackedDib, x, y) ;
```

这种函式包括太多的巢状函式呼叫，以致於效率非常低而且很慢。本章的後面将讨论一种我认为更好的方法。

另外，您将注意到，其中许多函式都需要对 OS/2 相容的 DIB 采取不同的处

理程序；这样，函式将频繁地检查 BITMAPINFO 结构的第一个栏位是否与 BITMAPCOREHEADER 结构的大小相同。

特别注意最後一个函式 PackedDibCreatePalette。这个函式用 DIB 中的颜色表来建立调色盘。如果 DIB 中没有颜色表（这意味著 DIB 的每图素有 16、24 或 32 位元），那么就不建立调色盘。我们有时会将从 DIB 颜色表建立的调色盘称为 DIB **自己的** 调色盘。

PACKEDIB 档案都放在 SHOWDIB3，如程式 16-13 所示。

程式 16-13 SHOWDIB3

```

SHOWDIB3.C
/*-----
    SHOWDIB3.C --          Displays DIB with native palette
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "PackedDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
        = szAppName ;
    wndclass.lpszClassName
        = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName, MB_ICONERROR) ;

        return 0 ;
    }
}

```

```

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #3: Native Palette"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static BITMAPINFO *    pPackedDib ;
    static HPALETTE        hPalette ;
    static int             cxClient, cyClient ;
    static OPENFILENAME    ofn ;
    static TCHAR           szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
    static TCHAR           szFilter[] =      TEXT      ("Bitmap      Files
(*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
    HDC                    hdc ;
    PAINTSTRUCT            ps ;

    switch (message)
    {
    case  WM_CREATE:
        ofn.lStructSize      = sizeof (OPENFILENAME) ;
        ofn.hwndOwner        = hwnd ;
        ofn.hInstance        = NULL ;
        ofn.lpstrFilter       = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter    = 0 ;
        ofn.nFilterIndex     = 0 ;
        ofn.lpstrFile         = szFileName ;
        ofn.nMaxFile         = MAX_PATH ;
        ofn.lpstrFileTitle    = szTitleName ;
        ofn.nMaxFileTitle    = MAX_PATH ;
        ofn.lpstrInitialDir   = NULL ;
        ofn.lpstrTitle        = NULL ;

```



```
        ofn.Flags                = 0 ;
        ofn.nFileOffset          = 0 ;
        ofn.nFileExtension       = 0 ;
        ofn.lpstrDefExt           = TEXT ("bmp") ;
        ofn.lCustData             = 0 ;
        ofn.lpfHook               = NULL ;
        ofn.lpTemplateName       = NULL ;

        return 0 ;

case WM_SIZE:

    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_COMMAND:

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

        // Show the File Open dialog box

        if (!GetOpenFileName (&ofn))
            return 0 ;

        // If there's an existing packed DIB, free the memory

        if (pPackedDib)
        {
            free (pPackedDib) ;
            pPackedDib = NULL ;
        }

        // If there's an existing logical palette, delete it

        if (hPalette)
        {
            DeleteObject (hPalette) ;
            hPalette = NULL ;
        }

        // Load the packed DIB into memory

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        pPackedDib = PackedDibLoad (szFileName) ;

        ShowCursor (FALSE) ;
```

```

        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        if (pPackedDib)
        {
            // Create the palette from the DIB color table

            hPalette = PackedDibCreatePalette (pPackedDib) ;
        }
        else
        {
            MessageBox (        hwnd, TEXT ("Cannot load DIB file"),
                szAppName, 0) ;
        }
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;
    }
    break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }

    if (pPackedDib)
        SetDIBitsToDevice      (hdc,          0,0,PackedDibGetWidth
(pPackedDib),
                                PackedDibGetHeight
(pPackedDib),
                                0,0,0,PackedDibGetHeight (pPackedDib),
                                PackedDibGetBitsPtr (pPackedDib),
                                pPackedDib,
                                DIB_RGB_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

```

```

        ReleaseDC (hwnd, hdc) ;
        return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    if (pPackedDib)
        free (pPackedDib) ;

    if (hPalette)
        DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

SHOWDIB3.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB3 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",          IDM_FILE_OPEN
    END
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib3.rc

#define IDM_FILE_OPEN          40001

```

SHOWDIB3 中的视窗讯息处理程式将 packed DIB 指标作为静态变数来维护，视窗讯息处理程式在「File Open」命令期间呼叫 PACKEDIB.C 中的 PackedDibLoad 函式时获得了此指标。在处理此命令的过程中，SHOWDIB3 也呼叫 PackedDibCreatePalette 来获得可能用於 DIB 的调色盘。注意，无论 SHOWDIB3 什么时候准备载入新的 DIB，都应先释放前一个 DIB 的记忆体，并删除前一个 DIB 的调色盘。在处理 WM_DESTROY 讯息的程序中，最後的 DIB 最後释放，最後的调色盘最後删除。

处理 WM_PAINT 讯息很简单：如果存在调色盘，则 SHOWDIB3 将它选进装置内容并显现它。然後它呼叫 SetDIBitsToDevice，并传递有关 DIB 的函式资讯（例如宽、高和指向 DIB 图素位元的指标），这些资讯从 PACKEDIB 中的函式获得。

另外，请记住 SHOWDIB3 依据 DIB 中的颜色表建立了调色盘。如果在 DIB 中没有颜色表——通常是 16 位元、24 位元和 32 位元 DIB 的情况——就不建立调色盘。在 8 位元显示模式下显示 DIB 时，它只能用标准保留的 20 种颜色显示。

对这个问题有两种解决方法：第一种是简单地使用「通用」调色盘，这种调色盘适用於许多图形。您也可以自己建立调色盘。第二种解决方法是分析 DIB 的图素位元，并决定要显示图像的最佳颜色。很明显，第二种方法将涉及更多的工作（对於程式写作者和处理器都是如此），但是我将在本章结束之前告诉您如何使用第二种方法。

「通用」调色盘

程式 16-14 所示的 SHOWDIB4 程式建立了一个通用的调色盘，它用於显示载入到程式中的所有 DIB。另外，SHOWDIB4 与 SHOWDIB3 非常相似。

程式 16-14 SHOWDIB4

```
SHOWDIB4.C
/*-----
-
    SHOWDIB4.C --      Displays DIB with "all-purpose" palette
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "..\ShowDib3\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib4") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
```

```

{
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS            wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = szAppName ;
    wndclass.lpszClassName = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #4: All-Purpose
Palette"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

/*-----
---
CreateAllPurposePalette: Creates a palette suitable for a wide variety
of images; the palette has 247 entries, but 15 of them are
duplicates or match the standard 20 colors.
-----

```

```

-*/

HPALETTE CreateAllPurposePalette (void)
{
    HPALETTE hPalette ;
    int      i, incr, R, G, B ;
    LOGPALETTE *   plp ;

    plp = malloc (sizeof (LOGPALETTE) + 246 * sizeof (PALETTEENTRY)) ;
    plp->palVersion = 0x0300 ;
    plp->palNumEntries = 247 ;

    // The following loop calculates 31 gray shades, but 3 of them
    // will match the standard 20 colors

    for (i = 0, G = 0, incr = 8 ; G <= 0xFF ; i++, G += incr)
    {
        plp->palPalEntry[i].peRed    = (BYTE) G ;
        plp->palPalEntry[i].peGreen  = (BYTE) G ;
        plp->palPalEntry[i].peBlue   = (BYTE) G ;
        plp->palPalEntry[i].peFlags  = 0 ;

        incr = (incr == 9 ? 8 : 9) ;
    }

    // The following loop is responsible for 216 entries, but 8 of
    // them will match the standard 20 colors, and another
    // 4 of them will match the gray shades above.

    for (R = 0 ; R <= 0xFF ; R += 0x33)
    for (G = 0 ; G <= 0xFF ; G += 0x33)
    for (B = 0 ; B <= 0xFF ; B += 0x33)
    {
        plp->palPalEntry [i].peRed      = (BYTE) R ;
        plp->palPalEntry [i].peGreen    = (BYTE) G ;
        plp->palPalEntry [i].peBlue     = (BYTE) B ;
        plp->palPalEntry [i].peFlags    = 0 ;

        i++ ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{

```

```

static      BITMAPINFO *      pPackedDib ;
static      HPALETTE          hPalette ;
static      int                cxClient, cyClient ;
static      OPENFILENAME      ofn ;
static      TCHAR              szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
static      TCHAR              szFilter[] =      TEXT      ("Bitmap      Files
(*.BMP)\0*.bmp\0")
      TEXT ("All Files (*.*)\0*.*\0\0") ;
HDC
PAINTSTRUCT                                hdc ;
switch (message)
{
case WM_CREATE:
      ofn.lStructSize      = sizeof (OPENFILENAME) ;
      ofn.hwndOwner        = hwnd ;
      ofn.hInstance        = NULL ;
      ofn.lpstrFilter      = szFilter ;
      ofn.lpstrCustomFilter = NULL ;
      ofn.nMaxCustFilter   = 0 ;
      ofn.nFilterIndex     = 0 ;
      ofn.lpstrFile        = szFileName ;
      ofn.nMaxFile         = MAX_PATH ;
      ofn.lpstrFileTitle   = szTitleName ;
      ofn.nMaxFileTitle    = MAX_PATH ;
      ofn.lpstrInitialDir  = NULL ;
      ofn.lpstrTitle       = NULL ;
      ofn.Flags            = 0 ;
      ofn.nFileOffset      = 0 ;
      ofn.nFileExtension   = 0 ;
      ofn.lpstrDefExt      = TEXT ("bmp") ;
      ofn.lCustData        = 0 ;
      ofn.lpfnHook         = NULL ;
      ofn.lpTemplateName   = NULL ;

      // Create the All-Purpose Palette

      hPalette = CreateAllPurposePalette () ;
      return 0 ;

case WM_SIZE:
      cxClient = LOWORD (lParam) ;
      cyClient = HIWORD (lParam) ;
      return 0 ;

case WM_COMMAND:
      switch (LOWORD (wParam))
      {
case IDM_FILE_OPEN:

```

```

// Show the File Open dialog box

if (!GetOpenFileName (&ofn))
    return 0 ;

// If there's an existing packed DIB, free the memory
if (pPackedDib)
{
    free (pPackedDib) ;
    pPackedDib = NULL ;
}

// Load the packed DIB into memory

SetCursor      (LoadCursor      (NULL,
IDC_WAIT)) ;

ShowCursor (TRUE) ;

pPackedDib      =      PackedDibLoad
(szFileName) ;

ShowCursor (FALSE) ;
SetCursor      (LoadCursor      (NULL,
IDC_ARROW)) ;

if (!pPackedDib)
{
    MessageBox (    hwnd, TEXT ("Cannot load DIB file"),
szAppName, 0) ;
}
InvalidateRect (hwnd, NULL, TRUE) ;
return 0 ;
}
break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (pPackedDib)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;

        SetDIBitsToDevice      (hdc, 0, 0, PackedDibGetWidth
(pPackedDib),
PackedDibGetHeight (pPackedDib),

```



```

                                0,0,0,PackedDibGetHeight (pPackedDib),
                                PackedDibGetBitsPtr (pPackedDib),
                                pPackedDib,
        DIB_RGB_COLORS) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if ((HWND) wParam != hwnd)

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        UpdateColors (hdc) ;

        ReleaseDC (hwnd, hdc) ;
        break ;

case WM_DESTROY:
    if (pPackedDib)
        free (pPackedDib) ;

    DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

SHOWDIB4.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////
/
// Menu
SHOWDIB4 MENU DISCARDABLE
BEGIN

```

```

    POPUP "&Open"
    BEGIN
        MENUITEM "&File",                IDM_FILE_OPEN
    END
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib4.rc

#define IDM_FILE_OPEN                40001

```

在处理 WM_CREATE 讯息时，SHOWDIB4 将呼叫 CreateAllPurposePalette，并在程式中保留该调色盘，而在 WM_DESTROY 讯息处理期间删除它。因为程式知道调色盘一定存在，所以在处理 WM_PAINT、WM_QUERYNEWPALETTE 或 WM_PALETTECHANGED 讯息时，不必检查调色盘的存在。

CreateAllPurposePalette 函式似乎是用 247 个项目来建立逻辑调色盘，它超出了系统调色盘中允许程式正常存取的 236 个项目。的确如此，不过这样做很方便。这些项目中有 15 个被复制或者映射到 20 种标准的保留颜色中。

CreateAllPurposePalette 从建立 31 种灰阶开始，即 0x00、0x09、0x11、0x1A、0x22、0x2B、0x33、0x3C、0x44、0x4D、0x55、0x5E、0x66、0x6F、0x77、0x80、0x88、0x91、0x99、0xA2、0xAA、0xB3、0xBB、0xC4、0xCC、0xD5、0xDD、0xE6、0xEE、0xF9 和 0xFF 的红色、绿色和蓝色值。注意，第一个、最后一个和中间的项目都在标准的 20 种保留颜色中。下一个函式用红色、绿色和蓝色值的所有组合建立了颜色 0x00、0x33、0x66、0x99、0xCC 和 0xFF。这样就共有 216 种颜色，但是其中 8 种颜色复制了标准的 20 种保留颜色，而另外 4 个复制了前面计算的灰阶。如果将 PALETTEENTRY 结构的 peFlags 栏位设为 0，则 Windows 将不把复制的项目放进系统调色盘。

显然地，实际的程式不希望计算 16 位元、24 位元或者 32 位元 DIB 的最佳调色盘，程式将继续使用 DIB 颜色表来显示 8 位元 DIB。SHOWDIB4 不完成这项工作，它只对每件事都使用通用调色盘。因为 SHOWDIB4 是一个展示程式，而且您可以与 SHOWDIB3 显示的 8 位元 DIB 进行比较。如果看一些人像的彩色 DIB，那么您可能会得出这样的结论：SHOWDIB4 没有足够的颜色来精确地表示鲜艳的色调。

如果用 SHOWDIB4 中的 CreateAllPurposePalette 函式来试验（可能是通过将逻辑调色盘的大小减少到只有几个项目的方法），您将发现当调色盘选进装置内容时，Windows 将只使用调色盘中的颜色，而不使用标准的 20 种颜色调色盘的颜色。

中间色调色盘

Windows API 包括一个通用调色盘，程式可以通过呼叫 CreateHalftonePalette 来获得该调色盘。使用此调色盘的方法与使用从 SHOWDIB4 中的 CreateAllPurposePalette 获得调色盘的方法相同，或者您也可以与点阵图缩放模式中的 HALFTONE 设定——用 SetStretchBltMode 设定——一起使用。程式 16-15 所示的 SHOWDIB5 程式展示了使用中间色调色盘的方法。

程式 16-15 SHOWDIB5

```
SHOWDIB5.C
/*-----
    SHOWDIB5.C --      Displays DIB with halftone palette
                        (c) Charles Petzold, 1998
    -----*/

#include <windows.h>
#include "..\ShowDib3\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib5") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = szAppName ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
MB_ICONERROR) ;
    }
}
```

```

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #5: Halftone Palette"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static      BITMAPINFO *      pPackedDib ;
    static      HPALETTE          hPalette ;
    static      int                cxClient, cyClient ;
    static      OPENFILENAME      ofn ;
    static      TCHAR              szFileName      [MAX_PATH],      szTitleName
[MAX_PATH] ;
    static      TCHAR              szFilter[] =      TEXT      ("Bitmap      Files
(*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*.*\0\0") ;
    HDC          hdc ;
    PAINTSTRUCT  ps ;

    switch (message)
    {
    case  WM_CREATE:
        ofn.lStructSize      = sizeof (OPENFILENAME) ;
        ofn.hwndOwner        = hwnd ;
        ofn.hInstance        = NULL ;
        ofn.lpstrFilter      = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter   = 0 ;
        ofn.nFilterIndex     = 0 ;
        ofn.lpstrFile        = szFileName ;
        ofn.nMaxFile         = MAX_PATH ;
        ofn.lpstrFileTitle   = szTitleName ;

```

```
        ofn.nMaxFileTitle           = MAX_PATH ;
        ofn.lpstrInitialDir         = NULL ;
        ofn.lpstrTitle              = NULL ;
        ofn.Flags                   = 0 ;
        ofn.nFileOffset             = 0 ;
        ofn.nFileExtension          = 0 ;
        ofn.lpstrDefExt              = TEXT ("bmp") ;
        ofn.lCustData                = 0 ;
        ofn.lpfnHook                 = NULL ;
        ofn.lpTemplateName          = NULL ;

        // Create the All-Purpose Palette

        hdc = GetDC (hwnd) ;
        hPalette = CreateHalftonePalette (hdc) ;
        ReleaseDC (hwnd, hdc) ;
        return 0 ;

case WM_SIZE:

        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_COMMAND:

        switch (LOWORD (wParam))
        {
        case IDM_FILE_OPEN:

                // Show the File Open dialog box

                if (!GetOpenFileName (&ofn))
                        return 0 ;

                // If there's an existing packed DIB, free the memory

                if (pPackedDib)
                {
                        free (pPackedDib) ;
                        pPackedDib = NULL ;
                }

                // Load the packed DIB into memory

                SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                ShowCursor (TRUE) ;

                pPackedDib = PackedDibLoad (szFileName) ;
```

```
        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        if (!pPackedDib)
        {
            MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
                szAppName, 0) ;
        }
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;
    }
    break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (pPackedDib)
    {
        // Set halftone stretch mode

        SetStretchBltMode (hdc, HALFTONE) ;
        SetBrushOrgEx (hdc, 0, 0, NULL) ;

        // Select and realize halftone palette

        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;

        // StretchDIBits rather than SetDIBitsToDevice
        StretchDIBits (  hdc,0,0,PackedDibGetWidth (pPackedDib),
            PackedDibGetHeight (pPackedDib),
            0,0,PackedDibGetWidth (pPackedDib),
            PackedDibGetHeight (pPackedDib),
            PackedDibGetBitsPtr (pPackedDib),
            pPackedDib,
            DIB_RGB_COLORS,
            SRCCOPY) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
```

```

        return TRUE ;

case WM_PALETTECHANGED:
    if ((HWND) wParam != hwnd)

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        UpdateColors (hdc) ;

        ReleaseDC (hwnd, hdc) ;
        break ;

case WM_DESTROY:
    if (pPackedDib)
        free (pPackedDib) ;

    DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

SHOWDIB5.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

///
/

// Menu

SHOWDIB5 MENU DISCARDABLE

BEGIN

POPUP "&Open"

BEGIN

MENUITEM "&File", IDM_FILE_OPEN

END

END

RESOURCE.H (摘录)

// Microsoft Developer Studio generated include file.

// Used by ShowDib5.rc

#define IDM_FILE_OPEN 40001

SHOWDIB5 程式类似於 SHOWDIB4, SHOWDIB4 中不使用 DIB 中的颜色表, 而使用适用於图像范围更大的调色盘。为此, SHOWDIB5 使用了由 Windows 支援的逻辑调色盘, 其代号可以从 CreateHalftonePalette 函式获得。

中间色调色盘并不比 SHOWDIB4 中的 CreateAllPurposePalette 函式所建立的调色盘更复杂。的确，如果只是拿来自用，结果是相似的。然而，如果您呼叫下面两个函式：

```
SetStretchBltMode (hdc, HALFTONE) ;
                        SetBrushOrgEx (hdc, x, y, NULL) ;
```

其中，x 和 y 是 DIB 左上角的装置座标，并且如果您用 StretchDIBits 而不是 SetDIBitsToDevice 来显示 DIB，那么结果会让您吃惊：颜色色调要比不设定点阵图缩放模式来使用 CreateAllPurposePalette 或者 CreateHalftonePalette 更精确。Windows 使用一种混色图案来处理中间色调色盘上的颜色，以使其更接近 8 位元显示卡上原始图像的颜色。与您所想像的一样，这样做的缺点是需要更多的处理时间。

索引调色盘颜色

现在开始处理 SetDIBitsToDevice、StretchDIBits、CreateDIBitmap、SetDIBits、GetDIBits 和 CreateDIBSection 的 fClrUse 参数。通常，您将这个参数设定为 DIB_RGB_COLORS(等於 0)。不过，您也能将它设定为 DIB_PAL_COLORS。在这种情况下，假定 BITMAPINFO 结构中的颜色表不包括 RGB 颜色值，而是包括逻辑调色盘中颜色项目的 16 位元索引。逻辑调色盘是作为第一个参数传递给函式的装置内容中目前选择的那个。实际上，在 CreateDIBSection 中，之所以需要指定一个非 NULL 的装置内容代号作为第一个参数，只是因为使用了 DIB_PAL_COLORS。

DIB_PAL_COLORS 能为您做些什么呢？它可能提高一些性能。考虑一下在 8 位元显示模式下呼叫 SetDIBitsToDevice 显示的 8 位元 DIB。Windows 首先必须在 DIB 颜色表的所有颜色中搜索与设备可用颜色最接近的颜色。然後设定一个小表，以便将 DIB 图素值映射到设备图素。也就是说，最多需要搜索 256 次最接近的颜色。但是如果 DIB 颜色表中含有从装置内容中选择颜色的逻辑调色盘项目索引，那么就可能跳过搜索。

除了使用调色盘索引以外，程式 16-16 所示的 SHOWDIB6 程式与 SHOWDIB3 相似。

程式 16-16 SHOWDIB6

```
SHOWDIB6.C
/*-----
-
    SHOWDIB6.C --      Display DIB with palette indices
                                (c) Charles Petzold, 1998
-----
*/
```



```

#include <windows.h>
#include "..\ShowDib3\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib6") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #6: Palette Indices"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

```

```

    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND  hwnd,   UINT  message,   WPARAM  wParam,LPARAM
lParam)
{
    static      BITMAPINFO *      pPackedDib ;
    static      HPALETTE          hPalette ;
    static      int               cxClient, cyClient ;
    static      OPENFILENAME      ofn ;
    static      TCHAR             szFileName      [MAX_PATH],      szTitleName
[MAX_PATH] ;
    static      TCHAR             szFilter[] =      TEXT      ("Bitmap      Files
(*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
    HDC          hdc ;
    int          i, iNumColors ;
    PAINTSTRUCT  ps ;
    WORD         *      pwIndex ;

    switch (message)
    {
    case  WM_CREATE:
        ofn.lStructSize      = sizeof (OPENFILENAME) ;
        ofn.hwndOwner        = hwnd ;
        ofn.hInstance        = NULL ;
        ofn.lpstrFilter      = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter   = 0 ;
        ofn.nFilterIndex     = 0 ;
        ofn.lpstrFile        = szFileName ;
        ofn.nMaxFile         = MAX_PATH ;
        ofn.lpstrFileTitle   = szTitleName ;
        ofn.nMaxFileTitle    = MAX_PATH ;
        ofn.lpstrInitialDir  = NULL ;
        ofn.lpstrTitle       = NULL ;
        ofn.Flags             = 0 ;
        ofn.nFileOffset      = 0 ;
        ofn.nFileExtension   = 0 ;
        ofn.lpstrDefExt      = TEXT ("bmp") ;
        ofn.lCustData         = 0 ;
        ofn.lpfnHook         = NULL ;
        ofn.lpTemplateName   = NULL ;

        return 0 ;

    case  WM_SIZE:

```

```
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
        case IDM_FILE_OPEN:

            // Show the File Open dialog box
            if (!GetOpenFileName (&ofn))
                return 0 ;

            // If there's an existing packed DIB, free the memory

            if (pPackedDib)
            {
                free (pPackedDib) ;
                pPackedDib = NULL ;
            }

            // If there's an existing logical palette, delete it

            if (hPalette)
            {
                DeleteObject (hPalette) ;
                hPalette = NULL ;
            }

            // Load the packed DIB into memory

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            pPackedDib = PackedDibLoad (szFileName) ;

            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            if (pPackedDib)
            {
                // Create the palette from the DIB color table

                hPalette = PackedDibCreatePalette (pPackedDib) ;

                // Replace DIB color table with indices

                if (hPalette)
```

```
        {
            iNumColors = PackedDibGetNumColors (pPackedDib) ;
            pwIndex = (WORD *)
                PackedDibGetColorTablePtr (pPackedDib) ;

            for (i = 0 ; i < iNumColors ; i++)
                pwIndex[i] = (WORD) i ;
        }
        else
        {
            MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
                szAppName, 0) ;
        }
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;
    }
    break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }

    if (pPackedDib)
        SetDIBitsToDevice (hdc, 0, 0,
            PackedDibGetWidth (pPackedDib),
            PackedDibGetHeight (pPackedDib),
            0, 0, 0, PackedDibGetHeight (pPackedDib),
            PackedDibGetBitsPtr (pPackedDib),
            pPackedDib,
            DIB_PAL_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
```

```

        ReleaseDC (hwnd, hdc) ;
        return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    if (pPackedDib)
        free (pPackedDib) ;

    if (hPalette)
        DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

SHOWDIB6.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB6 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",                                IDM_FILE_OPEN
    END
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib6.rc
//
#define IDM_FILE_OPEN            40001

```

SHOWDIB6 将 DIB 载入到记忆体并由此建立了调色盘以後, SHOWDIB6 简单地用以 0 开始的 WORD 索引替换了 DIB 颜色表中的颜色。PackedDibGetNumColors 函式将表示有多少种颜色, 而 PackedDibGetColorTablePtr 函式传回指向 DIB 颜色表起始位置的指标。

注意, 只有直接从 DIB 颜色表来建立调色盘时, 此技术才可行。如果使用通用调色盘, 则必须搜索最接近的颜色, 以获得放入 DIB 的索引。

如果要使用调色盘索引, 那么请在将 DIB 储存到磁片之前, 确实替换掉 DIB 中的颜色表。另外, 不要将包含调色盘索引的 DIB 放入剪贴簿。实际上, 在显示之前, 将调色盘索引放入 DIB, 然後将 RGB 颜色值放回, 会更安全一些。

调色盘和点阵图物件

程式 16-17 中的 SHOWDIB7 程式显示了如何使用与 DIB 相关联的调色盘, 这些 DIB 是使用 CreateDIBitmap 函式转换成 GDI 点阵图物件的。

程式 16-17 SHOWDIB7

```
SHOWDIB7.C
/*-----
    SHOWDIB7.C --          Shows DIB converted to DDB
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "..\ShowDib3\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib7") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style
                = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
                = WndProc ;
    wndclass.cbClsExtra
                = 0 ;
    wndclass.cbWndExtra
                = 0 ;
    wndclass.hInstance
                = hInstance ;
    wndclass.hIcon
                = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
                = LoadCursor (NULL, IDC_ARROW) ;
```

```

    wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName       = szAppName ;
    wndclass.lpszClassName      = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
        MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #7: Converted to DDB"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP          hBitmap ;
    static HPALETTE         hPalette ;
    static int              cxClient, cyClient ;
    static OPENFILENAME     ofn ;
    static TCHAR            szFileName      [MAX_PATH],      szTitleName
[MAX_PATH] ;
    static TCHAR            szFilter[] =      TEXT      ("Bitmap      Files
(*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*.*\0\0") ;
    BITMAP                  bitmap ;
    BITMAPINFO              * pPackedDib ;
    HDC                     hdc, hdcMem ;
    PAINTSTRUCT              ps ;

    switch (message)
    {

```

```
case WM_CREATE:
    ofn.lStructSize      = sizeof (OPENFILENAME) ;
    ofn.hwndOwner        = hwnd ;
    ofn.hInstance        = NULL ;
    ofn.lpstrFilter       = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter    = 0 ;
    ofn.nFilterIndex     = 0 ;
    ofn.lpstrFile         = szFileName ;
    ofn.nMaxFile         = MAX_PATH ;
    ofn.lpstrFileTitle    = szTitleName ;
    ofn.nMaxFileTitle    = MAX_PATH ;
    ofn.lpstrInitialDir   = NULL ;
    ofn.lpstrTitle        = NULL ;
    ofn.Flags             = 0 ;
    ofn.nFileOffset       = 0 ;
    ofn.nFileExtension    = 0 ;
    ofn.lpstrDefExt       = TEXT ("bmp") ;
    ofn.lCustData         = 0 ;
    ofn.lpfHook           = NULL ;
    ofn.lpTemplateName    = NULL ;

    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
        case IDM_FILE_OPEN:

            // Show the File Open dialog box

            if (!GetOpenFileName (&ofn))
                return 0 ;

            // If there's an existing packed DIB, free the memory

            if (hBitmap)
            {
                DeleteObject (hBitmap) ;
                hBitmap = NULL ;
            }

            // If there's an existing logical palette, delete it
```



```
    if (hPalette)
    {
        DeleteObject (hPalette) ;
        hPalette = NULL ;
    }

    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (pPackedDib)
    {
        // Create palette from the DIB and select it into DC

        hPalette = PackedDibCreatePalette (pPackedDib) ;

        hdc = GetDC (hwnd) ;

        if (hPalette)
        {
            SelectPalette (hdc, hPalette, FALSE) ;
            RealizePalette (hdc) ;
        }

        // Create the DDB from the DIB
        hBitmap = CreateDIBitmap(hdc, (PBITMAPINFOHEADER) pPackedDib,
            CBM_INIT, PackedDibGetBitsPtr (pPackedDib),
            pPackedDib, DIB_RGB_COLORS) ;
        ReleaseDC (hwnd, hdc) ;

        // Free the packed-DIB memory

        free (pPackedDib) ;
    }
    else
    {
        MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
            szAppName, 0) ;
    }

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
}
```

```
        break ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }
    if (hBitmap)
    {
        GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;
        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;
        BitBlt (hdc, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
                hdcMem, 0, 0, SRCCOPY) ;

        DeleteDC (hdcMem) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
```

```

        if (hBitmap)
            DeleteObject (hBitmap) ;

        if (hPalette)
            DeleteObject (hPalette) ;

        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

SHOWDIB7.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB7 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",                IDM_FILE_OPEN
    END
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib7.rc

#define IDM_FILE_OPEN    40001

```

与前面的程式一样，SHOWDIB7 获得了一个指向 packed DIB 的指标，该 DIB 回应功能表的「File」、「Open」命令。程式从 packed DIB 建立了调色盘，然後——还是在 WM_COMMAND 讯息的处理过程中——获得了用於视讯显示的装置内容，并选进调色盘，显现调色盘。然後 SHOWDIB7 呼叫 CreateDIBitmap 以便从 DIB 建立 DDB。如果调色盘没有选进装置内容并显现，那么 CreateDIBitmap 建立的 DDB 将不使用逻辑调色盘中的附加颜色。

呼叫 CreateDIBitmap 以後，该程式将释放 packed DIB 占用的记忆体空间。pPackedDib 变数不是静态变数。相反的，SHOWDIB7 按静态变数保留了点阵图代号 (hBitmap) 和逻辑调色盘代号 (hPalette)。

在 WM_PAINT 讯息处理期间，调色盘再次选进装置内容并显现。GetObject 函式可获得点阵图的宽度和高度。然後，程式通过建立相容的记忆体装置内容在显示区域显示点阵图，选进点阵图，并执行 BitBlt。显示 DDB 时所用的调色

盘，必须与从 CreateDIBitmap 呼叫建立时所用的一样。

如果将点阵图复制到剪贴簿，则最好使用 packed DIB 格式。然後 Windows 可以将点阵图物件提供给希望使用这些点阵图的程式。然而，如果需要将点阵图物件复制到剪贴簿，则首先要获得视讯装置内容并显现调色盘。这允许 Windows 依据目前的系统调色盘将 DDB 转换为 DIB。

调色盘和 DIB 区块

最後，程式 16-18 所示的 SHOWDIB8 说明了如何使用带有 DIB 区块的调色盘。

程式 16-18 SHOWDIB8

```
SHOWDIB8.C
/*-----
    SHOWDIB8.C --          Shows DIB converted to DIB section
                                (c) Charles Petzold, 1998
    -----*/
/

#include <windows.h>
#include "..\ShowDib3\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib8") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),

```

```

        szAppName, MB_ICONERROR) ;

        return 0 ;

    }

    hwnd = CreateWindow (  szAppName, TEXT ("Show DIB #8: DIB Section"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HBITMAP          hBitmap ;
    static HPALETTE         hPalette ;
    static int              cxClient, cyClient ;
    static OPENFILENAME      ofn ;
    static PBYTE            pBits ;
    static TCHAR            szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
    static TCHAR            szFilter[] =   TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*.*\0\0") ;
    BITMAP                  bitmap ;
    BITMAPINFO              *    pPackedDib ;
    HDC                     hdc, hdcMem ;
    PAINTSTRUCT             ps ;

    switch (message)
    {
    case  WM_CREATE:
        ofn.lStructSize      = sizeof (OPENFILENAME) ;
        ofn.hwndOwner        = hwnd ;
        ofn.hInstance        = NULL ;
        ofn.lpstrFilter      = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter   = 0 ;
        ofn.nFilterIndex     = 0 ;
        ofn.lpstrFile        = szFileName ;
        ofn.nMaxFile        = MAX_PATH ;
    }
}

```

```
        ofn.lpstrFileName      = szTitleName ;
        ofn.nMaxFileName      = MAX_PATH ;
        ofn.lpstrInitialDir    = NULL ;
        ofn.lpstrTitle         = NULL ;
        ofn.Flags               = 0 ;
        ofn.nFileOffset        = 0 ;
        ofn.nFileExtension     = 0 ;
        ofn.lpstrDefExt        = TEXT ("bmp") ;
        ofn.lCustData           = 0 ;
        ofn.lpfnHook            = NULL ;
        ofn.lpTemplateName     = NULL ;

        return 0 ;

case WM_SIZE:

    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_COMMAND:

    switch (LOWORD (wParam))
    {
        case IDM_FILE_OPEN:

            // Show the File Open dialog box

            if (!GetOpenFileName (&ofn))
                return 0 ;

            // If there's an existing packed DIB, free the memory

            if (hBitmap)
            {
                DeleteObject (hBitmap) ;
                hBitmap = NULL ;
            }

            // If there's an existing logical palette, delete it

            if (hPalette)
            {
                DeleteObject (hPalette) ;
                hPalette = NULL ;
            }

            // Load the packed DIB into memory

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;
```

```

        pPackedDib = PackedDibLoad (szFileName) ;
        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        if (pPackedDib)
        {
            // Create the DIB section from the DIB

hBitmap = CreateDIBSection (NULL,pPackedDib,DIB_RGB_COLORS,&pBits,NULL, 0) ;

            // Copy the bits

            CopyMemory (pBits, PackedDibGetBitsPtr (pPackedDib),
                PackedDibGetBitsSize (pPackedDib)) ;

            // Create palette from the DIB

            hPalette = PackedDibCreatePalette (pPackedDib) ;

            // Free the packed-DIB memory

                free (pPackedDib) ;
            }
            else
            {
                MessageBox (    hwnd, TEXT ("Cannot load DIB file"),
                    szAppName, 0) ;
            }
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;
        }
        break ;
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }
    if (hBitmap)
    {
        GetObject (hBitmap, sizeof (BITMAP),
&bitmap) ;

        hdcMem = CreateCompatibleDC (hdc) ;
        SelectObject (hdcMem, hBitmap) ;

```

```
        BitBlt (    hdc,    0, 0, bitmap.bmWidth, bitmap.bmHeight,
                  hdcMem, 0, 0, SRCCOPY) ;

        DeleteDC (hdcMem) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC          (hwnd) ;
    SelectPalette        (hdc, hPalette, FALSE) ;
    RealizePalette       (hdc) ;
    InvalidateRect       (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    if (hBitmap)
        DeleteObject (hBitmap) ;

    if (hPalette)
        DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

[SHOWDIB8.RC](#) (摘录)

//Microsoft Developer Studio generated resource script.


```

#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
SHOWDIB8 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",          IDM_FILE_OPEN
    END
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by ShowDib8.rc

#define IDM_FILE_OPEN          40001

```

在 SHOWDIB7 和 SHOWDIB8 中的 WM_PAINT 处理是一样的：两个程式都将点阵图代号 (hBitmap) 和逻辑调色盘代号 (hPalette) 作为静态变数。调色盘被选进装置内容并显现，点阵图的宽度和高度从 GetObject 函式获得，程式建立记忆体装置内容并选进点阵图，然後通过呼叫 BitBlt 将点阵图显示到显示区域。

两个程式之间最大的差别在於处理「File」、「Open」功能表命令的程序。在获得指向 packed DIB 的指标并建立了调色盘以後，SHOWDIB7 必须将调色盘选进视讯装置内容，并在呼叫 CreateDIBitmap 之前显现。SHOWDIB8 在获得 packed DIB 指标以後呼叫 CreateDIBSection。不必将调色盘选进装置内容，这是因为 CreateDIBSection 不将 DIB 转换成设备相关的格式。的确，CreateDIBSection 的第一个参数（即装置内容代号）的唯一用途在於您是否使用 DIB_PAL_COLORS 旗标。

呼叫 CreateDIBSection 以後，SHOWDIB8 将图素位元从 packed DIB 复制到从 CreateDIBSection 函式传回的记忆体位置，然後呼叫 PackedDibCreatePalette。尽管此函式便於程式使用，但是 SHOWDIB8 将依据从 GetDIBColorTable 函式传回的资讯建立调色盘。

DIB 处理程式库

就是现在——经过我们长时间地学习 GDI 点阵图物件、装置无关点阵图、DIB 区块和 Windows 调色盘管理器之後——我们才做好了开发一套有助於处理点阵图的函式的准备。

前面的 PACKEDIB 档案展示了一种可能的方法：记忆体中的 packed DIB 只

用指向它的指标表示。程式所的有关 DIB 的全部资讯都可以从存取表头资讯结构的函式获得。然而，实际上到「get pixel」和「set pixel」常式时，这种方法就会产生严重的执行问题。图像处理任务当然需要存取点阵图位元，并且这些函式也应该尽可能地快。

可能的 C++ 的解决方式中包括建立 DIB 类别，这时指向 packed DIB 的指标正好是一个成员变数。其他成员变数和成员函式有助於更快地执行获得和设定 DIB 中的图素的常式。不过，因为我在第一章已经指出，对于本书您只需要了解 C，使用 C++ 将是其他书的范围。

当然，用 C++ 能做的事情用 C 也能做。一个好的例子就是许多 Windows 函式都使用代号。除了将代号当作数值以外，应用程式对它还了解什么呢？程式知道代号引用特殊的函式物件，还知道函式用于处理现存的物件。显然，作业系统按某种方式用代号来引用物件的内部资讯。代号可以与结构指标一样简单。

例如，假设有一个函式集，这些函式都使用一个称为 HDIB 的代号。HDIB 是什么呢？它可能在某个表头档案中定义如下：

```
typedef void * HDIB ;
```

此定义用「不关您的事」回答了「HDIB 是什么」这个问题。

然而，实际上 HDIB 可能是结构指标，该结构不仅包括指向 packed DIB 的指标，还包括其他资讯：

```
typedef struct
{
    BITMAPINFO *      pPackedDib ;
    int               cx, cy, cBitsPerPixel, cBytesPerRow ;
    BYTE              * pBits ;
}
DIBSTRUCTURE, * PDIBSTRUCTURE ;
```

此结构的其他五个栏位包括从 packed DIB 中引出的资讯。当然，结构中这些值允许更快速地存取它们。不同的 DIB 程式库函式都可以处理这个结构，而不是 pPackedDib 指标。可以按下面的方法来执行 DibGetPixelPointer 函式：

```
BYTE * DibGetPixelPointer (HDIB hdib, int x, int y)
{
    PDIBSTRUCTURE pdib = hdib ;
    return pdib->pBits + y * pdib->cBytesPerRow +
           x * pdib->cBitsPerPixel / 8 ;
}
```

当然，这种方法可能要比 PACKEDIB.C 中执行「get pixel」常式快。

由于这种方法非常合理，所以我决定放弃 packed DIB，并改用处理 DIB 区块的 DIB 程式库。这实际上使我们对 packed DIB 的处理有更大的弹性（也就是说，能够在装置无关的方式下操纵 DIB 图素位元），而且在 Windows NT 下执行

时将更有效。

DIBSTRUCT 结构

DIBHELP.C 档案——如此命名是因为对处理 DIB 提供帮助——有上千行，并在几个小部分中显示。但是首先让我们看一下 DIBHELP 函式所处理的结构，该结构在 DIBHELP.C 中定义如下：

```
typedef struct
{
    PBYTE      * ppRow ;          // array of row pointers
    int         iSignature ;      // = "Dib "
    HBITMAP     hBitmap ;         // handle returned from CreateDIBSection
    BYTE        * pBits ;         // pointer to bitmap bits
    DIBSECTION  ds ;              // DIBSECTION structure
    int         iRShift[3] ;      // right-shift values for color masks
    int         iLShift[3] ;      // left-shift values for color masks
}
DIBSTRUCT, * PDIBSTRUCT ;
```

现在跳过第一个栏位。它之所以为第一个栏位是因为它使某些巨集更易於使用——在讨论完其他栏位以後再来理解第一个栏位就更容易了。

在 DIBHELP.C 中，当 DIB 建立的函式首先设定了此结构时，第二个栏位就设定为文字字串「Dib」的二进位值。通过一些 DIBHELP 函式，第二个栏位将用於结构有效指标的一个标记。

第三个栏位，即 hBitmap，是从 CreateDIBSection 函式传回的点阵图代号。您将想起该代号可有多种使用方式，它与我们在第十四章遇到的 GDI 点阵图物件的代号用法一样。不过，从 CreateDIBSection 传回的代号将涉及按装置无关格式储存的点阵图，该点阵图格式一直储存到通过呼叫 BitBlt 和 StretchBlt 来将位元图画到输出设备。

DIBSTRUCT 的第四个栏位是指向点阵图位元的指标。此值也可由 CreateDIBSection 函式设定。您将想起，作业系统将控制这个记忆体块，但应用程式有存取它的许可权。在删除点阵图代号时，记忆体块将自动释放。

DIBSTRUCT 的第五个栏位是 DIBSECTION 结构。如果您有从 CreateDIBSection 传回的点阵图代号，那么您可以将代号传递给 GetObject 函式以获得有关 DIBSECTION 结构中的点阵图资讯：

```
GetObject (hBitmap, sizeof (DIBSECTION), &ds) ;
```

作为提示，DIBSECTION 结构在 WINGDI.H 中定义如下：

```
typedef struct tagDIBSECTION {
    BITMAP                dsBm ;
    BITMAPINFOHEADER dsBmih ;
    DWORD                dsBitFields[3] ;    // Color masks
```

```

        HANDLE        dshSection ;
        DWORD         dsOffset ;
    }
DIBSECTION, * PDIBSECTION ;

```

第一个栏位是 BITMAP 结构，它与 CreateBitmapIndirect 一起建立点阵图物件，与 GetObject 一起传回关于 DDB 的资讯。第二个栏位是 BITMAPINFOHEADER 结构。不管点阵图资讯结构是否传递给 CreateDIBSection 函式，DIBSECTION 结构总有 BITMAPINFOHEADER 结构而不是其他结构，例如 BITMAPCOREHEADER 结构。这意味著在存取此结构时，DIBHELP.C 中的许多函式都不必检查与 OS/2 相容的 DIB。

对于 16 位元和 32 位元的 DIB，如果 BITMAPINFOHEADER 结构的 biCompression 栏位是 BI_BITFIELDS，那么在资讯表头结构后面通常有三个遮罩值。这些遮罩值决定如何将 16 位元和 32 位图素值转换成 RGB 颜色。遮罩储存在 DIBSECTION 结构的第三个栏位中。

DIBSECTION 结构的最後两个栏位指的是 DIB 区块，此区块由档案映射建立。DIBHELP 不使用 CreateDIBSection 的这个特性，因此可以忽略这些栏位。

DIBSTRUCT 的最後两个栏位储存左右移位值，这些值用于处理 16 位元和 32 位元 DIB 的颜色遮罩。我们将在第十五章讨论这些移位值。

让我们再回来看一下 DIBSTRUCT 的第一个栏位。正如我们所看到的一样，在开始建立 DIB 时，此栏位设定为指向一个指标阵列的指标，该阵列中的每个指标都指向 DIB 中的一行图素。这些指标允许以更快的方式来获得 DIB 图素位元，同时也被定义，以便顶行可以首先引用 DIB 图素位元。此阵列的最後一个元素——引用 DIB 图像的最底行——通常等于 DIBSTRUCT 的 pBits 栏位。

资讯函式

DIBHELP.C 以定义 DIBSTRUCT 结构开始，然後提供一个函式集，此函式集允许应用程式获得有关 DIB 区块的资讯。程式 16-19 显示了 DIBHELP.C 的第一部分。

程式 16-19 DIBHELP.C 档案的第一部分

```

DIBHELP.C (第一部分)
/*-----
        DIBHELP.C --      DIB Section Helper Routines
                                (c) Charles Petzold, 1998
    -----
*/

#include <windows.h>
#include "dibhelp.h"

```

```

#define HDIB_SIGNATURE (* (int *) "Dib ")
typedef struct
{
    PBYTE          *    ppRow ;    // must be first field for macros!
    int             iSignature ;
    HBITMAP         hBitmap ;
    BYTE           *    pBits ;
    DIBSECTION      ds ;
    int             iRShift[3] ;
    int             iLShift[3] ;
}
DIBSTRUCT, * PDIBSTRUCT ;

/*-----
--
    DibIsValid: Returns TRUE if hdib points to a valid DIBSTRUCT
-----
-*/

BOOL DibIsValid (HDIB hdib)
{
    PDIBSTRUCT pdib = hdib ;
    if (pdib == NULL)
        return FALSE ;
    if (IsBadReadPtr (pdib, sizeof (DIBSTRUCT)))
        return FALSE ;
    if (pdib->iSignature != HDIB_SIGNATURE)
        return FALSE ;
    return TRUE ;
}

/*-----
--
    DibBitmapHandle: Returns the handle to the DIB section bitmap object
-----
-*/

HBITMAP DibBitmapHandle (HDIB hdib)
{
    if (!DibIsValid (hdib))
        return NULL ;
    return ((PDIBSTRUCT) hdib)->hBitmap ;
}

/*-----
-
    DibWidth: Returns the bitmap pixel width

```

```
-----
-*/

int DibWidth (HDIB h dib)
{
    if (!DibIsValid (h dib))
        return 0 ;
    return ((PDIBSTRUCT) h dib)->ds.dsBm.bmWidth ;
}

/*-----
-
    DibHeight:  Returns the bitmap pixel height
-----
*/

int DibHeight (HDIB h dib)
{
    if (!DibIsValid (h dib))
        return 0 ;
    return ((PDIBSTRUCT) h dib)->ds.dsBm.bmHeight ;
}

/*-----
-
    DibBitCount:  Returns the number of bits per pixel
-----
*/

int DibBitCount (HDIB h dib)
{
    if (!DibIsValid (h dib))
        return 0 ;

    return ((PDIBSTRUCT) h dib)->ds.dsBm.bmBitsPixel ;
}

/*-----
--
    DibRowLength:  Returns the number of bytes per row of pixels
-----
-*/

int DibRowLength (HDIB h dib)
{
    if (!DibIsValid (h dib))
        return 0 ;
    return 4 * ((DibWidth (h dib) * DibBitCount (h dib) + 31) / 32) ;
}
```

```

}

/*-----
-
    DibNumColors:      Returns the number of colors in the color table
-----
*/

int DibNumColors (HDIB hdib)
{
    PDIBSTRUCT pdib = hdib ;

    if (!DibIsValid (hdib))
        return 0 ;

    if (pdib->ds.dsBmih.biClrUsed != 0)
    {
        return pdib->ds.dsBmih.biClrUsed ;
    }
    else if (DibBitCount (hdib) <= 8)
    {
        return 1 << DibBitCount (hdib) ;
    }
    return 0 ;
}

/*-----
-
    DibMask: Returns one of the color masks
-----*
/

DWORD      DibMask (HDIB hdib, int i)
{
    PDIBSTRUCT pdib = hdib ;

    if (!DibIsValid (hdib) || i < 0 || i > 2)
        return 0 ;

    return pdib->ds.dsBitFields[i] ;
}

/*-----
--
    DibRShift: Returns one of the right-shift values
-----
-*/

int      DibRShift (HDIB hdib, int i)

```

```
{
    PDIBSTRUCT pdib = hdib ;

    if (!DibIsValid (hdib) || i < 0 || i > 2)
        return 0 ;

    return pdib->iRShift[i] ;
}

/*-----
--
    DibLShift:  Returns one of the left-shift values
-----
*/

int DibLShift (HDIB hdib, int i)
{
    PDIBSTRUCT pdib = hdib ;

    if (!DibIsValid (hdib) || i < 0 || i > 2)
        return 0 ;

    return pdib->iLShift[i] ;
}

/*-----
-
    DibCompression:  Returns the value of the biCompression field
-----
*/
int DibCompression (HDIB hdib)
{
    if (!DibIsValid (hdib))
        return 0 ;

    return ((PDIBSTRUCT) hdib)->ds.dsBmih.biCompression ;
}

/*-----
-
    DibIsAddressable:  Returns TRUE if the DIB is not compressed
-----
*/

BOOL DibIsAddressable (HDIB hdib)
{
    int iCompression ;
```



```

        if (!DibIsValid (hdib))
            return FALSE ;

        iCompression = DibCompression (hdib) ;

        if ( iCompression == BI_RGB || iCompression == BI_BITFIELDS)
            return TRUE ;

        return FALSE ;
}

/*-----
-
    These functions return the sizes of various components of the DIB section
    AS THEY WOULD APPEAR in a packed DIB. These functions aid in converting
    the DIB section to a packed DIB and in saving DIB files.
-----
-*/

DWORD DibInfoHeaderSize (HDIB hdib)
{
    if (!DibIsValid (hdib))
        return 0 ;
    return ((PDIBSTRUCT) hdib)->ds.dsBmih.biSize ;
}

DWORD      DibMaskSize (HDIB hdib)
{
    PDIBSTRUCT pdib = hdib ;
    if (!DibIsValid (hdib))
        return 0 ;

    if (pdib->ds.dsBmih.biCompression == BI_BITFIELDS)
        return 3 * sizeof (DWORD) ;

    return 0 ;
}

DWORD DibColorSize (HDIB hdib)
{
    return DibNumColors (hdib) * sizeof (RGBQUAD) ;
}

DWORD DibInfoSize (HDIB hdib)
{
    return DibInfoHeaderSize(hdib) + DibMaskSize(hdib) + DibColorSize(hdib) ;
}

```

```
DWORD DIBBitsSize (HDIB hdib)
{
    PDIBSTRUCT pdib = hdib ;

    if (!DIBIsValid (hdib))
        return 0 ;

    if (pdib->ds.dsBmih.biSizeImage != 0)
    {
        return pdib->ds.dsBmih.biSizeImage ;
    }
    return DIBHeight (hdib) * DIBRowLength (hdib) ;
}

DWORD DIBTotalSize (HDIB hdib)
{
    return DIBInfoSize (hdib) + DIBBitsSize (hdib) ;
}

/*-----
-
    These functions return pointers to the various components of the DIB
    section.
-----
-*/

BITMAPINFOHEADER * DIBInfoHeaderPtr (HDIB hdib)
{
    if (!DIBIsValid (hdib))
        return NULL ;
    return & (((PDIBSTRUCT) hdib)->ds.dsBmih) ;
}

DWORD * DIBMaskPtr (HDIB hdib)
{
    PDIBSTRUCT pdib = hdib ;
    if (!DIBIsValid (hdib))
        return 0 ;
    return pdib->ds.dsBitfields ;
}

void * DIBBitsPtr (HDIB hdib)
{
    if (!DIBIsValid (hdib))
        return NULL ;
    return ((PDIBSTRUCT) hdib)->pBits ;
}

/*-----
```

```

-
    DibSetColor:  Obtains entry from the DIB color table
-----
-*/

BOOL DibGetColor (HDIB hdib, int index, RGBQUAD * prgb)
{
    PDIBSTRUCT pdib = hdib ;
    HDC          hdcMem ;
    int          iReturn ;

    if (!DibIsValid (hdib))
        return 0 ;
    hdcMem = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMem, pdib->hBitmap) ;
    iReturn = GetDIBColorTable (hdcMem, index, 1, prgb) ;
    DeleteDC (hdcMem) ;
    return iReturn ? TRUE : FALSE ;
}

/*-----
--
    DibGetColor:  Sets an entry in the DIB color table
-----
*/

BOOL DibSetColor (HDIB hdib, int index, RGBQUAD * prgb)
{
    PDIBSTRUCT pdib = hdib ;
    HDC          hdcMem ;
    int          iReturn ;

    if (!DibIsValid (hdib))
        return 0 ;
    hdcMem = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMem, pdib->hBitmap) ;
    iReturn = SetDIBColorTable (hdcMem, index, 1, prgb) ;
    DeleteDC (hdcMem) ;

    return iReturn ? TRUE : FALSE ;
}

```

DIBHELP.C 中的大部分函式是不用解释的。DibIsValid 函式能有助於保护整个系统。在试图引用 DIBSTRUCT 中的资讯之前，其他函式都呼叫 DibIsValid。所有这些函式都有（而且通常是只有）HDIB 型态的第一个参数，（我们将立即看到）该参数在 DIBHELP.H 中定义为空指标。这些函式可以将此参数储存到 PDIBSTRUCT，然後再存取结构中的栏位。

注意传回 BOOL 值的 DIBIsAddressable 函式。DIBIsNotCompressed 函式也可以呼叫此函式。传回值表示独立的 DIB 图素能否定址。

以 DIBInfoHeaderSize 开始的函式集将取得 DIB 区块中不同元件出现在 packed DIB 中的大小。与我们所看到的一样，这些函式有助於将 DIB 区块转换成 packed DIB，并储存 DIB 档案。这些函式的後面是获得指向不同 DIB 元件的指标的函式集。

尽管 DIBHELP.C 包括名称为 DIBInfoHeaderPtr 的函式，而且该函式将获得指向 BITMAPINFOHEADER 结构的指标，但还是没有函式可以获得 BITMAPINFO 结构指标——即接在 DIB 颜色表後面的资讯结构。这是因为在处理 DIB 区块时，应用程式并不直接存取这种型态的结构。BITMAPINFOHEADER 结构和颜色遮罩都在 DIBSECTION 结构中有效，而且从 CreateDIBSection 函式传回指向图素位元的指标，这时通过呼叫 GetDIBColorTable 和 SetDIBColorTable，就只能间接存取 DIB 颜色表。这些功能都封装到 DIBHELP 的 DIBGetColor 和 DIBSetColor 函式里头了。

在 DIBHELP.C 的後面，档案 DIBCopyToInfo 配置一个指向 BITMAPINFO 结构的指标，并填充资讯，但是那与获得指向记忆体中现存结构的指标不完全相同。

读、写图素

应用程式维护 packed DIB 或 DIB 区块的一个引人注目的优点是能够直接操作 DIB 图素位元。程式 16-20 所示的 DIBHELP.C 第二部分列出了提供此功能的函式。

程式 16-20 DIBHELP.C 档案的第二部分

```
DIBHELP.C (第二部分)
/*-----
--
        DIBPixelPtr:      Returns a pointer to the pixel at position (x, y)
-----
*/

BYTE * DIBPixelPtr (HDIB hdib, int x, int y)
{
    if (!DIBIsAddressable (hdib))
        return NULL ;
    if (x < 0 || x >= DIBWidth (hdib) || y < 0 || y >= DIBHeight (hdib))
        return NULL ;
    return (((PDIBSTRUCT) hdib)->ppRow)[y] + (x * DIBBitCount (hdib) >> 3) ;
}

/*-----
```

```

-
    DibGetPixel:  Obtains a pixel value at (x, y)
-----
-*/

DWORD DibGetPixel (HDIB hdib, int x, int y)
{
    PBYTE pPixel ;
    if (!(pPixel = DibPixelPtr (hdib, x, y)))
        return 0 ;
    switch (DibBitCount (hdib))
    {
    case 1:  return 0x01 & (* pPixel >> (7 - (x & 7))) ;
    case 4:  return 0x0F & (* pPixel >> (x & 1 ? 0 : 4)) ;
    case 8:  return * pPixel ;
    case 16: return * (WORD *) pPixel ;
    case 24: return 0x00FFFFFF & * (DWORD *) pPixel ;
    case 32: return * (DWORD *) pPixel ;
    }
    return 0 ;
}

/*-----
    DibSetPixel:  Sets a pixel value at (x, y)
-----*/

BOOL DibSetPixel (HDIB hdib, int x, int y, DWORD dwPixel)
{
    PBYTE pPixel ;
    if (!(pPixel = DibPixelPtr (hdib, x, y)))
        return FALSE ;
    switch (DibBitCount (hdib))
    {
    case 1:  * pPixel &= ~(1 << (7 - (x & 7))) ;
            * pPixel |= dwPixel << (7 - (x & 7)) ;
            break ;
    case 4:  * pPixel &= 0x0F << (x & 1 ? 4 : 0) ;
            * pPixel |= dwPixel << (x & 1 ? 0 : 4) ;
            break ;
    case 8:  * pPixel = (BYTE) dwPixel ;
            break ;
    case 16: * (WORD *) pPixel = (WORD) dwPixel ;
            break ;
    case 24: * (RGBTRIPLE *) pPixel = * (RGBTRIPLE *) &dwPixel ;
            break ;
    }
}

```

```

        case 32:      *      (DWORD *) pPixel = dwPixel ;
                        break ;

        default:
                        return FALSE ;

    }
    return TRUE ;
}

/*-----
-
    DibGetPixelColor:  Obtains the pixel color at (x, y)
-----*/

BOOL DibGetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb)
{
    DWORD          dwPixel ;
    int             iBitCount ;
    PDIBSTRUCT      pdib = hdib ;

                        // Get bit count; also use this as a validity check

    if (0 == (iBitCount = DibBitCount (hdib)))
        return FALSE ;
                        // Get the pixel value
    dwPixel = DibGetPixel (hdib, x, y) ;
                        // If the bit-count is 8 or less, index the color table
    if (iBitCount <= 8)
        return DibGetColor (hdib, (int) dwPixel, prgb) ;
                        // If the bit-count is 24, just use the pixel
    else if (iBitCount == 24)
    {
        *      (RGBTRIPLE *) prgb = * (RGBTRIPLE *) & dwPixel ;
        prgb->rgbReserved = 0 ;
    }

                        // If the bit-count is 32 and the biCompression field is BI_RGB,
                        // just use the pixel

    else if (iBitCount == 32 &&
              pdib->ds.dsBmih.biCompression == BI_RGB)
    {
        *      prgb = * (RGBQUAD *) & dwPixel ;
    }

                        // Otherwise, use the mask and shift values
                        // (for best performance, don't use DibMask and DibShift

```

```

functions)
    else
    {
        prgb->rgbRed = (BYTE) (((pdib->ds.dsBitFields[0] & dwPixel)
            >> pdib->iRShift[0]) << pdib->iLShift[0]) ;

        prgb->rgbGreen=(BYTE) ((pdib->ds.dsBitFields[1] & dwPixel)
            >> pdib->iRShift[1]) << pdib->iLShift[1]) ;

        prgb->rgbBlue=(BYTE) (((pdib->ds.dsBitFields[2] & dwPixel)
            >> pdib->iRShift[2]) << pdib->iLShift[2]) ;
    }
    return TRUE ;
}

/*-----
---
    DibSetPixelColor:  Sets the pixel color at (x, y)
-----
-*/

BOOL DibSetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb)
{
    DWORD          dwPixel ;
    int             iBitCount ;
    PDIBSTRUCT      pdib = hdib ;

    // Don't do this function for DIBs with color tables

    iBitCount = DibBitCount (hdib) ;
    if (iBitCount <= 8)
        return FALSE ;
    // The rest is just the opposite of DibGetPixelColor
    else if (iBitCount == 24)
    {
        * (RGBTRIPLE *) & dwPixel = * (RGBTRIPLE *) prgb ;
        dwPixel &= 0x00FFFFFF ;
    }
    else if (iBitCount == 32 &&
        pdib->ds.dsBmih.biCompression == BI_RGB)
    {
        * (RGBQUAD *) & dwPixel = * prgb ;
    }

    else
    {
        dwPixel =          (((DWORD)  prgb->rgbRed >> pdib->iLShift[0])
            << pdib->iRShift[0]) ;
    }
}

```

```

    dwPixel |= (((DWORD) prgb->rgbGreen >> pdib->iLShift[1])
<< pdib->iRShift[1]) ;

    dwPixel |= (((DWORD) prgb->rgbBlue >> pdib->iLShift[2])
<< pdib->iRShift[2]) ;
}

DibSetPixel (hdib, x, y, dwPixel) ;
return TRUE ;
}

```

这部分 DIBHELP.C 从 DibPixelPtr 函式开始, 该函式获得指向储存 (或部分储存) 有特殊图素的位元组的指标。回想一下 DIBSTRUCT 结构的 ppRow 栏位, 那是个指向 DIB 中由顶行开始排列的图素行位址的指标。这样,

```
((PDIBSTRUCT) hdib)->pprow)[0]
```

就是指向 DIB 顶行最左端图素的指标, 而

```
((PDIBSTRUCT) hdib)->ppRow)[y] + (x * DibBitCount (hdib) >> 3)
```

是指向位於 (x, y) 的图素的指标。注意, 如果 DIB 中的图素不可被定址 (即如果已压缩), 或者如果函式的 x 和 y 参数是负数或相对位於 DIB 外面的区域, 则函式将传回 NULL。此检查降低了函式 (和所有依赖於 DibPixelPtr 的函式) 的执行速度, 下面我将讲述一些更快的常式。

档案後面的 DibGetPixel 和 DibSetPixel 函式利用了 DibPixelPtr。對於 8 位元、16 位元和 32 位元 DIB, 这些函式只记录指向合适资料尺寸的指标, 并存取图素值。對於 1 位元和 4 位元的 DIB, 则需要遮罩和移位角度。

DibGetColor 函式按 RGBQUAD 结构获得图素颜色。對於 1 位元、4 位元和 8 位元 DIB, 这包括使用图素值来从 DIB 颜色表获得颜色。對於 16 位元、24 位元和 32 位元 DIB, 通常必须将图素值遮罩和移位以得到 RGB 颜色。DibSetPixel 函式则相反, 它允许从 RGBQUAD 结构设定图素值。该函式只为 16 位元、24 位元和 32 位元 DIB 定义。

建立和转换

程式 16-21 所示的 DIBHELP 第三部分和最後部分展示了如何建立 DIB 区块, 以及如何将 DIB 区块与 packed DIB 相互转换。

程式 16-21 DIBHELP.C 档案的第三部分和最後部分

DIBHELP.C (第三部分)

```

/*-----
    Calculating shift values from color masks is required by the
    DibCreateFromInfo function.
    -----*/

```



```

*/

static int MaskToRShift (DWORD dwMask)
{
    int iShift ;
    if (dwMask == 0)
        return 0 ;
    for (iShift = 0 ; !(dwMask & 1) ; iShift++)
        dwMask >>= 1 ;
    return iShift ;
}

static int MaskToLShift (DWORD dwMask)
{
    int iShift ;
    if (dwMask == 0)
        return 0 ;
    while (!(dwMask & 1))
        dwMask >>= 1 ;
    for (iShift = 0 ; dwMask & 1 ; iShift++)
        dwMask >>= 1 ;
    return 8 - iShift ;
}

/*-----
--
    DibCreateFromInfo: All DIB creation functions ultimately call this one.
    This function is responsible for calling CreateDIBSection, allocating
    memory for DIBSTRUCT, and setting up the row pointer.
-----
-*/

HDIB DibCreateFromInfo (BITMAPINFO * pbmi)
{
    BYTE * pBits ;
    DIBSTRUCT * pdib ;
    HBITMAP hBitmap ;
    int i, iRowLength, cy, y ;

    hBitmap = CreateDIBSection (NULL, pbmi, DIB_RGB_COLORS, &pBits, NULL, 0) ;
    if (hBitmap == NULL)
        return NULL ;
    if (NULL == (pdib = malloc (sizeof (DIBSTRUCT))))
    {
        DeleteObject (hBitmap) ;
        return NULL ;
    }

    pdib->iSignature = HDIB_SIGNATURE ;

```

```

pdib->hBitmap      = hBitmap ;
pdib->pBits         = pBits ;

GetObject (hBitmap, sizeof (DIBSECTION), &pdib->ds) ;
    // Notice that we can now use the DIB information functions
    //   defined above.

    // If the compression is BI_BITFIELDS, calculate shifts from
masks

if (DibCompression (pdib) == BI_BITFIELDS)
{
    for (i = 0 ; i < 3 ; i++)
    {
        pdib->iLShift[i]      =      MaskToLShift
(pdib->ds.dsBitfields[i]) ;
        pdib->iRShift[i]      =      MaskToRShift
(pdib->ds.dsBitfields[i]) ;
    }
}

    // If the compression is BI_RGB, but bit-count is 16 or 32,
    //   set the bitfields and the masks
else if (DibCompression (pdib) == BI_RGB)
{
    if (DibBitCount (pdib) == 16)
    {
        pdib->ds.dsBitfields[0] = 0x00007C00 ;
        pdib->ds.dsBitfields[1] = 0x000003E0 ;
        pdib->ds.dsBitfields[2] = 0x0000001F ;

        pdib->iRShift   [0]   =      10      ;
        pdib->iRShift   [1]   =      5       ;
        pdib->iRShift   [2]   =      0       ;

        pdib->iLShift   [0]   =      3       ;
        pdib->iLShift   [1]   =      3       ;
        pdib->iLShift   [2]   =      3       ;
    }
    else if (DibBitCount (pdib) == 24 || DibBitCount (pdib) == 32)
    {
        pdib->ds.dsBitfields[0] = 0x00FF0000 ;
        pdib->ds.dsBitfields[1] = 0x0000FF00 ;
        pdib->ds.dsBitfields[2] = 0x000000FF ;

        pdib->iRShift   [0]   =      16      ;
        pdib->iRShift   [1]   =      8       ;
        pdib->iRShift   [2]   =      0       ;
    }
}

```

```

        pdib->iLShift    [0]    =        0        ;
        pdib->iLShift    [1]    =        0        ;
        pdib->iLShift    [2]    =        0        ;
    }
}

    // Allocate an array of pointers to each row in the DIB
cy = DibHeight (pdib) ;
if (NULL == (pdib->ppRow = malloc (cy * sizeof (BYTE *))))
{
    free (pdib) ;
    DeleteObject (hBitmap) ;
    return NULL ;
}

    // Initialize them.
iRowLength = DibRowLength (pdib) ;
if (pbmi->bmiHeader.biHeight > 0)                // ie, bottom up
{
    for (y = 0 ; y < cy ; y++)
        pdib->ppRow[y] = pBits + (cy - y - 1) * iRowLength ;
}
else

// top down
{
    for (y = 0 ; y < cy ; y++)
        pdib->ppRow[y] = pBits + y * iRowLength ;
}
return pdib ;
}

/*-----
    DibDelete:  Frees all memory for the DIB section
-----*/

BOOL DibDelete (HDIB hdib)
{
    DIBSTRUCT * pdib = hdib ;
    if (!DibIsValid (hdib))
        return FALSE ;
    free (pdib->ppRow) ;
    DeleteObject (pdib->hBitmap) ;
    free (pdib) ;
    return TRUE ;
}

```

```

/*-----
--
    DibCreate: Creates an HDIB from explicit arguments
-----
*/

HDIB DibCreate (int cx, int cy, int cBits, int cColors)
{
    BITMAPINFO *    pbmi ;
    DWORD           dwInfoSize ;
    HDIB            hDib ;
    int             cEntries ;

    if (cx <= 0 || cy <= 0 ||
        ((cBits != 1) && (cBits != 4) && (cBits != 8) &&
         (cBits != 16) && (cBits != 24) && (cBits != 32)))
    {
        return NULL ;
    }

    if ( cColors != 0)
        cEntries = cColors ;
    else if (cBits <= 8)
        cEntries = 1 << cBits ;

    dwInfoSize = sizeof (BITMAPINFOHEADER) + (cEntries - 1) * sizeof (RGBQUAD);

    if (NULL == (pbmi = malloc (dwInfoSize)))
    {
        return NULL ;
    }

    ZeroMemory (pbmi, dwInfoSize) ;

    pbmi->bmiHeader.biSize           = sizeof (BITMAPINFOHEADER) ;
    pbmi->bmiHeader.biWidth          = cx ;
    pbmi->bmiHeader.biHeight         = cy ;
    pbmi->bmiHeader.biPlanes         = 1 ;
    pbmi->bmiHeader.biBitCount       = cBits ;
    pbmi->bmiHeader.biCompression   = BI_RGB ;
    pbmi->bmiHeader.biSizeImage      = 0 ;
    pbmi->bmiHeader.biXPelsPerMeter  = 0 ;
    pbmi->bmiHeader.biYPelsPerMeter  = 0 ;
    pbmi->bmiHeader.biClrUsed        = cColors ;
    pbmi->bmiHeader.biClrImportant   = 0 ;

    hDib = DibCreateFromInfo (pbmi) ;
    free (pbmi) ;
}

```

```

        return hDib ;
    }

/*-----
--
        DibCopyToInfo:          Builds BITMAPINFO structure.
                                Used by DibCopy and
DibCopyToDdb
-----
-*/

static BITMAPINFO * DibCopyToInfo (HDIB hdib)
{
    BITMAPINFO      *    pbmi ;
    int              i, iNumColors ;
    RGBQUAD         *    prgb ;
    if (!DibIsValid (hdib))
        return NULL ;
        // Allocate the memory
    if (NULL == (pbmi = malloc (DibInfoSize (hdib))))
        return NULL ;
        // Copy the information header
    CopyMemory (pbmi, DibInfoHeaderPtr (hdib), sizeof (BITMAPINFOHEADER));

        // Copy the possible color masks

    prgb = (RGBQUAD *) ((BYTE *) pbmi + sizeof (BITMAPINFOHEADER)) ;
    if (DibMaskSize (hdib))
    {
        CopyMemory (prgb, DibMaskPtr (hdib), 3 * sizeof (DWORD)) ;
        prgb = (RGBQUAD *) ((BYTE *) prgb + 3 * sizeof (DWORD)) ;
    }

        // Copy the color table
    iNumColors = DibNumColors (hdib) ;
    for (i = 0 ; i < iNumColors ; i++)
        DibGetColor (hdib, i, prgb + i) ;
    return pbmi ;
}

/*-----
--
        DibCopy:  Creates a new DIB section from an existing DIB section,
                    possibly swapping the DIB width and height.
-----
-*/

HDIB DibCopy (HDIB hdibSrc, BOOL fRotate)
{

```

```

    BITMAPINFO      *      pbmi ;
    BYTE            *      pBitsSrc, * pBitsDst ;
    HDIB             hdibDst ;

    if (!DibIsValid (hdibSrc))
        return NULL ;
    if (NULL == (pbmi = DibCopyToInfo (hdibSrc)))
        return NULL ;

    if (fRotate)
    {
        pbmi->bmiHeader.biWidth = DibHeight (hdibSrc) ;
        pbmi->bmiHeader.biHeight = DibWidth (hdibSrc) ;
    }

    hdibDst = DibCreateFromInfo (pbmi) ;
    free (pbmi) ;

    if ( hdibDst == NULL)
        return NULL ;

                                // Copy the bits

    if (!fRotate)
    {
        pBitsSrc = DibBitsPtr (hdibSrc) ;
        pBitsDst = DibBitsPtr (hdibDst) ;

                                CopyMemory (pBitsDst,  pBitsSrc,  DibBitsSize
(hdibSrc)) ;
    }
    return hdibDst ;
}

/*-----
--
    DibCopyToPackedDib is generally used for saving DIBs and for
    transferring DIBs to the clipboard. In the second case, the second
    argument should be set to TRUE so that the memory is allocated
    with the GMEM_SHARE flag.
-----
-*/

BITMAPINFO * DibCopyToPackedDib (HDIB hdib, BOOL fUseGlobal)
{
    BITMAPINFO      *      pPackedDib ;
    BYTE            *      pBits ;
    DWORD            dwDibSize ;
    HDC              hdcMem ;

```

```
HGLOBAL          hGlobal ;
int              iNumColors ;
PDIBSTRUCT      pdib = hdib ;
RGBQUAD         *   prgb ;

if (!DibIsValid (hdib))
    return NULL ;
    // Allocate memory for packed DIB
dwDibSize = DibTotalSize (hdib) ;
if (fUseGlobal)
{
    hGlobal = GlobalAlloc (GHND | GMEM_SHARE, dwDibSize) ;
    pPackedDib = GlobalLock (hGlobal) ;
}
else
{
    pPackedDib = malloc (dwDibSize) ;
}

if (pPackedDib == NULL)
    return NULL ;
    // Copy the information header
CopyMemory (pPackedDib, &pdib->ds.dsBmih, sizeof (BITMAPINFOHEADER)) ;
prgb = (RGBQUAD *) ((BYTE *) pPackedDib + sizeof (BITMAPINFOHEADER)) ;
    // Copy the possible color masks
if (pdib->ds.dsBmih.biCompression == BI_BITFIELDS)
{
    CopyMemory (prgb, pdib->ds.dsBitFields, 3 * sizeof (DWORD)) ;
    prgb = (RGBQUAD *) ((BYTE *) prgb + 3 * sizeof (DWORD)) ;
}
    // Copy the color table
if (iNumColors = DibNumColors (hdib))
{
    hdcMem = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMem, pdib->hBitmap) ;
    GetDIBColorTable (hdcMem, 0, iNumColors, prgb) ;
    DeleteDC (hdcMem) ;
}

pBits = (BYTE *) (prgb + iNumColors) ;
    // Copy the bits
CopyMemory (pBits, pdib->pBits, DibBitsSize (pdib)) ;
    // If last argument is TRUE, unlock global memory block and
    // cast it to pointer in preparation for return

if (fUseGlobal)
{
    GlobalUnlock (hGlobal) ;
```

```

        pPackedDib = (BITMAPINFO *) hGlobal ;
    }
    return pPackedDib ;
}

/*-----
DibCopyFromPackedDib is generally used for pasting DIBs from the
clipboard.
-----*/

HDIB DibCopyFromPackedDib (BITMAPINFO * pPackedDib)
{
    BYTE          *      pBits ;
    DWORD          dwInfoSize, dwMaskSize, dwColorSize ;
    int            iBitCount ;
    PDIBSTRUCT     pdib ;

    // Get the size of the information header and do validity
check

    dwInfoSize = pPackedDib->bmiHeader.biSize ;
    if ( dwInfoSize != sizeof (BITMAPCOREHEADER) &&
        dwInfoSize != sizeof (BITMAPINFOHEADER) &&
        dwInfoSize != sizeof (BITMAPV4HEADER) &&
        dwInfoSize != sizeof (BITMAPV5HEADER) )
    {
        return NULL ;
    }

    // Get the possible size of the color masks

    if (dwInfoSize == sizeof (BITMAPINFOHEADER) &&
        pPackedDib->bmiHeader.biCompression == BI_BITFIELDS)
    {
        dwMaskSize = 3 * sizeof (DWORD) ;
    }
    else
    {
        dwMaskSize = 0 ;
    }

    // Get the size of the color table
    if (dwInfoSize == sizeof (BITMAPCOREHEADER))
    {
        iBitCount = ((BITMAPCOREHEADER *) pPackedDib)->bcBitCount ;

        if (iBitCount <= 8)
        {
            dwColorSize = (1 << iBitCount) * sizeof (RGBTRIPLE) ;
        }
    }
}

```



```

        else
            dwColorSize = 0 ;
    }
    else
        // all non-OS/2 compatible DIBs
    {
        if (pPackedDib->bmiHeader.biClrUsed > 0)
        {
            dwColorSize = pPackedDib->bmiHeader.biClrUsed * sizeof (RGBQUAD);
        }
        else if (pPackedDib->bmiHeader.biBitCount <= 8)
        {
            dwColorSize = (1 <<
pPackedDib->bmiHeader.biBitCount) * sizeof (RGBQUAD) ;
        }
        else
        {
            dwColorSize = 0 ;
        }
    }

    // Finally, get the pointer to the bits in the packed DIB
    pBits = (BYTE *) pPackedDib + dwInfoSize + dwMaskSize + dwColorSize ;
    // Create the HDIB from the packed-DIB pointer
    pdib = DibCreateFromInfo (pPackedDib) ;
    // Copy the pixel bits
    CopyMemory (pdib->pBits, pBits, DibBitsSize (pdib)) ;
    return pdib ;
}

/*-----
--
    DibFileLoad:  Creates a DIB section from a DIB file
-----
-*/
HDIB DibFileLoad (const TCHAR * szFileName)
{
    BITMAPFILEHEADER    bmfh ;
    BITMAPINFO           *    pbmi ;
    BOOL                bSuccess ;
    DWORD                dwInfoSize,    dwBitsSize,
dwBytesRead ;
    HANDLE                hFile ;
    HDIB                  hDib ;

    // Open the file: read access, prohibit write access

    hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
    if (hFile == INVALID_HANDLE_VALUE)

```

```
        return NULL ;
        // Read in the BITMAPFILEHEADER
bSuccess = ReadFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
                    &dwBytesRead, NULL) ;

if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
    || (bmfh.bfType != * (WORD *) "BM"))
{
    CloseHandle (hFile) ;
    return NULL ;
}

    // Allocate memory for the information structure & read it in
dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
if (NULL == (pbmi = malloc (dwInfoSize)))
{
    CloseHandle (hFile) ;
    return NULL ;
}

bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;
if (!bSuccess || (dwBytesRead != dwInfoSize))
{
    CloseHandle (hFile) ;
    free (pbmi) ;
    return NULL ;
}

    // Create the DIB
hDib = DibCreateFromInfo (pbmi) ;
free (pbmi) ;

if (hDib == NULL)
{
    CloseHandle (hFile) ;
    return NULL ;
}

    // Read in the bits
dwBitsSize = bmfh.bfSize - bmfh.bfOffBits ;
bSuccess = ReadFile ( hFile, ((PDIBSTRUCT) hDib)->pBits,
                    dwBitsSize, &dwBytesRead, NULL) ;
CloseHandle (hFile) ;
if (!bSuccess || (dwBytesRead != dwBitsSize))
{
    DibDelete (hDib) ;
    return NULL ;
}
return hDib ;
}
```

```

/*-----
    DibFileSave:  Saves a DIB section to a file
-----*/

*/

BOOL DibFileSave (HDIB hdib, const TCHAR * szFileName)
{
    BITMAPFILEHEADER    bmfh ;
    BITMAPINFO           *    pbmi ;
    BOOL                 bSuccess ;
    DWORD                dwTotalSize, dwBytesWritten ;
    HANDLE                hFile ;

    hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
                        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;
    if (hFile == INVALID_HANDLE_VALUE)
        return FALSE ;

    dwTotalSize                = DibTotalSize (hdib) ;
    bmfh.bfType                 = * (WORD *) "BM" ;
    bmfh.bfSize                 = sizeof (BITMAPFILEHEADER) + dwTotalSize ;
    bmfh.bfReserved1            = 0 ;
    bmfh.bfReserved2            = 0 ;
    bmfh.bfOffBits              = bmfh.bfSize - DibBitsSize (hdib) ;

    // Write the BITMAPFILEHEADER

    bSuccess = WriteFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
                           &dwBytesWritten, NULL) ;

    if (!bSuccess || (dwBytesWritten != sizeof (BITMAPFILEHEADER)))
    {
        CloseHandle (hFile) ;
        DeleteFile (szFileName) ;
        return FALSE ;
    }

    // Get entire DIB in packed-DIB format
    if (NULL == (pbmi = DibCopyToPackedDib (hdib, FALSE)))
    {
        CloseHandle (hFile) ;
        DeleteFile (szFileName) ;
        return FALSE ;
    }

    // Write out the packed DIB
    bSuccess = WriteFile (hFile, pbmi, dwTotalSize, &dwBytesWritten, NULL) ;
    CloseHandle (hFile) ;
    free (pbmi) ;

    if (!bSuccess || (dwBytesWritten != dwTotalSize))

```

```

    {
        DeleteFile (szFileName) ;
        return FALSE ;
    }
    return TRUE ;
}

/*-----
-
    DibCopyToDdb:  For more efficient screen displays
-----*/
/
HBITMAP DibCopyToDdb (HDIB hdib, HWND hwnd, HPALETTE hPalette)
{
    BITMAPINFO      *    pbmi ;
    HBITMAP          hBitmap ;
    HDC              hdc ;

    if (!DibIsValid (hdib))
        return NULL ;
    if (NULL == (pbmi = DibCopyToInfo (hdib)))
        return NULL ;
    hdc = GetDC (hwnd) ;
    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }

    hBitmap = CreateDIBitmap (hdc, DibInfoHeaderPtr (hdib), CBM_INIT,
                             DibBitsPtr (hdib), pbmi,
DIB_RGB_COLORS) ;

    ReleaseDC (hwnd, hdc) ;
    free (pbmi) ;

    return hBitmap ;
}

```

这部分的 DIBHELP.C 档案从两个小函式开始, 这两个函式根据 16 位元和 32 位元 DIB 的颜色遮罩得到左、右移位值。这些函式在第十五章「颜色遮罩」一节说明。

DibCreateFromInfo 函式是 DIBHELP 中唯一呼叫 CreateDIBSection 并为 DIBSTRUCT 结构配置记忆体的函式。其他所有建立和复制函式都重复此函式。DibCreateFromInfo 唯一的参数是指向 BITMAPINFO 结构的指标。此结构的颜色表必须存在, 但是它不必用有效的值填充。呼叫 CreateDIBSection 之後, 该函

式将初始化 DIBSTRUCT 结构的所有栏位。注意，在设定 DIBSTRUCT 结构的 ppRow 栏位的值时（指向 DIB 行位址的指标），DIB 有由下而上和由上而下的不同储存方式。ppRow 开头的元素就是 DIB 的顶行。

DibDelete 删除 DibCreateFromInfo 中建立的点阵图，同时释放在该函式中配置的记忆体。

DibCreate 可能比 DibCreateFromInfo 更像一个从应用程式呼叫的函式。前三个参数提供图素的宽度、高度和每图素的位数。最後一个参数可以设定为 0（用於颜色表的内定尺寸），或者设定为非 0（表示比每图素位元数所需要的颜色表更小的颜色表）。

DibCopy 函式根据现存的 DIB 区块建立新的 DIB 区块，并用 DibCreateInfo 函式为 BITMAPINFO 结构配置了记忆体，还填了所有的资料。DibCopy 函式的一个 BOOL 参数指出是否在建立新的 DIB 时交换了 DIB 的宽度和高度。我们将在後面看到此函式的用法。

DibCopyToPackedDib 和 DibCopyFromPackedDib 函式的使用通常与透过剪贴簿传递 DIB 相关。DibFileLoad 函式从 DIB 档案建立 DIB 区块；DibFileSave 函式将资料储存在 DIB 档案。

最後，DibCopyToDdb 函式根据 DIB 建立 GDI 点阵图物件。注意，该函式需要目前调色盘的代号和程式视窗的代号。程式视窗代号用於获得选进并显现调色盘的装置内容。只有这样，函式才可以呼叫 CreateDIBitmap。这曾在本章前面的 SHOWDIB7 中展示。

DIBHELP 表头档案和巨集

DIBHELP.H 表头档案如程式 16-22 所示。

程式 16-22 DIBHELP.H 档案

```
DIBHELP.H
/*-----
   DIBHELP.H header file for DIBHELP.C
   -----*/

typedef void * HDIB ;
    // Functions in DIBHELP.C
BOOL DibIsValid (HDIB h dib) ;
HBITMAP DibBitmapHandle (HDIB h dib) ;
int DibWidth (HDIB h dib) ;
int DibHeight (HDIB h dib) ;
int DibBitCount (HDIB h dib) ;
int DibRowLength (HDIB h dib) ;
```

```

int DibNumColors (HDIB hdib) ;
DWORD DibMask (HDIB hdib, int i) ;
int DibRShift (HDIB hdib, int i) ;
int DibLShift (HDIB hdib, int i) ;
int DibCompression (HDIB hdib) ;
BOOL DibIsAddressable (HDIB hdib) ;
DWORD DibInfoHeaderSize (HDIB hdib) ;
DWORD DibMaskSize (HDIB hdib) ;
DWORD DibColorSize (HDIB hdib) ;
DWORD DibInfoSize (HDIB hdib) ;
DWORD DibBitsSize (HDIB hdib) ;
DWORD DibTotalSize (HDIB hdib) ;
BITMAPINFOHEADER * DibInfoHeaderPtr (HDIB hdib) ;
DWORD * DibMaskPtr (HDIB hdib) ;
void * DibBitsPtr (HDIB hdib) ;
BOOL DibGetColor (HDIB hdib, int index, RGBQUAD * prgb) ;
BOOL DibSetColor (HDIB hdib, int index, RGBQUAD * prgb) ;
BYTE * DibPixelPtr (HDIB hdib, int x, int y) ;
DWORD DibGetPixel (HDIB hdib, int x, int y) ;
BOOL DibSetPixel (HDIB hdib, int x, int y, DWORD dwPixel) ;
BOOL DibGetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb) ;
BOOL DibSetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb) ;
HDIB DibCreateFromInfo (BITMAPINFO * pbmi) ;
BOOL DibDelete (HDIB hdib) ;
HDIB DibCreate (int cx, int cy, int cBits, int cColors) ;
HDIB DibCopy (HDIB hdibSrc, BOOL fRotate) ;
BITMAPINFO * DibCopyToPackedDib (HDIB hdib, BOOL fUseGlobal) ;
HDIB DibCopyFromPackedDib (BITMAPINFO * pPackedDib) ;
HDIB DibFileLoad (const TCHAR * szFileName) ;
BOOL DibFileSave (HDIB hdib, const TCHAR * szFileName) ;
HBITMAP DibCopyToDdb (HDIB hdib, HWND hwnd, HPALETTE hPalette) ;
HDIB DibCreateFromDdb (HBITMAP hBitmap) ;

/*-----
-
    Quickie no-bounds-checked pixel gets and sets
-----
-*/

#define DibPixelPtr1(hdib, x, y) (((* (PBYTE **) hdib) [y]) + ((x) >> 3))
#define DibPixelPtr4(hdib, x, y) (((* (PBYTE **) hdib) [y]) + ((x) >> 1))
#define DibPixelPtr8(hdib, x, y) (((* (PBYTE **) hdib) [y]) + (x)
)
#define DibPixelPtr16(hdib, x, y) \
((WORD *) (((* (PBYTE **) hdib) [y])
+ (x) * 2))

#define DibPixelPtr24(hdib, x, y) \

```

```

                                                    ((RGBTRIPLE *) ((*) (PBYTE **))
hdib) [y]) + (x) * 3))
#define DIB_PIXEL_PTR32(hdib, x, y) \
                                                    ((DWORD *) ((*) (PBYTE **))
hdib) [y]) + (x) * 4))

#define DIB_GET_PIXEL1(hdib, x, y) \
                                                    (0x01 & (* DIB_PIXEL_PTR1 (hdib, x, y) >> (7 -
((x) & 7))))

#define DIB_GET_PIXEL4(hdib, x, y) \
                                                    (0x0F & (* DIB_PIXEL_PTR4 (hdib, x, y) >> ((x)
& 1 ? 0 : 4)))

#define DIB_GET_PIXEL8(hdib, x, y) \
                                                    (* DIB_PIXEL_PTR8
(hdib, x, y))
#define DIB_GET_PIXEL16(hdib, x, y) \
                                                    (* DIB_PIXEL_PTR16 (hdib, x, y))
#define DIB_GET_PIXEL24(hdib, x, y) \
                                                    (* DIB_PIXEL_PTR24 (hdib, x, y))
#define DIB_GET_PIXEL32(hdib, x, y) \
                                                    (* DIB_PIXEL_PTR32 (hdib, x, y))

#define DIB_SET_PIXEL1(hdib, x, y, p) \
                                                    \
                                                    ((*) DIB_PIXEL_PTR1 (hdib, x, y) &= ~(1 << (7 - ((x) & 7))),
\
                                                    (* DIB_PIXEL_PTR1 (hdib, x, y) |= ((p) << (7 - ((x) & 7))))

#define DIB_SET_PIXEL4(hdib, x, y, p) \
                                                    \
                                                    ((*) DIB_PIXEL_PTR4 (hdib, x, y) &= (0x0F << ((x) & 1 ? 4 :
0))), \
                                                    (* DIB_PIXEL_PTR4 (hdib, x, y) |= ((p) << ((x) & 1 ? 0 :
4))))

#define DIB_SET_PIXEL8(hdib, x, y, p) \
                                                    (* DIB_PIXEL_PTR8 (hdib, x, y) = p)
#define DIB_SET_PIXEL16(hdib, x, y, p) \
                                                    (* DIB_PIXEL_PTR16 (hdib, x, y) = p)
#define DIB_SET_PIXEL24(hdib, x, y, p) \
                                                    (* DIB_PIXEL_PTR24 (hdib, x, y) = p)
#define DIB_SET_PIXEL32(hdib, x, y, p) \
                                                    (* DIB_PIXEL_PTR32 (hdib, x, y) = p)

```

这个表头档案将 HDIB 定义为空指标(void*)。应用程式的确不需要了解 HDIB 所指结构的内部结构。此表头档案还包括 DIBHELP.C 中所有函式的说明, 还有一些巨集——非常特殊的巨集。

如果再看一看 DIBHELP.C 中的 DIB_PIXEL_PTR、DIB_GET_PIXEL 和 DIB_SET_PIXEL 函式, 并试图提高它们的执行速度表现, 那么您将看到两种可能的解决方法。第一种, 可以删除所有的检查保护, 并相信应用程式不会使用无效参数呼叫函式。还可以删除一些函式呼叫, 例如 DIB_BIT_COUNT, 并使用指向 DIBSTRUCT 结构内部的指标来直接获得资讯。

提高执行速度表现另一项较不明显的方法是删除所有对每图素位元数的处理方式，同时分离出处理不同 DIB 函式——例如 DibGetPixel1、DibGetPixel4、DibGetPixel8 等等。下一个最佳化步骤是删除整个函式呼叫，将其处理动作透过 inline function 或巨集中进行合并。

DIBHELP.H 采用巨集的方法。它依据 DibPixelPtr、DibGetPixel 和 DibSetPixel 函式提出了三套巨集。这些巨集都明确对应於特殊的图素位元数。

DIBBLE 程式

DIBBLE 程式，如程式 16-23 所示，使用 DIBHELP 函式和巨集工作。尽管 DIBBLE 是本书中最长的程式，它确实只是一些作业的粗略范例，这些作业可以在简单的数位影像处理程式中找到。对 DIBBLE 的明显改进是转换成了多重文件介面 (MDI: multiple document interface)，我们将在第十九章学习有关多重文件介面的知识。

程式 16-23 DIBBLE

```
DIBBLE.C
/*-----
-
-       DIBBLE.C --       Bitmap and Palette Program
-                               (c) Charles Petzold, 1998
-
-*/

#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"
#include "dibconv.h"
#include "resource.h"

#define WM_USER_SETSCROLLS          (WM_USER + 1)
#define WM_USER_DELETEDIB          (WM_USER + 2)
#define WM_USER_DELETEPAL          (WM_USER + 3)
#define WM_USER_CREATEPAL          (WM_USER + 4)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Dibble") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG             msg ;
```



```

WNDCLASS      wndclass ;

wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc     = WndProc ;
wndclass.cbClsExtra      = 0 ;
wndclass.cbWndExtra      = 0 ;
wndclass.hInstance      = hInstance ;
wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground   = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName     = szAppName ;
wndclass.lpszClassName   = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires
Windows NT!"),
                                szAppName,
MB_ICONERROR) ;

    return 0 ;
}

hwnd =    CreateWindow (    szAppName, szAppName,
    WS_OVERLAPPEDWINDOW | WM_VSCROLL | WM_HSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

hAccel = LoadAccelerators (hInstance, szAppName) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return msg.wParam ;
}

/*-----
--
DisplayDib:          Displays or prints DIB actual size or stretched
                        depending on menu selection

```

```

-----
-*/

int DisplayDib ( HDC hdc, HBITMAP hBitmap, int x, int y,
                int cxClient, int cyClient,
                WORD          wShow,          BOOL
fHalftonePalette)
{
    BITMAP          bitmap ;
    HDC              hdcMem ;
    int              cxBitmap, cyBitmap, iReturn ;
    GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;
    cxBitmap      =    bitmap.bmWidth ;
    cyBitmap      =    bitmap.bmHeight ;

    SaveDC (hdc) ;
    if (fHalftonePalette)
        SetStretchBltMode (hdc, HALFTONE) ;
    else
        SetStretchBltMode (hdc, COLORONCOLOR) ;
    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;

    switch (wShow)
    {
    case  IDM_SHOW_NORMAL:
        if (fHalftonePalette)
            iReturn = StretchBlt (hdc, 0, 0,
                                min (cxClient, cxBitmap - x),
                                min (cyClient, cyBitmap - y),
                                hdcMem, x, y, min (cxClient, cxBitmap - x),
                                min (cyClient, cyBitmap - y),
                                SRCCOPY);
        else
            iReturn = BitBlt (hdc, 0, 0, min (cxClient,
                                cxBitmap - x),
                                min (cyClient, cyBitmap - y),
                                hdcMem, x, y, SRCCOPY) ;
            break ;

    case  IDM_SHOW_CENTER:
        if (fHalftonePalette)
            iReturn = StretchBlt (hdc, (cxClient - cxBitmap) / 2,
                                (cyClient - cyBitmap) / 2,
                                cxBitmap, cyBitmap,
                                hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY);
        else
            iReturn = BitBlt (hdc, (cxClient - cxBitmap) / 2,

```

```

        cyClient - cyBitmap) / 2,
        cxBitmap, cyBitmap,
        hdcMem, 0, 0, SRCCOPY) ;
        break ;
case IDM_SHOW_STRETCH:
        iReturn = StretchBlt (hdc, 0, 0, cxClient, cyClient,
        hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
        break ;

case IDM_SHOW_ISOSTRETCH:
        SetMapMode (hdc, MM_ISOTROPIC) ;
        SetWindowExtEx (hdc, cxBitmap, cyBitmap, NULL) ;
        SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
        SetWindowOrgEx (hdc, cxBitmap / 2, cyBitmap / 2, NULL) ;
        SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

        iReturn = StretchBlt (hdc, 0, 0, cxBitmap, cyBitmap,
        hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
        break ;
}
DeleteDC (hdcMem) ;
RestoreDC (hdc, -1) ;
return iReturn ;
}

/*-----
-
DibFlipHorizontal: Calls non-optimized DibSetPixel and DibGetPixel
-----*/

HDIB DibFlipHorizontal (HDIB hdibSrc)
{
    HDIB hdibDst ;
    int cx, cy, x, y ;

    if (!DibIsAddressable (hdibSrc))
        return NULL ;
    if (NULL == (hdibDst = DibCopy (hdibSrc, FALSE)))
        return NULL ;
    cx = DibWidth (hdibSrc) ;
    cy = DibHeight (hdibSrc) ;

    for (x = 0 ; x < cx ; x++)
        for (y = 0 ; y < cy ; y++)
        {
            DibSetPixel (hdibDst, x, cy - 1 - y, DibGetPixel (hdibSrc, x, y)) ;
        }
}

```

```

        return hdibDst ;
    }

/*-----
-
    DibRotateRight: Calls optimized DibSetPixelx and DibGetPixelx
-----
-*/

HDIB DibRotateRight (HDIB hdibSrc)
{
    HDIB  hdibDst ;
    int   cx, cy, x, y ;

    if (!DibIsAddressable (hdibSrc))
        return NULL ;

    if (NULL == (hdibDst = DibCopy (hdibSrc, TRUE)))
        return NULL ;

    cx = DibWidth (hdibSrc) ;
    cy = DibHeight (hdibSrc) ;

    switch (DibBitCount (hdibSrc))
    {
    case      1:
        for ( x = 0 ; x < cx ; x++)
        for ( y = 0 ; y < cy ; y++)
            DibSetPixel1 (hdibDst, cy - y - 1, x,
                DibGetPixel1 (hdibSrc, x, y)) ;
        break ;

    case      4:
        for ( x = 0 ; x < cx ; x++)
        for ( y = 0 ; y < cy ; y++)
            DibSetPixel4 (hdibDst, cy - y - 1, x,
                DibGetPixel4 (hdibSrc, x, y)) ;
        break ;

    case      8:
        for ( x = 0 ; x < cx ; x++)
        for ( y = 0 ; y < cy ; y++)
            DibSetPixel8 (hdibDst, cy - y - 1, x,
                DibGetPixel8 (hdibSrc, x, y)) ;
        break ;

    case     16:
        for (x = 0 ; x < cx ; x++)

```

```

        for (y = 0 ; y < cy ; y++)
            DibSetPixel16 (hdibDst, cy - y - 1, x,
                DibGetPixel16 (hdibSrc, x, y)) ;
        break ;

case    24:
        for (x = 0 ; x < cx ; x++)
        for (y = 0 ; y < cy ; y++)
            DibSetPixel24 (hdibDst, cy - y - 1, x,
                DibGetPixel24 (hdibSrc, x, y)) ;
        break ;

case    32:
        for (x = 0 ; x < cx ; x++)
        for (y = 0 ; y < cy ; y++)
            DibSetPixel32 (hdibDst, cy - y - 1, x,
                DibGetPixel32 (hdibSrc, x, y)) ;
        break ;
    }
    return hdibDst ;
}

/*-----
    PaletteMenu: Uncheck and check menu item on palette menu
-----*/

void PaletteMenu (HMENU hMenu, WORD wItemNew)
{
    static WORD wItem = IDM_PAL_NONE ;
    CheckMenuItem (hMenu, wItem, MF_UNCHECKED) ;
    wItem = wItemNew ;
    CheckMenuItem (hMenu, wItem, MF_CHECKED) ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static      BOOL                fHalftonePalette ;
    static      DOCINFO              di                =
{sizeof(DOCINFO),TEXT("Dibble:Printing")} ;
    static      HBITMAP              hBitmap ;
    static      HDIB                 hdib ;
    static      HMENU                hMenu ;
    static      HPALETTE              hPalette ;
    static      int                  cxClient, cyClient, iVscroll,
iHscroll ;
    static      OPENFILENAME          ofn ;
    static      PRINTDLG              printdlg = { sizeof (PRINTDLG) } ;

```

```

        static TCHAR szFileName [MAX_PATH],
szTitleName [MAX_PATH] ;
        static TCHAR szFilter[] = TEXT ("Bitmap
Files (*.BMP)\0*.bmp\0")
        TEXT ("All Files (*.*)\0*.*\0\0") ;
static TCHAR * szCompression[] = {
        TEXT("BI_RGB"),TEXT("BI_RLE8"),TEXT("BI_RLE4"),
        TEXT("BI_BITFIELDS"),TEXT("Unknown")} ;
static WORD wShow = IDM_SHOW_NORMAL ;
BOOL fSuccess ;
BYTE * pGlobal ;
HDC hdc, hdcPrn ;
HGLOBAL hGlobal ;
HDIB hdibNew ;
int iEnable, cxPage, cyPage, iConvert ;
PAINTSTRUCT ps ;
SCROLLINFO si ;
TCHAR szBuffer [256] ;

switch (message)
{
case WM_CREATE:

        // Save the menu handle in a static variable

        hMenu = GetMenu (hwnd) ;

        // Initialize the OPENFILENAME structure for the File Open
        // and File Save dialog boxes.

        ofn.lStructSize = sizeof (OPENFILENAME) ;
        ofn.hwndOwner = hwnd ;
        ofn.hInstance = NULL ;
        ofn.lpstrFilter = szFilter ;
        ofn.lpstrCustomFilter = NULL ;
        ofn.nMaxCustFilter = 0 ;
        ofn.nFilterIndex = 0 ;
        ofn.lpstrFile = szFileName ;
        ofn.nMaxFile = MAX_PATH ;
        ofn.lpstrFileTitle = szTitleName ;
        ofn.nMaxFileTitle = MAX_PATH ;
        ofn.lpstrInitialDir = NULL ;
        ofn.lpstrTitle = NULL ;
        ofn.Flags = OFN_OVERWRITEPROMPT ;
        ofn.nFileOffset = 0 ;
        ofn.nFileExtension = 0 ;
        ofn.lpstrDefExt = TEXT ("bmp") ;
        ofn.lCustData = 0 ;

```

```

        ofn.lpfHook          = NULL ;
        ofn.lpTemplateName  = NULL ;
        return 0 ;

case WM_DISPLAYCHANGE:
    SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
    SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
    return 0 ;

case WM_SIZE:
    // Save the client area width and height in static
variables.

    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    wParam = FALSE ;
    // fall through
    // WM_USER_SETSCROLLS: Programmer-defined Message!
    // Set the scroll bars. If the display mode is not normal,
    // make them invisible. If wParam is TRUE, reset the
    // scroll bar position.

case WM_USER_SETSCROLLS:
    if (hdib == NULL || wShow != IDM_SHOW_NORMAL)
    {
        si.cbSize          =          sizeof
(SCROLLINFO) ;

        si.fMask           = SIF_RANGE ;
        si.nMin            = 0 ;
        si.nMax            = 0 ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    }
    else
    {
        // First the vertical scroll

        si.cbSize          = sizeof (SCROLLINFO) ;
        si.fMask            = SIF_ALL ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;
        si.nMin             = 0 ;
        si.nMax             = DibHeight (hdib) ;
        si.nPage            = cyClient ;
        if ((BOOL) wParam)
            si.nPos = 0 ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;
    }

```

```

        iVscroll = si.nPos ;

        // Then the horizontal scroll

        GetScrollInfo (hwnd, SB_HORZ, &si) ;
        si.nMin          = 0 ;
        si.nMax          = DibWidth (hdib) ;
        si.nPage         = cxClient ;

        if ((BOOL) wParam)
            si.nPos = 0 ;
        SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
        GetScrollInfo (hwnd, SB_HORZ, &si) ;
        iHscroll = si.nPos ;
    }
    return 0 ;

    // WM_VSCROLL: Vertically scroll the DIB

case WM_VSCROLL:
    si.cbSize = sizeof (SCROLLINFO) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;

    iVscroll = si.nPos ;

    switch (LOWORD (wParam))
    {
    case SB_LINEUP:    si.nPos - = 1 ;          break ;
    case SB_LINEDOWN:  si.nPos + = 1 ;          break ;
    case SB_PAGEUP:    si.nPos - = si.nPage ;break ;
    case SB_PAGEDOWN:  si.nPos + = si.nPage ;break ;
    case SB_THUMBTRACK:si.nPos          = si.nTrackPos ;break ;
    default:
    break ;
    }

    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    if (si.nPos != iVscroll)
    {
        ScrollWindow (hwnd, 0, iVscroll - si.nPos, NULL, NULL) ;
        iVscroll = si.nPos ;
        UpdateWindow (hwnd) ;
    }

    return 0 ;

```



```

        // WM_HSCROLL: Horizontally scroll the DIB

case WM_HSCROLL:
    si.cbSize = sizeof (SCROLLINFO) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

    iHscroll = si.nPos ;

    switch (LOWORD (wParam))
    {
    case SB_LINELEFT: si.nPos -=1 ; break ;
    case SB_LINERIGHT: si.nPos +=1 ; break ;
    case SB_PAGELEFT: si.nPos -=si.nPage ;break ;
    case SB_PAGERIGHT: si.nPos +=si.nPage ;break ;
    case SB_THUMBTRACK:si.nPos =si.nTrackPos ;break ;
    default: break ;
    }

    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

    if (si.nPos != iHscroll)
    {
        ScrollWindow (hwnd, iHscroll - si.nPos, 0, NULL, NULL) ;
        iHscroll = si.nPos ;
        UpdateWindow (hwnd) ;
    }
    return 0 ;

// WM_INITMENUPOPUP: Enable or Gray menu items

case WM_INITMENUPOPUP:
    if (hdib)
        iEnable = MF_ENABLED ;
    else
        iEnable = MF_GRAYED ;

    EnableMenuItem (hMenu, IDM_FILE_SAVE, iEnable) ;
    EnableMenuItem (hMenu, IDM_FILE_PRINT, iEnable) ;
    EnableMenuItem (hMenu, IDM_FILE_PROPERTIES, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_DELETE, iEnable) ;

    if (DibIsAddressable (hdib))
        iEnable = MF_ENABLED ;
    else

```

```

        iEnable = MF_GRAYED ;

EnableMenuItem (hMenu, IDM_EDIT_ROTATE,    iEnable) ;
EnableMenuItem (hMenu, IDM_EDIT_FLIP,      iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_01,     iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_04,     iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_08,     iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_16,     iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_24,     iEnable) ;
EnableMenuItem (hMenu, IDM_CONVERT_32,     iEnable) ;

switch (DibBitCount (hdib))
{
    case 1:  EnableMenuItem (hMenu, IDM_CONVERT_01, MF_GRAYED) ;
             break ;
    case 4:  EnableMenuItem (hMenu, IDM_CONVERT_04, MF_GRAYED) ;
             break ;
    case 8:  EnableMenuItem (hMenu, IDM_CONVERT_08, MF_GRAYED) ;
             break ;
    case 16: EnableMenuItem (hMenu, IDM_CONVERT_16, MF_GRAYED) ;
             break ;
    case 24: EnableMenuItem (hMenu, IDM_CONVERT_24, MF_GRAYED) ;
             break ;
    case 32: EnableMenuItem (hMenu, IDM_CONVERT_32, MF_GRAYED) ;
             break ;
}

    if (hdib && DibColorSize (hdib) > 0)
        iEnable = MF_ENABLED ;
    else
        iEnable = MF_GRAYED ;

    EnableMenuItem (hMenu, IDM_PAL_DIBTABLE,    iEnable) ;
    if (DibIsAddressable (hdib) && DibBitCount (hdib) > 8)
        iEnable = MF_ENABLED ;
    else
        iEnable = MF_GRAYED ;

    EnableMenuItem (hMenu, IDM_PAL_OPT_POP4,
iEnable) ;
    EnableMenuItem (hMenu, IDM_PAL_OPT_POP5,
iEnable) ;
    EnableMenuItem (hMenu, IDM_PAL_OPT_POP6,
iEnable) ;
    EnableMenuItem (hMenu, IDM_PAL_OPT_MEDCUT,
iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_PASTE,
        IsClipboardFormatAvailable (CF_DIB) ? MF_ENABLED : MF_GRAYED) ;

```

```
        return 0 ;

        // WM_COMMAND: Process all menu commands.

case WM_COMMAND:
    iConvert = 0 ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

        // Show the File Open dialog box

        if (!GetOpenFileName (&ofn))
            return 0 ;

        // If there's an existing DIB and palette, delete them

        SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;

        // Load the DIB into memory

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        hdib = DibFileLoad (szFileName) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        // Reset the scroll bars

        SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;

        // Create the palette and DDB

        SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;

        if (!hdib)
        {
            MessageBox (hwnd, TEXT ("Cannot load DIB file!"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
        }
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

case IDM_FILE_SAVE:
```

```

        // Show the File Save dialog box

        if (! GetSaveFileName (&ofn))
            return 0 ;

        // Save the DIB to memory

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        fSuccess = DibFileSave (hdib, szFileName) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        if (!fSuccess)
        MessageBox ( hwnd, TEXT ("Cannot save DIB file!"),
            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            return 0 ;

        case IDM_FILE_PRINT:
            if (!hdib)
                return 0 ;

                // Get printer DC

            printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS |
PD_NOSELECTION ;

            if (!PrintDlg (&printdlg))
                return 0 ;

            if (NULL == (hdcPrn = printdlg.hDC))
            {
                MessageBox( hwnd, TEXT ("Cannot obtain Printer DC"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    return 0 ;
            }

            // Check if the printer can print bitmaps

            if (!(RC_BITBLT & GetDeviceCaps (hdcPrn, RASTERCAPS)))
            {
                DeleteDC (hdcPrn) ;
                MessageBox (  hwnd, TEXT ("Printer cannot print bitmaps"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    return 0 ;
            }

```

```

        // Get size of printable area of page

        cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
        cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

        fSuccess = FALSE ;

        // Send the DIB to the printer

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
{
    DisplayDib (hdcPrn, DibBitmapHandle (hdib), 0, 0,
        cxPage, cyPage, wShow, FALSE) ;

        if (EndPage (hdcPrn) > 0)
        {
            fSuccess = TRUE ;
            EndDoc (hdcPrn) ;
        }
    }
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    DeleteDC (hdcPrn) ;

    if ( !fSuccess)
        MessageBox ( hwnd, TEXT ("Could not print bitmap"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

case  IDM_FILE_PROPERTIES:
    if (!hdib)
        return 0 ;

    wsprintf (szBuffer, TEXT ("Pixel width:\t%i\n")
        TEXT ("Pixel height:\t%i\n")
        TEXT ("Bits per pixel:\t%i\n")
        TEXT ("Number of colors:\t%i\n")
        TEXT ("Compression:\t%s\n"),
        DibWidth (hdib), DibHeight (hdib),
        DibBitCount (hdib), DibNumColors (hdib),
        szCompression [min (3, DibCompression (hdib))]) ;

    MessageBox (      hwnd, szBuffer, szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;

```

```
        return 0 ;

    case  IDM_APP_EXIT:
        SendMessage (hwnd, WM_CLOSE, 0, 0) ;
        return 0 ;

    case  IDM_EDIT_COPY:
    case  IDM_EDIT_CUT:
        if (!(hGlobal = DibCopyToPackedDib (hdib, TRUE)))
            return 0 ;

        OpenClipboard (hwnd) ;
        EmptyClipboard () ;
        SetClipboardData (CF_DIB, hGlobal) ;
        CloseClipboard () ;

        if (LOWORD (wParam) == IDM_EDIT_COPY)
            return 0 ;
        // fall through for IDM_EDIT_CUT
    case  IDM_EDIT_DELETE:
        SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case  IDM_EDIT_PASTE:
        OpenClipboard (hwnd) ;

        hGlobal = GetClipboardData (CF_DIB) ;
        pGlobal = GlobalLock (hGlobal) ;

        // If there's an existing DIB and palette, delete them.
        // Then convert the packed DIB to an HDIB.

        if (pGlobal)
        {
            SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
            hdib = DibCopyFromPackedDib ((BITMAPINFO *) pGlobal) ;
            SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
        }

        GlobalUnlock (hGlobal) ;
        CloseClipboard () ;
        // Reset the scroll bars

        SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;
```

```
case IDM_EDIT_ROTATE:
    if (hdibNew = DibRotateRight (hdib))
    {
        DibDelete (hdib) ;
        DeleteObject (hBitmap) ;
        hdib = hdibNew ;
        hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
        SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    else
    {
        MessageBox (    hwnd, TEXT ("Not enough memory"),
            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }
    return 0 ;

case IDM_EDIT_FLIP:
    if (hdibNew = DibFlipHorizontal (hdib))
    {
        DibDelete (hdib) ;
        DeleteObject (hBitmap) ;
        hdib = hdibNew ;
        hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    else
    {
        MessageBox (    hwnd, TEXT ("Not enough memory"),
            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }
    return 0 ;

case IDM_SHOW_NORMAL:
case IDM_SHOW_CENTER:
case IDM_SHOW_STRETCH:
case IDM_SHOW_ISOSTRETCH:
    CheckMenuItem (hMenu, wShow, MF_UNCHECKED) ;
    wShow = LOWORD (wParam) ;
    CheckMenuItem (hMenu, wShow, MF_CHECKED) ;
    SendMessage (hwnd, WM_USER_SETSCROLLS, FALSE, 0) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case IDM_CONVERT_32: iConvert += 8 ;
case IDM_CONVERT_24: iConvert += 8 ;
case IDM_CONVERT_16: iConvert += 8 ;
```

```
case IDM_CONVERT_08: iConvert += 4 ;
case IDM_CONVERT_04: iConvert += 3 ;
case IDM_CONVERT_01: iConvert += 1 ;
                    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                    ShowCursor (TRUE) ;

                    hdibNew = DibConvert (hdib, iConvert) ;

                    ShowCursor (FALSE) ;
                    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                    if (hdibNew)
                    {
SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
                    hdib = hdibNew ;
SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
                    InvalidateRect (hwnd, NULL, TRUE) ;
                    }
                    else
                    {
                    MessageBox (    hwnd, TEXT ("Not enough memory"),
szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                    }
                    return 0 ;

case IDM_APP_ABOUT:
MessageBox ( hwnd, TEXT ("Dibble (c) Charles Petzold, 1998"),
szAppName, MB_OK | MB_ICONEXCLAMATION) ;
return 0 ;
}

// All the other WM_COMMAND messages are from the palette
// items. Any existing palette is deleted, and the cursor
// is set to the hourglass.

SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
ShowCursor (TRUE) ;
// Notice that all messages for palette items are ended
// with break rather than return. This is to allow
// additional processing later on.

switch (LOWORD (wParam))
{
case IDM_PAL_DIBTABLE:
    hPalette = DibPalDibTable (hdib) ;
    break ;
```



```
        case IDM_PAL_HALFTONE:
            hdc = GetDC (hwnd) ;

            if (hPalette = CreateHalftonePalette (hdc))
                fHalftonePalette = TRUE ;

            ReleaseDC (hwnd, hdc) ;
            break ;

        case IDM_PAL_ALLPURPOSE:
            hPalette = DibPalAllPurpose () ;
            break ;

    case IDM_PAL_GRAY2:hPalette      = DibPalUniformGrays    (2)      ;
    break;
    case IDM_PAL_GRAY3:hPalette      = DibPalUniformGrays    (3)      ;
    break;
    case IDM_PAL_GRAY4:hPalette      = DibPalUniformGrays    (4)      ;
    break;
    case IDM_PAL_GRAY8:hPalette      = DibPalUniformGrays    (8)      ;
    break;
    case IDM_PAL_GRAY16:hPalette     = DibPalUniformGrays    (16)     ;
    break;
    case IDM_PAL_GRAY32:hPalette     = DibPalUniformGrays    (32)     ;
    break;
    case IDM_PAL_GRAY64:hPalette     = DibPalUniformGrays    (64)     ;
    break;
    case IDM_PAL_GRAY128:hPalette    = DibPalUniformGrays    (128)    ;
    break;
    case IDM_PAL_GRAY256:hPalette    = DibPalUniformGrays    (256)    ;
    break;
    case IDM_PAL_RGB222:hPalette     = DibPalUniformColors    (2,2,2) ;
    break;
    case IDM_PAL_RGB333:hPalette     = DibPalUniformColors    (3,3,3) ;
    break;
    case IDM_PAL_RGB444:hPalette     = DibPalUniformColors    (4,4,4) ;
    break;
    case IDM_PAL_RGB555:hPalette     = DibPalUniformColors    (5,5,5) ;
    break;
    case IDM_PAL_RGB666:hPalette     = DibPalUniformColors    (6,6,6) ;
    break;
    case IDM_PAL_RGB775:hPalette     = DibPalUniformColors    (7,7,5) ;
    break;
    case IDM_PAL_RGB757:hPalette     = DibPalUniformColors    (7,5,7) ;
    break;
    case IDM_PAL_RGB577:hPalette     = DibPalUniformColors    (5,7,7) ;
    break;
    case IDM_PAL_RGB884:hPalette     = DibPalUniformColors    (8,8,4) ;
```

```

break;
case IDM_PAL_RGB848:hPalette      =      DibPalUniformColors      (8,4,8);
break;
case IDM_PAL_RGB488:hPalette      =      DibPalUniformColors      (4,8,8);
break;
case IDM_PAL_OPT_POP4:hPalette    = DibPalPopularity (hdib, 4) ; break ;
case IDM_PAL_OPT_POP5:hPalette    = DibPalPopularity (hdib, 5) ; break ;
case IDM_PAL_OPT_POP6:hPalette    = DibPalPopularity (hdib, 6) ; break ;
case IDM_PAL_OPT_MEDCUT:hPalette = DibPalMedianCut  (hdib, 6) ; break ;
        }

        // After processing Palette items from the menu, the cursor
        //   is restored to an arrow, the menu item is checked, and
        //   the window is invalidated.

        hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        if (hPalette)
                PaletteMenu (hMenu, (LOWORD (wParam))) ;

        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

        // This programmer-defined message deletes an existing DIB
        //   in preparation for getting a new one.  Invoked during
        //   File Open command, Edit Paste command, and others.

case WM_USER_DELETEDIB:
        if (hdib)
        {
                                DibDelete (hdib) ;
                                hdib = NULL ;
        }
        SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
        return 0 ;

        // This programmer-defined message deletes an existing palette
        //   in preparation for defining a new one.

case WM_USER_DELETEPAL:
        if (hPalette)
        {
                                DeleteObject (hPalette) ;
                                hPalette = NULL ;
                                fHalftonePalette = FALSE ;

```

```
        PaletteMenu (hMenu, IDM_PAL_NONE) ;
    }
    if (hBitmap)
        DeleteObject (hBitmap) ;

    return 0 ;

// Programmer-defined message to create a new palette based on
// a new DIB.  If wParam == TRUE, create a DDB as well.

case WM_USER_CREATEPAL:
    if (hdib)
    {
        hdc = GetDC (hwnd) ;

        if (!(RC_PALETTE & GetDeviceCaps (hdc, RASTERCAPS)))
        {
            PaletteMenu (hMenu, IDM_PAL_NONE) ;
        }
        else if (hPalette = DibPalDibTable (hdib))
        {
            PaletteMenu (hMenu, IDM_PAL_DIBTABLE) ;
        }
        else if (hPalette = CreateHalftonePalette (hdc))
        {
            fHalftonePalette = TRUE ;
            PaletteMenu (hMenu, IDM_PAL_HALFTONE) ;
        }
        ReleaseDC (hwnd, hdc) ;

        if ((BOOL) wParam)
            hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
    }
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
    }
    if (hBitmap)
    {
        DisplayDib (hdc,
            fHalftonePalette ? DibBitmapHandle (hdib) : hBitmap,
            iHscroll, iVscroll,
```

```
        cxClient, cyClient,
        wShow, fHalftonePalette) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_QUERYNEWPALETTE:
    if (!hPalette)
        return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
        break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

case WM_DESTROY:
    if (hdib)
        DIBDelete (hdib) ;

    if (hBitmap)
        DeleteObject (hBitmap) ;

    if (hPalette)
        DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

DIBBLE.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
```

```

#include "afxres.h"
/////////////////////////////////////////////////////////////////
/
// Menu
DIBBLE MENU DISCARDABLE BEGIN POPUP "&File"
    BEGIN
        MENUITEM "&Open...\tCtrl+O",          IDM_FILE_OPEN
        MENUITEM "&Save...\tCtrl+S",          IDM_FILE_SAVE
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",          IDM_FILE_PRINT
        MENUITEM SEPARATOR
        MENUITEM "Propert&ies...",            IDM_FILE_PROPERTIES
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                      IDM_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "Cu&t\tCtrl+X",              IDM_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",              IDM_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",            IDM_EDIT_PASTE
        MENUITEM "&Delete\tDelete",          IDM_EDIT_DELETE
        MENUITEM SEPARATOR
        MENUITEM "&Flip",                    IDM_EDIT_FLIP
        MENUITEM "&Rotate",                  IDM_EDIT_ROTATE
    END
    POPUP "&Show"
    BEGIN
        MENUITEM "&Actual Size",            IDM_SHOW_NORMAL, CHECKED
        MENUITEM "&Center",                IDM_SHOW_CENTER
        MENUITEM "&Stretch to Window",        IDM_SHOW_STRETCH
        MENUITEM "Stretch &Isotropically",    IDM_SHOW_ISOSTRETCH
    END
    POPUP "&Palette"
    BEGIN
        MENUITEM "&None",                    IDM_PAL_NONE, CHECKED
        MENUITEM "&Dib ColorTable",          IDM_PAL_DIBTABLE
        MENUITEM "&Halftone",                IDM_PAL_HALFTONE
        MENUITEM "&All-Purpose",              IDM_PAL_ALLPURPOSE
    POPUP "&Gray Shades"
        BEGIN
            MENUITEM "&1. 2 Grays",          IDM_PAL_GRAY2
            MENUITEM "&2. 3 Grays",          IDM_PAL_GRAY3
            MENUITEM "&3. 4 Grays",          IDM_PAL_GRAY4
            MENUITEM "&4. 8 Grays",          IDM_PAL_GRAY8
            MENUITEM "&5. 16 Grays",         IDM_PAL_GRAY16
            MENUITEM "&6. 32 Grays",         IDM_PAL_GRAY32
            MENUITEM "&7. 64 Grays",         IDM_PAL_GRAY64
            MENUITEM "&8. 128 Grays",        IDM_PAL_GRAY128

```

```

    MENUITEM "&9. 256 Grays",          IDM_PAL_GRAY256
        END
        POPUP "&Uniform Colors"
    BEGIN
        MENUITEM "&1. 2R x 2G x 2B (8)",          IDM_PAL_RGB222
        MENUITEM "&2. 3R x 3G x 3B (27)",          IDM_PAL_RGB333
        MENUITEM "&3. 4R x 4G x 4B (64)",          IDM_PAL_RGB444
        MENUITEM "&4. 5R x 5G x 5B (125)",          IDM_PAL_RGB555
        MENUITEM "&5. 6R x 6G x 6B (216)",          IDM_PAL_RGB666
        MENUITEM "&6. 7R x 7G x 5B (245)",          IDM_PAL_RGB775
            MENUITEM "&7. 7R x 5B x 7B (245)",          IDM_PAL_RGB757
        MENUITEM "&8. 5R x 7G x 7B (245)",          IDM_PAL_RGB577
        MENUITEM "&9. 8R x 8G x 4B (256)",          IDM_PAL_RGB884
        MENUITEM "&A. 8R x 4G x 8B (256)",          IDM_PAL_RGB848
    MENUITEM "&B. 4R x 8G x 8B (256)",          IDM_PAL_RGB488
        END
        POPUP "&Optimized"
    BEGIN
        MENUITEM "&1. Popularity Algorithm (4 bits)"IDM_PAL_OPT_POP4
        MENUITEM "&2. Popularity Algorithm (5 bits)"IDM_PAL_OPT_POP5
        MENUITEM "&3. Popularity Algorithm (6 bits)"IDM_PAL_OPT_POP6
        MENUITEM "&4. Median Cut Algorithm ", IDM_PAL_OPT_MEDCUT
    END
END
POPUP "Con&vert"
BEGIN
    MENUITEM "&1. to 1 bit per pixel",          IDM_CONVERT_01
    MENUITEM "&2. to 4 bits per pixel",          IDM_CONVERT_04
    MENUITEM "&3. to 8 bits per pixel",          IDM_CONVERT_08
    MENUITEM "&4. to 16 bits per pixel",          IDM_CONVERT_16
    MENUITEM "&5. to 24 bits per pixel",          IDM_CONVERT_24
    MENUITEM "&6. to 32 bits per pixel",          IDM_CONVERT_32
END
POPUP "&Help"
BEGIN
    MENUITEM "&About",
        IDM_APP_ABOUT
END
END
////////////////////////////////////
/
// Accelerator
DIBBLE ACCELERATORS DISCARDABLE
BEGIN
    "C",          IDM_EDIT_COPY,          VIRTKEY, CONTROL, NOINVERT
    "O",          IDM_FILE_OPEN,          VIRTKEY, CONTROL, NOINVERT
    "P",          IDM_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
    "S",          IDM_FILE_SAVE,          VIRTKEY, CONTROL, NOINVERT

```

```
"V",      IDM_EDIT_PASTE,      VIRTKEY, CONTROL, NOINVERT
VK_DELETE,      IDM_EDIT_DELETE,      VIRTKEY, NOINVERT
"X",      IDM_EDIT_CUT,      VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
```

```
// Used by Dibble.rc
```

```
#define IDM_FILE_OPEN      40001
#define IDM_FILE_SAVE      40002
#define IDM_FILE_PRINT     40003
#define IDM_FILE_PROPERTIES 40004
#define IDM_APP_EXIT       40005
#define IDM_EDIT_CUT       40006
#define IDM_EDIT_COPY      40007
#define IDM_EDIT_PASTE     40008
#define IDM_EDIT_DELETE    40009
#define IDM_EDIT_FLIP      40010
#define IDM_EDIT_ROTATE    40011
#define IDM_SHOW_NORMAL    40012
#define IDM_SHOW_CENTER    40013
#define IDM_SHOW_STRETCH   40014
#define IDM_SHOW_ISOSTRETCH 40015
#define IDM_PAL_NONE       40016
#define IDM_PAL_DIBTABLE   40017
#define IDM_PAL_HALFTONE   40018
#define IDM_PAL_ALLPURPOSE 40019
#define IDM_PAL_GRAY2      40020
#define IDM_PAL_GRAY3      40021
#define IDM_PAL_GRAY4      40022
#define IDM_PAL_GRAY8      40023
#define IDM_PAL_GRAY16     40024
#define IDM_PAL_GRAY32     40025
#define IDM_PAL_GRAY64     40026
#define IDM_PAL_GRAY128    40027
#define IDM_PAL_GRAY256    40028
#define IDM_PAL_RGB222     40029
#define IDM_PAL_RGB333     40030
#define IDM_PAL_RGB444     40031
#define IDM_PAL_RGB555     40032
#define IDM_PAL_RGB666     40033
#define IDM_PAL_RGB775     40034
#define IDM_PAL_RGB757     40035
#define IDM_PAL_RGB577     40036
#define IDM_PAL_RGB884     40037
#define IDM_PAL_RGB848     40038
#define IDM_PAL_RGB488     40039
#define IDM_PAL_OPT_POP4    40040
```

```

#define IDM_PAL_OPT_POP5          40041
#define IDM_PAL_OPT_POP6          40042
#define IDM_PAL_OPT_MEDCUT        40043
#define IDM_CONVERT_01            40044
#define IDM_CONVERT_04            40045
#define IDM_CONVERT_08            40046
#define IDM_CONVERT_16            40047
#define IDM_CONVERT_24            40048
#define IDM_CONVERT_32            40049
#define IDM_APP_ABOUT             40050

```

DIBBLE 使用了两个其他档案，我将简要地说明它们。DIBCONV 档案 (DIBCONV.C 和 DIBCONV.H) 在两种不同格式之间转换——例如，从每图素 24 位元转换成每图素 8 位元。DIBPAL 档案 (DIBPAL.C 和 DIBPAL.H) 建立调色盘。

DIBBLE 维护 WndProc 中的三个重要的静态变数。这些是呼叫 hdib 的 HDIB 代号、呼叫 hPalette 的 HPALETTE 代号和呼叫 hBitmap 的 HBITMAP 代号。HDIB 来自 DIBHELP 中的不同函式；HPALETTE 来自 DIBPAL 中的不同函式或 CreateHalftonePalette 函式；而 HBITMAP 代号来自 DIBHELP.C 中的 DibCopyToDdb 函式并帮助加速萤幕显示，特别是在 256 色显示模式下。不过，无论在程式建立新的「DIB Section」（显而易见地）或在程式建立不同的调色盘（不很明显）时，这个代号都必须重新建立。

让我们从功能上而非循序渐进地来介绍一下 DIBBLE。

档案载入和储存

DIBBLE 可以在回应 IDM_FILE_LOAD 和 IDM_FILE_SAVE 的 WM_COMMAND 讯息处理过程中载入 DIB 档案并储存这些档案。在处理这些讯息处理期间，DIBBLE 通过分别呼叫 GetOpenFileName 和 GetSaveFileName 来启动公用档案对话方块。

对于「File」、「Save」功能表命令，DIBBLE 只需要呼叫 DibFileSave。对于「File」、「Open」功能表命令，DIBBLE 必须首先删除前面的 HDIB、调色盘和点阵图物件。它透过发送一个 WM_USER_DELETEDIB 讯息来完成这件事，此讯息通过呼叫 DibDelete 和 DeleteObject 来处理。然后 DIBBLE 呼叫 DIBHELP 中的 DibFileLoad 函式，发送 WM_USER_SETSCROLLS 和 WM_USER_CREATEPAL 讯息来重新设定卷动列并建立调色盘。WM_USER_CREATEPAL 讯息也位于程式从 DIB 区块建立的新的 DDB 位置。

显示、卷动和列印

DIBBLE 的功能表允许它以实际尺寸在显示区域左上角显示 DIB，或在显示区域中间显示 DIB，或伸展到填充显示区域，或者在保持纵横比的情况下尽量填

充显示区域。您可以在 DIBBLE 的「Show」功能表上来选择需要的选项。注意，这些与上一章的 SHOWDIB2 程式中四个选项相同。

在 WM_PAINT 讯息处理期间——也是处理「File」、「Print」命令的过程中——DIBBLE 呼叫 DisplayDib 函式。注意，DisplayDib 使用 BitBlt 和 StretchBlt，而不是使用 SetDIBitsToDevice 和 StretchDIBits。在 WM_PAINT 讯息处理期间，传递给函式的点阵图代号由 DibCopyToDdb 函式建立，并在 WM_USER_CREATEPAL 讯息处理期间呼叫。其中 DDB 与视讯装置内容相容。当处理「File」、「Print」命令时，DIBBLE 呼叫 DisplayDib，其中可用的 DIB 区块代号来自 DIBHELP.C 中的 DibBitmapHandle 函式。

另外要注意，DIBBLE 保留一个称作 fHalftonePalette 的静态 BOOL 变数，如果从 CreateHalftonePalette 函式中获得 hPalette，则此变数设定为 TRUE。这将迫使 DisplayDib 函式呼叫 StretchBlt 而不是呼叫 BitBlt，即使 DIB 被指定按实际尺寸显示。fHalftonePalette 变数也导致 WM_PAINT 处理程式将 DIB 区块代号传递给 DisplayDib 函式，而不是由 DibCopyToDdb 函式建立的点阵图代号。本章前面讨论过中间色调色盘的使用，并在 SHOWDIB5 程式中进行了展示。

第一次使用范例程式时，DIBBLE 允许在显示区域中卷动 DIB。只有按实际尺寸显示 DIB 时，才显示卷动列。在处理 WM_PAINT 时，WndProc 简单地将卷动列的目前位置传递给 DisplayDib 函式。

剪贴簿

对于「Cut」和「Copy」功能表项，DIBBLE 呼叫 DIBHELP 中的 DibCopyToPackedDib 函式。该函式将获得所有的 DIB 元件并将它们放入大的记忆体块中。

对于第一次使用本书中的某些范例程式来说，DIBBLE 从剪贴簿中粘贴 DIB。这包括呼叫 DibCopyFromPackedDib 函式，并替换视窗讯息处理程式前面储存的 HDIB、调色盘和点阵图。

翻转和旋转

DIBBLE 中的「Edit」功能表中除了常见的「Cut」、「Copy」、「Paste」和「Delete」选项之外，还包括两个附加项——「Flip」和「Rotate」。「Flip」选项使点阵图绕水平轴翻转——即上下颠倒翻转。「Rotate」选项使点阵图顺时针旋转 90 度。这两个函式都需要透过将它们从一个 DIB 复制到另一个来存取所有的 DIB 图素（因为这两个函式不需要建立新的调色盘，所以不删除和重新建立调色盘）。

「Flip」功能表选项使用 DibFlipHorizontal 函式, 此函式也位於 DIBBLE.C 档案。此函式呼叫 DibCopy 来获得 DIB 精确的副本。然後, 进入将原 DIB 中的图素复制到新 DIB 的回圈, 但是复制这些图素是为了上下翻转图像。注意, 此函式呼叫 DibGetPixel 和 DibSetPixel。这些是 DIBHELP.C 中的通用 (但不像我们所希望的那么快) 函式。

为了说明 DibGetPixel 和 DibSetPixel 函式与 DIBHELP.H 中执行更快的 DibGetPixel 和 DibSetPixel 巨集之间的区别, DibRotateRight 函式使用了巨集。然而, 首先要注意的是, 该函式呼叫 DibCopy 时, 第二个参数设定为 TRUE。这导致 DibCopy 翻转原 DIB 的宽度和高度来建立新的 DIB。另外, 图素位元不能由 DibCopy 函式复制。但是, DibRotateRight 函式有六个不同的回圈将图素位元从原 DIB 复制到新的 DIB——每一个都对应不同的 DIB 图素宽度 (1 位元、4 位元、8 位元、16 位元、24 位元和 32 位元)。虽然包括了更多的程式码, 但是函式更快了。

尽管可以使用「Flip Horizontal」和「Rotate Right」选项来产生「Flip Vertical」、「Rotate Left」和「Rotate 180」啊构 δ 埽 ũ 3 淌浇 笨又葱 兴 醒 ∠ 睢 1 暇梗珉 IBBLE 只是个展示程式而已。

简单调色盘：最佳化调色盘

在 DIBBLE 中, 您可以在 256 色视讯显示器上选择不同的调色盘来显示 DIB。这些都在 DIBBLE 的「Palette」功能表中列出。除了中间色调色盘以外, 其余的都直接由 Windows 函式呼叫建立, 建立不同调色盘的所有函式都由程式 16-24 所示的 DIBPAL 档案提供。

程式 16-24 DIBPAL 档案

```
DIBPAL.H
/*-----
    DIBPAL.H header file for DIBPAL.C
-----*/

HPALETTE DibPalDibTable (HDIB hdib) ;
HPALETTE DibPalAllPurpose (void) ;
HPALETTE DibPalUniformGrays (int iNum) ;
HPALETTE DibPalUniformColors (int iNumR, int iNumG, int iNumB) ;
HPALETTE DibPalVga (void) ;
HPALETTE DibPalPopularity (HDIB hdib, int iRes) ;
HPALETTE DibPalMedianCut (HDIB hdib, int iRes) ;
DIBPAL.C
/*-----
--
```

```

DIBPAL.C --          Palette-Creation Functions
                                (c) Charles Petzold, 1998

-----*/

#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"

/*-----
-
    DibPalDibTable: Creates a palette from the DIB color table
-----
-*/

HPALETTE DibPalDibTable (HDIB hdib)
{
    HPALETTE          hPalette ;
    int               i, iNum ;
    LOGPALETTE *      plp ;
    RGBQUAD           rgb ;

    if (0 == (iNum = DibNumColors (hdib)))
        return NULL ;
    plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries   = iNum ;

    for (i = 0 ; i < iNum ; i++)
    {
        DibGetColor (hdib, i, &rgb) ;
        plp->palPalEntry[i].peRed   = rgb.rgbRed ;
        plp->palPalEntry[i].peGreen = rgb.rgbGreen ;
        plp->palPalEntry[i].peBlue  = rgb.rgbBlue ;
        plp->palPalEntry[i].peFlags = 0 ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

/*-----
-
    DibPalA      llPurpose: Creates a palette suitable for a wide variety
                        of images; the palette has 247 entries, but 15 of
them are
                        duplicates or match the standard 20 colors.
-----
*/

```

```

HPALETTE DIBPalAllPurpose (void)
{
    HPALETTE          hPalette ;
    int               i, incr, R, G, B ;
    LOGPALETTE *      plp ;

    plp = malloc (sizeof (LOGPALETTE) + 246 * sizeof (PALETTEENTRY)) ;
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries    = 247 ;

    // The following loop calculates 31 gray shades, but 3
of them
    //                               will match the standard 20 colors

    for (i = 0, G = 0, incr = 8 ; G <= 0xFF ; i++, G += incr)
    {
        plp->palPalEntry[i].peRed      = (BYTE) G ;
        plp->palPalEntry[i].peGreen    = (BYTE) G ;
        plp->palPalEntry[i].peBlue     = (BYTE) G ;
        plp->palPalEntry[i].peFlags    = 0 ;

        incr = (incr == 9 ? 8 : 9) ;
    }

    // The following loop is responsible for 216 entries, but 8 of
    //                               them will match the standard 20 colors,
and another
    //                               4 of them will match the gray shades
above.

    for (R = 0 ; R <= 0xFF ; R += 0x33)
    for (G = 0 ; G <= 0xFF ; G += 0x33)
    for (B = 0 ; B <= 0xFF ; B += 0x33)
    {
        plp->palPalEntry[i].peRed      = (BYTE) R ;
        plp->palPalEntry[i].peGreen    = (BYTE) G ;
        plp->palPalEntry[i].peBlue     = (BYTE) B ;
        plp->palPalEntry[i].peFlags    = 0 ;

        i++ ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

/*-----

```

```

-
    DibPalUniformGrays:  Creates a palette of iNum grays, uniformly spaced
-----
*/

HPALETTE DibPalUniformGrays (int iNum)
{
    HPALETTE          hPalette ;
    int               i ;
    LOGPALETTE *      plp ;

    plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries   = iNum ;

    for (i = 0 ; i < iNum ; i++)
    {
        plp->palPalEntry[i].peRed      =
        plp->palPalEntry[i].peGreen    =
        plp->palPalEntry[i].peBlue     = (BYTE) (i * 255 / (iNum -
1)) ;

        plp->palPalEntry[i].peFlags = 0 ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

/*-----
    DibPalUniformColors:  Creates a palette of iNumR x iNumG x iNumB colors
-----
*/

HPALETTE DibPalUniformColors (int iNumR, int iNumG, int iNumB)
{
    HPALETTE          hPalette ;
    int               i, iNum, R, G, B ;
    LOGPALETTE *      plp ;

    iNum = iNumR * iNumG * iNumB ;
    plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
    plp->palVersion      = 0x0300 ;
    plp->palNumEntries   = iNumR * iNumG * iNumB ;

    i = 0 ;
    for (R = 0 ; R < iNumR ; R++)
    for (G = 0 ; G < iNumG ; G++)
    for (B = 0 ; B < iNumB ; B++)

```

```

    {
        plp->palPalEntry[i].peRed    = (BYTE) (R * 255 / (iNumR - 1)) ;
        plp->palPalEntry[i].peGreen  = (BYTE) (G * 255 / (iNumG - 1)) ;
        plp->palPalEntry[i].peBlue   = (BYTE) (B * 255 / (iNumB - 1)) ;
        plp->palPalEntry[i].peFlags  = 0 ;

        i++ ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

/*-----
-
    DibPalVga:  Creates a palette based on standard 16 VGA colors
-----*/

HPALETTE DibPalVga (void)
{
    static RGBQUAD rgb [16] = { 0x00, 0x00, 0x00, 0x00,
                                0x00, 0x00, 0x80, 0x00,
                                0x00, 0x80, 0x00, 0x00,
                                0x00, 0x80, 0x80, 0x00,
                                0x80, 0x00, 0x00, 0x00,
                                0x80, 0x00, 0x80, 0x00,
                                0x80, 0x80, 0x00, 0x00,
                                0x80, 0x80, 0x80, 0x00,
                                0xC0, 0xC0, 0xC0, 0x00,
                                0x00, 0x00, 0xFF, 0x00,
                                0x00, 0xFF, 0x00, 0x00,
                                0x00, 0xFF, 0xFF, 0x00,
                                0xFF, 0x00, 0x00, 0x00,
                                0xFF, 0x00, 0xFF, 0x00,
                                0xFF, 0xFF, 0x00, 0x00,
                                0xFF, 0xFF, 0xFF, 0x00 } ;

    HPALETTE hPalette ;
    int i ;
    LOGPALETTE * plp ;

    plp = malloc (sizeof (LOGPALETTE) + 15 * sizeof (PALETTEENTRY)) ;
    plp->palVersion    = 0x0300 ;
    plp->palNumEntries = 16 ;

    for (i = 0 ; i < 16 ; i++)
    {
        plp->palPalEntry[i].peRed    = rgb[i].rgbRed ;

```

```

        plp->palPalEntry[i].peGreen = rgb[i].rgbGreen ;
        plp->palPalEntry[i].peBlue  = rgb[i].rgbBlue  ;
        plp->palPalEntry[i].peFlags = 0 ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}

/*-----
-
    Macro used in palette optimization routines
-----*/

#define PACK_RGB(R,G,B,iRes) (((int) (R) |      ((int) (G) << (iRes)) |      \
                             ((int) (B) << ((iRes) + (iRes))))

/*-----
--
    AccumColorCounts: Fills up piCount (indexed by a packed RGB color)
    with counts of pixels of that color.
-----
-*/

static void AccumColorCounts (HDIB hdib, int * piCount, int iRes)
{
    int          x, y, cx, cy ;
    RGBQUAD      rgb ;

    cx = DibWidth (hdib) ;
    cy = DibHeight (hdib) ;

    for (y = 0 ; y < cy ; y++)
    for (x = 0 ; x < cx ; x++)
    {
        DibGetPixelColor (hdib, x, y, &rgb) ;

        rgb.rgbRed      >>= (8 - iRes) ;
        rgb.rgbGreen    >>= (8 - iRes) ;
        rgb.rgbBlue     >>= (8 - iRes) ;

        ++piCount [PACK_RGB (rgb.rgbRed,    rgb.rgbGreen,
rgb.rgbBlue, iRes)] ;
    }
}

/*-----

```

```

-
    DibPalPopularity:  Popularity algorithm for optimized colors
-----
-*/

HPALETTE DibPalPopularity (HDIB hdib, int iRes)
{
    HPALETTE          hPalette ;
    int               i, iArraySize, iEntry, iCount, iIndex, iMask,
R, G, B ;
    int               *    piCount ;
    LOGPALETTE *      plp ;

        // Validity checks

    if (DibBitCount (hdib) < 16)
        return NULL ;
    if (iRes < 3 || iRes > 8)
        return NULL ;
        // Allocate array for counting pixel colors
    iArraySize = 1 << (3 * iRes) ;
    iMask = (1 << iRes) - 1 ;

    if (NULL == (piCount = calloc (iArraySize, sizeof (int))))
        return NULL ;
        // Get the color counts
    AccumColorCounts (hdib, piCount, iRes) ;
        // Set up a palette
    plp = malloc (sizeof (LOGPALETTE) + 235 * sizeof (PALETTEENTRY)) ;
    plp->palVersion = 0x0300 ;
    for (iEntry = 0 ; iEntry < 236 ; iEntry++)
    {
        for (i = 0, iCount = 0 ; i < iArraySize ; i++)
            if (piCount[i] > iCount)
            {
                iCount = piCount[i] ;
                iIndex = i ;
            }
        if (iCount == 0)
            break ;
        R = (iMask & iIndex) << (8 - iRes) ;
        G = (iMask & (iIndex >> iRes)) << (8 - iRes) ;
        B = (iMask & (iIndex >> (iRes + iRes))) << (8 - iRes) ;

        plp->palPalEntry[iEntry].peRed    = (BYTE) R ;
        plp->palPalEntry[iEntry].peGreen  = (BYTE) G ;
        plp->palPalEntry[iEntry].peBlue   = (BYTE) B ;
    }
}

```



```

        plp->palPalEntry[iEntry].peFlags = 0 ;

        piCount [iIndex] = 0 ;
    }

    // On exit from the loop iEntry will be the number of stored
entries
    plp->palNumEntries = iEntry ;
    // Create the palette, clean up, and return the palette handle
    hPalette = CreatePalette (plp) ;
    free (piCount) ;
    free (plp) ;

    return hPalette ;
}

/*-----
    Structures used for implementing median cut algorithm
-----*/

typedef struct                                // defines dimension of a box
{
    int Rmin, Rmax, Gmin, Gmax, Bmin, Bmax ;
}
MINMAX ;
typedef struct                                // for Compare routine for qsort
{
    int                iBoxCount ;
    RGBQUAD            rgbBoxAv ;
}

BOXES ;
/*-----
--
    FindAverageColor: In a box
-----*/

static int FindAverageColor (    int * piCount, MINMAX mm,
                                int iRes, RGBQUAD * prgb)
{
    int R, G, B, iR, iG, iB, iTotal, iCount ;
    // Initialize some variables
    iTotal = iR = iG = iB = 0 ;
    // Loop through all colors in the box
    for (R = mm.Rmin ; R <= mm.Rmax ; R++)
    for (G = mm.Gmin ; G <= mm.Gmax ; G++)
    for (B = mm.Bmin ; B <= mm.Bmax ; B++)

```

```

{
    // Get the number of pixels of that color
    iCount = piCount [PACK_RGB (R, G, B, iRes)] ;
    // Weight the pixel count by the color value
    iR += iCount * R ;
    iG += iCount * G ;
    iB += iCount * B ;

    iTTotal += iCount ;
}

// Find the average color
prgb->rgbRed      = (BYTE) ((iR / iTTotal) << (8 - iRes)) ;
prgb->rgbGreen    = (BYTE) ((iG / iTTotal) << (8 - iRes)) ;
prgb->rgbBlue     = (BYTE) ((iB / iTTotal) << (8 - iRes)) ;

// Return the total number of pixels in the box

return iTTotal ;
}

/*-----
-
CutBox: Divide a box in two
-----*/
static void CutBox (int * piCount, int iBoxCount, MINMAX mm,
                   int iRes, int iLevel, BOXES * pboxes, int * piEntry)
{
    int          iCount, R, G, B ;
    MINMAX      mmNew ;

    // If the box is empty, return

    if (iBoxCount == 0)
        return ;

    // If the nesting level is 8, or the box is one pixel,
we're ready
    // to find the average color in the box and save it along
with
    // the number of pixels of that color

    if (iLevel == 8 || ( mm.Rmin == mm.Rmax &&
                        mm.Gmin == mm.Gmax &&
                        mm.Bmin == mm.Bmax))
    {
        pboxes[*piEntry].iBoxCount =
            FindAverageColor          (piCount,          mm,          iRes,

```

```

&pboxes[*piEntry].rgbBoxAv) ;
        (*piEntry) ++ ;
    }

    // Otherwise, if blue is the largest side, split it
    else if ((mm.Bmax - mm.Bmin > mm.Rmax - mm.Rmin) &&
             (mm.Bmax - mm.Bmin > mm.Gmax - mm.Gmin))
    {
        // Initialize a counter and loop through the blue side
        iCount = 0 ;
        for (B = mm.Bmin ; B < mm.Bmax ; B++)
        {
            // Accumulate all the pixels for each successive blue value
            for (R = mm.Rmin ; R <= mm.Rmax ; R++)
            for (G = mm.Gmin ; G <= mm.Gmax ; G++)
                iCount += piCount [PACK_RGB (R, G, B, iRes)] ;

            // If it's more than half the box count, we're there

            if (iCount >= iBoxCount / 2)
                break ;

            // If the next blue value will be the max, we're there
            if (B == mm.Bmax - 1)
                break ;
        }

        // Cut the two split boxes.
        // The second argument to CutBox is the new box count.
        // The third argument is the new min and max values.

        mmNew = mm ;
        mmNew.Bmin = mm.Bmin ;
        mmNew.Bmax = B ;

        CutBox ( piCount, iCount, mmNew, iRes, iLevel + 1,
                 pboxes, piEntry) ;

        mmNew.Bmin = B + 1 ;
        mmNew.Bmax = mm.Bmax ;

        CutBox ( piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
                 pboxes, piEntry) ;
    }

    // Otherwise, if red is the largest side, split it (just like blue)
    else if (mm.Rmax - mm.Rmin > mm.Gmax - mm.Gmin)
    {
        iCount = 0 ;
        for (R = mm.Rmin ; R < mm.Rmax ; R++)
        {

```

```

        for (B = mm.Bmin ; B <= mm.Bmax ; B++)
            for (G = mm.Gmin ; G <= mm.Gmax ; G++)
                iCount += piCount [PACK_RGB (R, G, B, iRes)] ;
        if (iCount >= iBoxCount / 2)
            break ;
        if (R == mm.Rmax - 1)
            break ;
    }

    mmNew = mm ;
    mmNew.Rmin = mm.Rmin ;
    mmNew.Rmax = R ;

    CutBox (    piCount, iCount, mmNew, iRes, iLevel + 1,
                pboxes, piEntry) ;

    mmNew.Rmin = R + 1 ;
    mmNew.Rmax = mm.Rmax ;
    CutBox (    piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
                pboxes, piEntry) ;
}

// Otherwise, split along the green size
else
{
    iCount = 0 ;
    for (G = mm.Gmin ; G < mm.Gmax ; G++)
    {
        for (B = mm.Bmin ; B <= mm.Bmax ; B++)
            for (R = mm.Rmin ; R <= mm.Rmax ; R++)
                iCount += piCount [PACK_RGB (R, G, B, iRes)] ;

        if ( iCount >= iBoxCount / 2)
            break ;

        if ( G == mm.Gmax - 1)
            break ;
    }

    mmNew = mm ;
    mmNew.Gmin = mm.Gmin ;
    mmNew.Gmax = G ;

    CutBox (    piCount, iCount, mmNew, iRes, iLevel + 1,
                pboxes, piEntry) ;

    mmNew.Gmin = G + 1 ;
    mmNew.Gmax = mm.Gmax ;

    CutBox (    piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
                pboxes, piEntry) ;
}

```

```

    }
}

/*-----
-
    Compare routine for qsort
-----
-*/

static int Compare (const BOXES * pbox1, const BOXES * pbox2)
{
    return pbox1->iBoxCount - pbox2->iBoxCount ;
}

/*-----
-
    DibPalMedianCut:  Creates palette based on median cut algorithm
-----*/
HPALETTE DibPalMedianCut (HDIB hdib, int iRes)
{
    BOXES          boxes [256] ;
    HPALETTE        hPalette ;
    int             i, iArraySize, iCount, R, G, B, iTotCount, iDim, iEntry = 0 ;
    int             * piCount ;
    LOGPALETTE *    plp ;
    MINMAX          mm ;

    // Validity checks

    if (DibBitCount (hdib) < 16)
        return NULL ;
    if (iRes < 3 || iRes > 8)
        return NULL ;
    // Accumulate counts of pixel colors
    iArraySize = 1 << (3 * iRes) ;
    if (NULL == (piCount = calloc (iArraySize, sizeof (int))))
        return NULL ;
    AccumColorCounts (hdib, piCount, iRes) ;
    // Find the dimensions of the total box
    iDim = 1 << iRes ;
    mm.Rmin = mm.Gmin = mm.Bmin = iDim - 1 ;
    mm.Rmax = mm.Gmax = mm.Bmax = 0 ;

    iTotCount = 0 ;
    for (R = 0 ; R < iDim ; R++)
    for (G = 0 ; G < iDim ; G++)
    for (B = 0 ; B < iDim ; B++)

```

```

        if ((iCount = piCount [PACK_RGB (R, G, B, iRes)]) > 0)
        {
            iTotCount += iCount ;
            if (R < mm.Rmin) mm.Rmin = R ;
            if (G < mm.Gmin) mm.Gmin = G ;
            if (B < mm.Bmin) mm.Bmin = B ;
            if (R > mm.Rmax) mm.Rmax = R ;
            if (G > mm.Gmax) mm.Gmax = G ;
            if (B > mm.Bmax) mm.Bmax = B ;
        }

        // Cut the first box (iterative function).
        // On return, the boxes structure will have up to 256 RGB values,
        //     one for each of the boxes, and the number of pixels in
        //     each box.
        // The iEntry value will indicate the number of non-empty boxes.

CutBox (piCount, iTotCount, mm, iRes, 0, boxes, &iEntry) ;
free (piCount) ;

        // Sort the RGB table by the number of pixels for each color
qsort (boxes, iEntry, sizeof (BOXES), Compare) ;
plp = malloc (sizeof (LOGPALETTE) + (iEntry - 1) * sizeof (PALETTEENTRY)) ;
if (plp == NULL)
    return NULL ;
plp->palVersion      = 0x0300 ;
plp->palNumEntries   = iEntry ;

for (i = 0 ; i < iEntry ; i++)
{
    plp->palPalEntry[i].peRed   = boxes[i].rgbBoxAv.rgbRed ;
    plp->palPalEntry[i].peGreen = boxes[i].rgbBoxAv.rgbGreen ;
    plp->palPalEntry[i].peBlue  = boxes[i].rgbBoxAv.rgbBlue ;
    plp->palPalEntry[i].peFlags = 0 ;
}

hPalette = CreatePalette (plp) ;
free (plp) ;
return hPalette ;
}

```

第一个函式——DibPalDibTable——看起来应该很熟悉。它根据 DIB 的颜色表建立了调色盘。这与本章前面的 SHOWDIB3 中所用到的 PACKEDIB.C 里的 PackedDibCreatePalette 函式相似。在 SHOWDIB3 中，只有当 DIB 有颜色表时才执行此函式。在 8 位元显示模式下试图显示 16 位元、24 位元或 32 位元 DIB 时，此函式就没用了。

预设情况下，执行在 256 色显示模式下时，DIBBLE 将首先尝试呼叫

DibPalDibTable 来根据 DIB 颜色表建立调色盘。如果 DIB 没有颜色表,则 DIBBLE 将呼叫 CreateHalftonePalette 并将 fHalftonePalette 变数设定为 TRUE。此逻辑发生在 WM_USER_CREATEPAL 讯息处理期间。

DIBPAL.C 也执行函式 DibPalAllPurpose, 因为此函式与 SHOWDIB4 中的 CreateAllPurposePalette 函式非常相似, 所以它看起来也很熟悉。您也可以从 DIBBLE 功能表中选择此调色盘。

在 256 色模式下显示点阵图最有趣的是, 您可以直接控制 Windows 用於显示图像的颜色。如果您选择并显现调色盘, 则 Windows 将使用此调色盘中的颜色, 而不是其他调色盘中的颜色。

例如, 您可以用 DibPalUniformGrays 函式来单独建立一种灰阶调色盘。使用两种灰阶的调色盘则只含有 00-00-00 (黑色) 和 FF-FF-FF (白色)。用此调色盘来输出图像将提供某些照片中常用的高对比「黑白」效果。使用 3 种灰阶将在黑色和白色中间添加中间灰色, 使用 4 种灰阶将添加 2 种灰阶。

用 8 种灰阶, 您就有可能看到明显的轮廓——相同灰阶的无规则斑点, 虽然很明显地执行了最接近颜色演算法, 但是一般仍不帶有任何审美判断。通常到 16 种灰阶就可以明显改善图像画质。使用 32 种灰阶差不多就可以消除全部轮廓了。而目前普遍认为 64 种灰阶是现在大多数显示设备的极限。在这点以上, 再提升也没什么边际效益了。在 6 位元颜色解析度的设备上提供超过 64 种灰阶看不出有什么改进之处。

迄今为止, 對於 8 位元显示模式下显示 16 位元、24 位元和 32 位元彩色 DIB, 我们最多就是能够设计通用调色盘 (这对灰阶图像很有效, 但通常不适於彩色图像) 或者使用中间色调色盘, 它用混色显示与通用颜色调色盘合用。

还应注意, 当您在 8 位元颜色模式下为大张 16 位元、24 位元或 32 位元 DIB 选择通用调色盘时, 为了要显示这些图像, 程式将花费一些时间依据 DIB 的内容来建立 GDI 点阵图物件。如果不需要调色盘, 则程式根据 DIB 来建立 DDB 的时间会更少 (用 8 位元彩色模式显示大 24 位元 DIB 时, 比较 SHOWDIB1 和 SHOWDIB4 的性能, 您也能看出这点区别)。这是为什么呢?

它按最接近颜色搜寻。通常, 用 8 位元显示模式显示 24 位元 DIB 时 (或者将 DIB 转换为 DDB), GDI 必须将 DIB 中的每个图素都与静态 20 种颜色中的一种相贴近。完成此操作的唯一方法是决定哪种静态颜色与图素颜色最接近。这包括计算图素与三维 RGB 颜色中每种静态颜色的距离。这将花些时间, 特别是在 DIB 图像中有上百万个图素时。

在建立 232 色调色盘时, 例如 DIBBLE 和 SHOWDIB4 中的通用调色盘, 您会很快将搜索最接近颜色的时间增加到超过 11 倍! GDI 现在必须彻底检查 232 种

颜色，而不是 20 种。那就是显示 DIB 的整个作业放慢的原因。

这里的教训是避免在 8 位元显示模式下显示 24 位元（或 16 位元，或 32 位元）DIB。您应该找出最接近 DIB 图像颜色范围的 256 色调色盘，来将它们转换成 8 位元 DIB。这经常称为「最佳调色盘」。当我研究这个问题的时候，Paul Heckbert 编写的〈Color Image Quantization for Frame Buffer Displays〉（刊登在 1982 年 7 月出版的《Computer Graphics》）对此问题有所帮助。

均匀分布

建立 256 色调色盘最简单的方法是选择范围统一的 RGB 颜色值，它与 DibPalAllPurpose 中的方法相似。此方法的优点是您不必检查 DIB 中的实际图素。这个函式是 DibPalCreateUniformColors，它依据范围统一的 RGB 三原色索引建立调色盘。

一个合理的分布包括 8 阶红色和绿色以及 4 阶蓝色（肉眼对蓝色较不敏感）。调色盘是 RGB 颜色值的集合，它是红色和绿色值 0x00、0x24、0x49、0x6D、0x92、0xB6、0xDB 和 0xFF 以及蓝色值 0x00、0x55、0xAA 和 0xFF 的所有可能的组合，共有 256 种颜色。另一种可能的统一分布调色盘使用 6 阶红色、绿色和蓝色。此调色盘是红色、绿色和蓝色值为 0x00、0x33、0x66、0x99、0xCC 和 0xFF 的所有可能的组合，调色盘中的颜色数是 6 的 3 次方，即 216。

这两个选项和其他几个选项都由 DIBBLE 提供。

「Popularity」演算法

「Popularity」演算法是 256 色调色盘问题相当明显的解决方法。您要做的就是走遍点阵图中的所有图素，并找出 256 种最普通的 RGB 颜色值。这些就是您在调色盘中使用的值。DIBPAL 的 DibPalPopularity 函式中实作了这种演算法。

不过，如果每种颜色都使用整个 24 位元，而且假设需要用整数来计算所有的颜色，那么阵列将占据 64MB 记忆体。另外，您可以发现点阵图中实际上没有（或很少）重复的 24 位元图素值，这样就没有所谓常见的颜色了。

要解决这个问题，您可以只使用每个红色、绿色和蓝色值中最重要的 n 位元——例如，6 位元而不是 8 位元。因为大多数的彩色扫描器和视讯显示卡都只有 6 位元的解析度，所以这样规定更有意义。这将阵列减少到大小更合理的 256KB 或 1MB。只使用 5 位元能将可用的颜色总数减少到 32,768。通常，使用 5 位元要比 6 位元的性能更好。对此，您可以用 DIBBLE 和一些图像颜色来自己检验。

「Median Cut」演算法

DIBPAL.C 中的 DIBPalMedianCut 函式执行 Paul Heckbert 的 Median Cut 演算法。此演算法在概念上相当简单，但在程式码中实作要比 Popularity 演算法更困难，它适合递归函式。

画出 RGB 颜色立方体。图像中的每个图素都是此立方体中的一个点。一些点可能代表图像中的多个图素。找出包括图像中所有图素的立体方块，找出此方块的最大尺寸，并将方块分成两个，每个方块都包括相同数量的图素。对于这两个方块，执行相同的操作。现在您就有 4 个方块，将这 4 个方块分成 8 个，然后再分成 16 个、32 个、64 个、128 个和 256 个。

现在您有 256 个方块，每个方块都包括相同数量的图素。取每个方块中图素 RGB 颜色值的平均值，并将结果用于调色盘。

实际上，这些方块通常包含图素的数量并不相同。例如，通常包括单个点的方块会有更多的图素。这发生在黑色和白色上。有时，一些方块里头根本没有图素。如果这样，您就可以省下更多的方块，但是我决定不这样做。

另一种最佳化调色盘的技术称为「octree quantization」，此技术由 Jeff Prosise 提出，并于 1996 年 8 月发表在《Microsoft Systems Journal》上（包含在 MSDN 的 CD 中）。

转换格式

DIBBLE 还允许将 DIB 从一种格式转换到另一种格式。这用到了 DIBCONV 档案中的 DIBConvert 函式，如程式 16-25 所示。

程式 16-25 DIBCONV 档案

```
DIBCONV.H

/*-----
--
--      DIBCONV.H header file for DIBCONV.C
--
--*/

HDIB DIBConvert (HDIB hdibSrc, int iBitCountDst) ;
DIBCONV.C
/*-----
--
--      DIBCONV.C --      Converts DIBs from one format to another
--
--      (c) Charles Petzold, 1998
--
--*/
```

```

#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"
#include "dibconv.h"

HDIB DibConvert (HDIB hdibSrc, int iBitCountDst)
{
    HDIB                hdibDst ;
    HPALETTE            hPalette ;
    int                 i, x, y, cx, cy, iBitCountSrc, cColors ;
    PALETTEENTRY pe ;
    RGBQUAD             rgb ;
    WORD                wNumEntries ;

    cx = DibWidth (hdibSrc) ;
    cy = DibHeight (hdibSrc) ;
    iBitCountSrc = DibBitCount (hdibSrc) ;

    if (iBitCountSrc == iBitCountDst)
        return NULL ;
    // DIB with color table to DIB with larger color table:
    if ((iBitCountSrc < iBitCountDst) && (iBitCountDst <= 8))
    {
        cColors = DibNumColors (hdibSrc) ;
        hdibDst = DibCreate (cx, cy, iBitCountDst, cColors) ;

        for (i = 0 ; i < cColors ; i++)
        {
            DibGetColor (hdibSrc, i, &rgb) ;
            DibSetColor (hdibDst, i, &rgb) ;
        }

        for (x = 0 ; x < cx ; x++)
        for (y = 0 ; y < cy ; y++)
        {
            DibSetPixel (hdibDst, x, y, DibGetPixel (hdibSrc, x, y)) ;
        }
    }
    // Any DIB to DIB with no color table
    else if (iBitCountDst >= 16)
    {
        hdibDst = DibCreate (cx, cy, iBitCountDst, 0) ;
        for (x = 0 ; x < cx ; x++)
        for (y = 0 ; y < cy ; y++)
        {
            DibGetPixelColor (hdibSrc, x, y, &rgb) ;
            DibSetPixelColor (hdibDst, x, y, &rgb) ;
        }
    }
}

```

```
    }  
}  
    // DIB with no color table to 8-bit DIB  
else if (iBitCountSrc >= 16 && iBitCountDst == 8)  
{  
    hPalette = DibPalMedianCut (hdibSrc, 6) ;  
    GetObject (hPalette, sizeof (WORD), &wNumEntries) ;  
  
    hdibDst = DibCreate (cx, cy, 8, wNumEntries) ;  
    for (i = 0 ; i < (int) wNumEntries ; i++)  
    {  
        GetPaletteEntries (hPalette, i, 1, &pe) ;  
        rgb.rgbRed          = pe.peRed ;  
        rgb.rgbGreen        = pe.peGreen ;  
        rgb.rgbBlue         = pe.peBlue ;  
        rgb.rgbReserved     = 0 ;  
  
        DibSetColor (hdibDst, i, &rgb) ;  
    }  
  
    for (x = 0 ; x < cx ; x++)  
    for (y = 0 ; y < cy ; y++)  
    {  
        DibGetPixelColor (hdibSrc, x, y, &rgb) ;  
  
        DibSetPixel (hdibDst, x, y,  
        GetNearestPaletteIndex (hPalette,  
        RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;  
    }  
    DeleteObject (hPalette) ;  
}  
    // Any DIB to monochrome DIB  
  
else if (iBitCountDst == 1)  
{  
    hdibDst = DibCreate (cx, cy, 1, 0) ;  
    hPalette = DibPalUniformGrays (2) ;  
  
    for (i = 0 ; i < 2 ; i++)  
    {  
        GetPaletteEntries (hPalette, i, 1, &pe) ;  
  
        rgb.rgbRed    = pe.peRed ;  
        rgb.rgbGreen  = pe.peGreen ;  
        rgb.rgbBlue   = pe.peBlue ;  
        rgb.rgbReserved = 0 ;  
  
        DibSetColor (hdibDst, i, &rgb) ;  
    }  
}
```

```

    }

    for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
        DibGetPixelColor (hdibSrc, x, y, &rgb) ;

        DibSetPixel (hdibDst, x, y,
            GetNearestPaletteIndex (hPalette,
                RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;
    }
    DeleteObject (hPalette) ;
}

// All non-monochrome DIBs to 4-bit DIB
else if (iBitCountSrc >= 8 && iBitCountDst == 4)
{
    hdibDst = DibCreate (cx, cy, 4, 0) ;
    hPalette = DibPalVga () ;

    for (i = 0 ; i < 16 ; i++)
    {
        GetPaletteEntries (hPalette, i, 1, &pe) ;
        rgb.rgbRed = pe.peRed ;
        rgb.rgbGreen = pe.peGreen ;
        rgb.rgbBlue = pe.peBlue ;
        rgb.rgbReserved = 0 ;

        DibSetColor (hdibDst, i, &rgb) ;
    }

    for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
        DibGetPixelColor (hdibSrc, x, y, &rgb) ;

        DibSetPixel (hdibDst, x, y,
            GetNearestPaletteIndex (hPalette,
                RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;
    }
    DeleteObject (hPalette) ;
}

// Should not be necessary
else
    hdibDst = NULL ;

return hdibDst ;
}

```

将 DIB 从一种格式转换成另一种格式需要几种不同的方法。

要将带有颜色表的 DIB 转换成另一种也带有颜色表但有较大的图素宽度的 DIB (亦即, 将 1 位元 DIB 转换成 4 位元或 8 位元 DIB, 或将 4 位元 DIB 转换成 8 位元 DIB), 所需要做的就是透过呼叫 `DibCreate` 来建立新的 DIB, 并在呼叫时带有希望的位元数以及与原始 DIB 中的颜色数相等的颜色数。然後函式复制图素位元和颜色表项目。

如果新的 DIB 没有颜色表 (即位元数是 16、24 或 32), 那么 DIB 只需要按新格式建立, 而且通过呼叫 `DibGetPixelColor` 和 `DibSetPixelColor` 从现有的 DIB 中复制图素位元。

下面的情况可能更普遍: 现有的 DIB 没有颜色表 (即位元数是 16、24 或 32), 而新的 DIB 每图素占 8 位元。这种情况下, `DibConvert` 呼叫 `DibPalMedianCut` 来为图像建立最佳化的调色盘。新 DIB 的颜色表设定为调色盘中的 RGB 值。`DibGetPixelColor` 函式从现有的 DIB 中获得图素颜色。透过呼叫 `GetNearestPaletteIndex` 来转换成 8 位元 DIB 中的图素值, 并透过呼叫 `DibSetPixel` 将图素值储存到 DIB。

当 DIB 需要转换成单色 DIB 时, 用包括两个项目——黑色和白色——的颜色表建立新的 DIB。另外, `GetNearestPaletteIndex` 有助於将现有 DIB 中的颜色转换成图素值 0 或 1。类似地, 当 8 个图素位元或更多位元的 DIB 要转换成 4 位元 DIB 时, 可从 `DibPalVga` 函式获得 DIB 颜色表, 同时 `GetNearestPaletteIndex` 也有助于计算图素值。

尽管 DIBBLE 示范了如何开始写一个图像处理程式基础, 但是程式最後还是没有全部完成, 我们总是会想到还有些功能没有加进去里头。但是很可惜的是, 我们现在得停止继续研究这些东西, 而往下讨论别的东西了。

第十七章 文字和字体

显示文字是本书所要解决的首要问题，现在我们来研究 Microsoft Windows 中各种有效字体和字体大小的使用方法以及调整文字的方式。

Windows 3.1 发表的 TrueType 使程式写作者和使用者以灵活的方式处理文字的能力大幅增强。TrueType 是轮廓字体技术，由 Apple Computer 公司和 Microsoft 公司开发，并被许多字体制造商支援。由於 TrueType 字体能够连续缩放，并能应用於视讯显示器和印表机，现在能够在 Windows 下实作真的 WYSIWYG (what you see is what you get: 所见即所得)。TrueType 也便於制作「奇妙」字体，例如旋转的字母、内部填充图案的字母或将它们用於剪裁区域，在本章我将展示它们。

简单的文字输出

让我们先来看看 Windows 为文字输出、影响文字的装置内容属性以及备用字体提供的各种函式。

文字输出函式

我已经在许多范例程式中使用过最常用的文字输出函式：

```
TextOut (hdc, xStart, yStart, pString, iCount) ;
```

参数 xStart 和 yStart 是逻辑座标上字串的起始点。通常，这是 Windows 开始绘制的第一个字母的左上角。TextOut 需要指向字串的指标和字串的长度，这个函式不能识别以 NULL 终止的字串。

TextOut 函式的 xStart 和 yStart 参数的含义可由 SetTextAlign 函式改变。TA_LEFT、TA_RIGHT 和 TA_CENTER 旗标影响使用 xStart 在水平方向上定位字串的方式。预设值是 TA_LEFT。如果在 SetTextAlign 函式中指定了 TA_RIGHT，则後面的 TextOut 呼叫会将字串的最後一个字元定位於 xStart，如果指定了 TA_CENTER，则字串的中心位於 xStart。

类似地，TA_TOP、TA_BOTTOM 和 TA_BASELINE 旗标影响字串的垂直位置。TA_TOP 是预设值，它意味著字串的字母顶端位於 yStart，使用 TA_BOTTOM 意味著字串位於 yStart 之上。可以使用 TA_BASELINE 定位字串，使基准线位於 yStart。基准线是如小写字母 p、q、y 等字母下部的线。

如果您使用 TA_UPDATECP 旗标呼叫 SetTextAlign，Windows 就会忽略 TextOut 的 xStart 和 yStart 参数，而使用由 MoveToEx、LineTo 或更改目前位

置的另一个函式设定的位置。TA_UPDATECP 旗标也使 TextOut 函式将目前位置更新为字串的结尾 (TA_LEFT) 或字串的开头 (TA_RIGHT)。这在使用多个 TextOut 呼叫显示一行文字时非常有用。当水平位置是 TA_CENTER 时, 在 TextOut 呼叫後, 目前位置不变。

您应该还记得, 第四章的一系列 SYSMETS 程式显示几列文字时, 对每一列都需要呼叫一个 TextOut, 其替代函式是 TabbedTextOut 函式:

```
TabbedTextOut ( hdc, xStart, yStart, pString, iCount,
                iNumTabs, piTabStops, xTabOrigin) ;
```

如果文字字串中含有嵌入的跳位字元 (‘\t’ 或 0x09), 则 TabbedTextOut 会根据传递给它的整数阵列将跳位字元扩展为空格。

TabbedTextOut 的前五个参数与 TextOut 相同, 第六个参数是跳位间隔数, 第七个是以图素为单位的跳位间隔阵列。例如, 如果平均字元宽度是 8 个图素, 而您希望每 5 个字元加一个跳位间隔, 则这个阵列将包含 40、80、120, 按递增顺序依此类推。

如果第六个和第七个参数是 0 或 NULL, 则跳位间隔按每八个平均字元宽度设定。如果第六个参数是 1, 则第七个参数指向一个整数, 表示跳位间隔重复增大的倍数 (例如, 如果第六个参数是 1, 并且第七个参数指向值为 30 的变数, 则跳位间隔设定在 30、60、90...图素处)。最後一个参数给出了从跳位间隔开始测量的逻辑 x 座标, 它与字串的起始位置可能相同也可能不同。

另一个进阶的文字输出函式是 ExtTextOut (字首 Ext 表示它是扩展的):

```
ExtTextOut (hdc, xStart, yStart, iOptions, &rect,
            pString, iCount, pxDistance) ;
```

第五个参数是指向矩形结构的指标, 在 iOptions 设定为 ETO_CLIPPED 时, 该结构为剪裁矩形, 在 iOptions 设定为 ETO_OPAQUE 时, 该结构为用目前背景色填充的背景矩形。这两种选择您可以都采用, 也可以都不采用。

最後一个参数是整数阵列, 它指定了字串中连续字元的间隔。程式可以使用它使字元间距变窄或变宽, 因为有时需要在较窄的列中调整单个文字。该参数可以设定为 NULL 来使用内定的字元间距。

用於写文字的高级函式是 DrawText, 我们第一次遇到它是在第三章讨论 HELLOWIN 程式时, 它不指定座标的起始位置, 而是通过 RECT 结构型态定义希望显示文字的区域:

```
DrawText (hdc, pString, iCount, &rect, iFormat) ;
```

和其他文字输出函式一样, DrawText 需要指向字串的指标和字串的长度。然而, 如果在 DrawText 中使用以 NULL 结尾的字串, 就可以将 iCount 设定为-1, Windows 会自动计算字串的长度。

当 iFormat 设定为 0 时, Windows 会将文字解释为一系列由 carriage return

字元 (‘\r’ 或 0x0D) 或 linefeed 字元 (‘\n’ 或 0x0A) 分隔的行。文字从矩形的左上角开始, carriage return 字元或 linefeed 字元被解释为换行字元, 因此 Windows 会结束目前行而开始新的一行。新的一行从矩形的左侧开始, 在上一行的下面空开一个字元的高度 (没有外部间隔)。包含字母的任何文字都应该显示在所剪裁矩形底部的右边或下边。

您可以使用 iFormat 参数更改 DrawText 的内定操作, iFormat 由一个或多个旗标组成。DT_LEFT 旗标 (预设值) 指定了左对齐的行, DT_RIGHT 指定了向右对齐的行, 而 DT_CENTER 指定了位於矩形左边和右边中间的行。因为 DT_LEFT 的值是 0, 所以如果只需要左对齐, 就不需要包含识别字。

如果您不希望将 carriage return 字元或 linefeed 字元解释为换行字元, 则可以包括识别字 DT_SINGLELINE。然後, Windows 会把 carriage return 字元和 linefeed 字元解释为可显示的字元, 而不是控制字元。在使用 DT_SINGLELINE 时, 还可以将行指定为位於矩形的顶端 (DT_TOP)、底端 (DT_BOTTOM) 或者中间 (DT_VCENTER, V 表示垂直)。

在显示多行文字时, Windows 通常只在 carriage return 字元或 linefeed 字元处换行。然而, 如果行的长度超出了矩形的宽度, 则可以使用 DT_WORDBREAK 旗标, 它使 Windows 在行内字的末尾换行。對於单行或多行文字的显示, Windows 会把超出矩形的文字部分截去, 可以使用 DT_NOCLIP 跳过这个操作, 这个旗标还加快了函式的速度。当 Windows 确定多行文字的行距时, 它通常使用不带外部间距的字元高度, 如果您想在行距中加入外部间距, 就可以使用旗标 DT_EXTERNALLEADING。

如果文字中包含跳位字元 (‘\t’ 或 0x09), 则您需要包括旗标 DT_EXPANDTABS。在内定情况下, 跳位间隔设定於每八个字元的位置。通过使用旗标 DT_TABSTOP, 您可以指定不同的跳位间隔, 在这种情况下, iFormat 的高位元组包含了每个新跳位间隔的字元位置数值。不过我建议避免使用 DT_TABSTOP, 因为 iFormat 的高位元组也用於其他旗标。

DT_TABSTOP 旗标存在的问题, 可以由新的函式 DrawTextEx 来解决, 它含有一个额外的参数:

```
DrawTextEx (hdc, pString, iCount, &rect, iFormat, &drawtextparams);
```

最後一个参数是指向 DRAWTEXT_PARAMS 结构的指标, 它的定义如下:

```
typedef struct tagDRAWTEXT_PARAMS
{
    UINT    cbSize ;                // size of structure
    int     iTabLength ;            // size of each tab stop
    int     iLeftMargin ;           // left margin
    int     iRightMargin ;          // right margin
}
```



```
UINT uiLengthDrawn ;           // receives number of characters processed
} DRAWTEXT_PARAMS, * LPDRAWTEXT_PARAMS ;
```

中间三个栏位是以平均字元的增量为单位的。

文字的装置内容属性

除了上面讨论的 `SetTextAlign` 外，其他几个装置内容属性也对文字产生了影响。在内定的装置内容下，文字颜色是黑色，但您可以用下面的叙述进行更改：

```
SetTextColor (hdc, rgbColor) ;
```

使用画笔的颜色和画刷的颜色，Windows 把 `rgbColor` 的值转换为纯色，您可以通过呼叫 `GetTextColor` 取得目前文字的颜色。

Windows 在矩形的背景区域中显示文字，它可能根据背景模式的设定进行着色，也可能不这样做。您可以使用

```
SetBkMode (hdc, iMode) ;
```

更改背景模式，其中 `iMode` 的值为 `OPAQUE` 或 `TRANSPARENT`。内定的背景模式为 `OPAQUE`，它表示 Windows 使用背景颜色来填充矩形的背景。您可以使用

```
SetBkColor (hdc, rgbColor) ;
```

来改变背景颜色。`rgbColor` 的值是转换为纯色的值。内定背景色是白色。

如果两行文字靠得太近，其中一个的背景矩形就会遮盖另一个的文字。由於这种原因，我通常希望内定的背景模式是 `TRANSPARENT`。在背景模式为 `TRANSPARENT` 的情况下，Windows 会忽略背景色，也不对矩形背景区域着色。Windows 也使用背景模式和背景色对点和虚线之间的空隙及阴影刷中阴影间的区域着色，就像第五章所讨论的那样。

许多 Windows 程式将 `WHITE_BRUSH` 指定为 Windows 用於擦出视窗背景的画刷，画刷在视窗类别结构中指定。然而，您可能希望您程式的视窗背景与使用者在「控制台」中设定的系统颜色保持一致，在这种情况下，可以在 `WNDCLASS` 结构中指定背景颜色的这种方式：

```
wndclass.hbrBackground = COLOR_WINDOW + 1 ;
```

当您想要在显示区域书写文字时，可以使用目前系统颜色设定文字色和背景色：

```
SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;
SetBkColor (hdc, GetSysColor (COLOR_WINDOW)) ;
```

完成这些以後，就可以使您的程式随系统颜色的更改而变化：

```
case WM_SYSCOLORCHANGE :
    InvalidateRect (hwnd, NULL, TRUE) ;
    break ;
```

另一个影响文字的装置内容属性是字元间距。它的预设值是 0，表示 Windows

不在字元之间添加任何空间，但您可以使用以下函式插入空间：

```
SetTextCharacterExtra (hdc, iExtra) ;
```

参数 iExtra 是逻辑单位，Windows 将其转换为最接近的图素，它可以是 0。如果您将 iExtra 取为负值（希望将字元紧紧压在一起），Windows 会接受这个数值的绝对值——也就是说，您不能使 iExtra 的值小于 0。您可以通过呼叫 GetTextCharacterExtra 取得目前的字元间距，Windows 在传回该值前会将图素间距转换为逻辑单位。

使用备用字体

当您呼叫 TextOut、TabbedTextOut、ExtTextOut、DrawText 或 DrawTextEx 书写文字时，Windows 使用装置内容中目前选择的字体。字体定义了特定的字样和大小。以不同字体显示文字的最简单方法是使用 Windows 提供的备用字体，然而，它的范围是很有限的。

您可以呼叫下面的函式取得某种备用字体的代号：

```
hFont = GetStockObject (iFont) ;
```

其中，iFont 是几个识别字之一。然後，您就可以将该字体选入装置内容：

```
SelectObject (hdc, hFont) ;
```

这些您也可以只用一步完成：

```
SelectObject (hdc, GetStockObject (iFont)) ;
```

在内定的装置内容中选择的字体称为系统字体，能够由 GetStockObject 的 SYSTEM_FONT 参数识别。这是调和的 ANSI 字元集字体。在 GetStockObject 中指定 SYSTEM_FIXED_FONT（我在本书的前面几个程式中应用过），可以获得等宽字体的代号，这一字体与 Windows 3.0 以前的系统字体相容。在您希望所有的字体都具有相同宽度时，这是很方便的。

备用字体 OEM_FIXED_FONT 也称为终端机字体，是 Windows 在 MS-DOS 命令提示视窗中使用的字体，它包括与原始 IBM-PC 扩展字元集相容的字元集。Windows 在视窗标题列、功能表和对话方块的文字中使用 DEFAULT_GUI_FONT。

当您新字体选入装置内容时，必须使用 GetTextMetrics 计算字元的高度和平均宽度。如果选择了调和字体，那么一定要注意，字元的平均宽度只是个平均值，某些字元会比它宽或比它窄。在本章的後面，您会了解到确定由不同宽度字元所组成的字串总宽度的方法。

尽管 GetStockObject 确实提供了存取不同字体的最简单方式，但是您还不能充分控制项 Windows 所提供的字体。不久，您会看到指定字体字样和大小的方法。

字体的背景

本章剩余的部分致力於处理不同的字体。但是在您接触这些特定程式码前，对 Windows 使用字体的基本知识有一个深入的了解是很有好处的。

字体型態

Windows 支援两大类字体，即所谓的「GDI 字体」和「设备字体」。GDI 字体储存在硬碟的档案中，而设备字体是输出设备本来就有的。例如，通常印表机都具有内建的设备字体集。

GDI 字体有三种样式：点阵字体，笔划字体和 TrueType 字体。

点阵字体的每个字元都以点阵图图素图案的形式储存，每种点阵字体都有特定的纵横比和字元大小。Windows 通过简单地复制图素的行或列就可以由 GDI 点阵字体产生更大的字元。然而，只能以整数倍放大字体，并且不能超过一定的限度。由於这种原因，GDI 点阵字体又称为「不可缩放的」字体。它们不能随意地放大或缩小。点阵字体的主要优点是显示性能（显示速度很快）和可读性（因为是手工设计的，所以尽可能清晰）。

字体是通过字体名称识别的，点阵字体的字体名称为：

- System （用於 SYSTEM_FONT）
- FixedSys （用於 SYSTEM_FIXED_FONT）
- Terminal （用於 OEM_FIXED_FONT）
- Courier
- MS Serif
- MS Sans Serif （用於 DEFAULT_GUI_FONT）
- Small Fonts

每个点阵字体只有几种大小（不超过 6 种）。Courier 字体是定宽字体，外形与用打字机打出的字体相似。「Serif」指字体字母笔划在结束时拐个小弯。「sans serif」字体不是 serif 类的字体。在 Windows 的早期版本中，MS (Microsoft) Serif 和 MS Sans Serif 字体被称为 Tms Rmn（指它与 Times Roman 相似）和 Helv（与 Helvetica 相似）。Small Fonts 是专为显示小字设计的。

在 Windows3.1 以前，除了 GDI 字体外，Windows 所提供的字体只有笔划字体。笔划字体是以「连结点」的方式定义的一系列线段，笔划字体可以连续地缩放，这意味著同样的字体可以用於具有任何解析度的图形输出设备，并且字体可以放大或缩小到任意尺寸。不过，它的性能不好，小字体的可读性也很糟，而大字体由於笔划是单根直线而显得很单薄。笔划字体有时也称为绘图机字体，

因为它们特别适合於绘图机，但是不适合於别的场合。笔划字体的字样有：Modern、Roman 和 Script。

对於 GDI 点阵字体和 GDI 笔划字体，Windows 都可以「合成」粗体、斜体、加底线和加删除线，而不需要为每种属性另外储存字体。例如，对於斜体，Windows 只需要将字元的上部向右移动就可以了。

接下来是 TrueType，我将在本章的剩部分主要讨论它。

TrueType 字体

TrueType 字体的单个字元是通过填充的直线和曲线的轮廓来定义的。Windows 可以通过改变定义轮廓的座标对 TrueType 字体进行缩放。

当程式开始使用特定大小的 TrueType 字体时，Windows「点阵化」字体。这就是说 Windows 使用 TrueType 字体档案中包括的「提示」对每个字元的连结直线和曲线的座标进行缩放。这些提示可以补偿误差，避免合成的字元变得很难看（例如，在某些字体中，大写 H 的两竖应该一样宽，但盲目地缩放字体可能会导致其中一竖的图素比另一竖宽。有了提示就可以避免这些现象发生）。然後，每个字元的合成轮廓用於建立字元的点阵图，这些点阵图储存在记忆体以备将来使用。

最初，Windows 使用了 13 种 TrueType 字体，它们的字体名称如下：

- Courier New
- Courier New Bold
- Courier New Italic
- Courier New Bold Italic
- Times New Roman
- Times New Roman Bold
- Times New Roman Italic
- Times New Roman Bold Italic
- Arial
- Arial Bold
- Arial Italic
- Arial Bold Italic
- Symbol

在新的 Windows 版本中，这个列表更长了。在此特别指出，我将使用 Lucida Sans Unicode 字体，它包括了一些在世界其他地方使用的字母表。

三个主要字体系列与点阵字体相似，Courier New 是定宽字体。它看起来就

像是打字机输出的字体。Times New Roman 是 Times 字体的复制品，该字体最初为《Times of London》设计，并用在许多印刷材料上，它具有很好的可读性。Arial 是 Helvetica 字体的复制品，是一种 sans serif 字体。Symbol 字体包含了手写符号集。

属性或样式

在上面的 TrueType 字体列表中，您会注意到，Courier、Times New Roman 和 Arial 的粗体和斜体是带有自己字体名称的单独字体，这一命名与传统的板式一致。然而，电脑使用者认为粗体和斜体只是已有字体的特殊「属性」。Windows 在定义点阵字体命名、列举和选择的方式时，采用了属性的方法。但对于 TrueType 字体，更倾向于使用传统的命名方式。

这种冲突在 Windows 中还没有完全解决，简而言之，您可以完全通过命名或特定属性来选择字体。然而在处理字体列举时，应用程式需要系统中的字体列表，正如您所预料，这种双重处理使问题复杂化了。

点值

在传统的版式中，您可以用字体名称和大小来指定字体，字体的大小以点的单位来表示。一点与 1/72 英寸很接近——它们非常接近，因此在电脑中它通常定义为 1/72 英寸。点值通常描述为字母顶端（不包括发音符号）到字母底端的高度，例如，字母「bq」的总高度。这是一个考虑字体大小的简单方式，但它通常不是很精确。

字体的点值实际上是排版设计的概念而不是度量概念。特定字体中字元的大小可能会大于或小于其点值所表示的大小。在传统的排版中，您使用点值来指定字体的大小，在电脑排版中，还有其他方法来确定字元的实际大小。

间隔和间距

在第四章我们曾提到，可以通过呼叫 GetTextMetrics 取得装置内容中目前选择的字体资讯，我们也多次使用过这个函式。图 4-3 显示了 FONTMETRIC 结构中字体的垂直大小。

TEXTMETRIC 结构的另一个栏位是 tmExternalLeading，词「间隔(leading)」来自排字工人在金属字块间插入的铅，它用于在两行文字之间产生空白。tmInternalLeading 值与为发音符号保留的空间有关，tmExternalLeading 表示字元的连续行之间所留的附加空间。程式写作者可以使用或忽略外部的间隔值。

当我们说一个字体是 8 点或 12 点时，指的是不带内部间隔的高度。某种大

写字母上的发音符号占据了分隔行的间距。这样，TEXTMETRIC 结构的 tmHeight 值实际指行间距而不是字体的点值。字体的点值可由 tmHeight 减 tmInternalLeading 得到。

逻辑英寸问题

正如我们在第五章〈设备的大小〉一节中所讨论的，Windows 98 将系统字体定义为带有 12 点行距的 10 点字体。根据在「显示属性」对话方块中选择的是「小字体」还是「大字体」，该字体的 tmHeight 值为 16 或 20 图素，tmHeight 减去 tmInternalLeading 的值为 13 或 16 图素。这样，字体的选择就暗指以每英寸的点数为单位的设备解析度，选择「小字体」即为 96dpi，选择「大字体」即为 120dpi。

您可以用 LOGPIXELSX 或 LOGPIXELSY 参数呼叫 GetDeviceCaps 来取得该设备解析度。因此，96 或 120 图素在萤幕上占有的度量距离可以称为「逻辑英寸」。如果您用尺测量萤幕并计算图素，就可能发现逻辑英寸要比实际的英寸大一些，为什么会这样呢？

在纸张上，每英寸放设 14 个 8 点的字元很方便阅读。如果您在作文书处理或写作应用程式时，可能希望在显示器上显示清晰的 8 点字型，但如果使用视讯显示器的实际尺寸，就没有足够的图素清晰地显示字元。即使显示器具有足够的解析度，在萤幕上阅读 8 点字体仍然会有问题。当人们阅读纸上的印刷物时，眼睛与文字的距离通常为一英尺，而使用视讯显示器时，这个距离通常为两英尺。

逻辑英寸有效地对萤幕进行了放大，能够显示小至 8 点的清晰字体。而且，每英寸 96 点使 640 图素的最小显示大小等於大约 6.5 英寸。这恰恰是在页边距为 1 英寸的 8.5 英寸宽的纸上列印的文字的宽度。因而，逻辑英寸也利用了萤幕宽度，尽可能大地显示文字。

您可能还记得在第五章，Windows NT 的做法有些不同。在 Windows NT 中，从 GetDeviceCaps 中得到的 LOGPIXELSX（每英寸的图素数）值不等於 HORZRES 值(图素数)除以 HORZSIZE 值(毫米数)再乘以 25.4 的值。以此类似，LOGPIXELSY、VERTRES 和 VERTSIZE 也不一致。Windows 在为不同映射方式计算视窗和偏移范围时，使用 HORZRES、HORZSIZE、VERTRES 和 VERTSIZE 值。然而，显示文字的程式最好不要使用根据 LOGPIXELSX 和 LOGPIXELSY 使用假定的显示解析度，这一点与 Windows 98 更为一致。

所以，在 Windows NT 下，当程式以特定的点值显示文字时，它可能不使用 Windows 提供的映射方式，程式根据与 Windows 98 一样的每英寸的逻辑图素数

来定义自己的映射方式。我将这种用于文字的映射方式称为「Logical Twips」映射方式。您可以设定如下：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;  
SetWindowExtEx (hdc, 1440, 1440, NULL) ;  
SetViewportExt (hdc, GetDeviceCaps (hdc, LOGPIXELSX),  
                GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
```

使用这种映射方式设定，您能够以点值的 20 倍来指定字体大小，例如，为 12 点字取 240。注意，与 MM_TWIPS 映射方式不同，y 值在萤幕中向下增长，这在显示文字的连续行时很方便。

请记住，逻辑英寸与实际英寸间的差异仅对显示器存在。在列印设备上，GDI 和尺是完全一致的。

逻辑字体

既然我们已经明确了逻辑英寸和逻辑单位的概念，那么现在我们就来讨论逻辑字体。

逻辑字体是一个 GDI 物件，它的代号储存在 HFONT 型态的变数中，逻辑字体是字体的描述。和逻辑画笔及逻辑画刷一样，它是抽象的物件，只有当应用程序呼叫 SelectObject 将它选入装置内容时，它才成为真实的物件。例如，对于逻辑画笔，您可以为画笔指定任意的颜色，但是在您将画笔选入装置内容时，Windows 才将其转换为设备中有效的颜色。只有此时，Windows 才知道设备的色彩能力。

逻辑字体的建立和选择

您可以透过呼叫 CreateFont 或 CreateFontIndirect 来建立逻辑字体。CreateFontIndirect 函式接受一个指向 LOGFONT 结构的指标，该结构有 14 个栏位。CreateFont 函式接受 14 个参数，它们与 LOGFONT 结构的 14 个栏位形式相同。它们是仅有的两个建立逻辑字体的函式（我提到这一点，是因为 Windows 中有许多用于其他字体操作的函式）。因为很难记住 14 个栏位，所以很少使用 CreateFont。因此，我主要讨论 CreateFontIndirect。

有三种基本的方式用于定义 LOGFONT 结构中的栏位，以便呼叫 CreateFontIndirect：

- 您可以简单地将 LOGFONT 结构的栏位设定为所需的字体特征。在这种情况下，在呼叫 SelectObject 时，Windows 使用「字体映射」演算法从设备上有效的字体中选择与这些特征最匹配的字体。由于这依赖于视讯显示器和印表机上的有效字体，所以其结果可能与您的要求有相当大的差

别。

- 您可以列举设备上的所有字体并从中选择，甚至用对话方块把它们显示给使用者。我将在本章後面讨论字体列举函式。不过，它们现在已经不常用了，因为第三种方法也可以进行列举。
- 您可以采用简单的方法并呼叫 ChooseFont 函式，我在第十一章曾讨论过这个函式，能够使用 LOGFONT 结构直接建立字体。

在本章，我使用第一种和第三种方法。

下面是建立、选择和删除逻辑字体的程序：

1. 通过呼叫 CreateFont 或 CreateFontIndirect 建立逻辑字体，这些函式传回 HFONT 型态的逻辑字体代号。
2. 使用 SelectObject 将逻辑字体选入装置内容，Windows 会选择与逻辑字体最匹配的真实字体。
3. 使用 GetTextMetrics (及可能用到的其他函式) 确定真实字体的大小和特徵。在该字体选入装置内容後，可以使用这些资讯来适当地设定文字的间距。
4. 在使用完逻辑字体後，呼叫 DeleteObject 删除逻辑字体，当字体选入有效的装置内容时，不要删除字体，也不要删除备用字体。

GetTextFace 函式使程式能够确定目前选入装置内容的字体名称：

```
GetTextFace (hdc, sizeof (szFaceName) / sizeof (TCHAR), szFaceName) ;
```

详细的字体资讯可以从 GetTextMetrics 中得到：

```
GetTextMetrics (hdc, &textmetric) ;
```

其中，textmetric 是 TEXTMETRIC 型态的变数，它具有 20 个栏位。

稍後我将详细讨论 LOGFONT 和 TEXTMETRIC 结构的栏位，这两个结构有一些相似的栏位，所以它们容易混淆。现在您只需记住，LOGFONT 用於定义逻辑字体，而 TEXTMETRIC 用於取得目前选入装置内容中的字体资讯。

PICKFONT 程式

使用程式 17-1 所示的 PICKFONT，可以定义 LOGFONT 结构的许多栏位。这个程式建立逻辑字体，并在逻辑字体选入装置内容後显示真实字体的特徵。这是个方便的程式，通过它我们可以了解逻辑字体映射为真实字体的方式。

程式 17-1 PICKFONT

```
PICKFONT.C
/*-----
--
--          PICKFONT.C --          Create Logical Font
--
--                                     (c) Charles Petzold, 1998
-------*/
/
```



```

#include <windows.h>
#include "resource.h"

// Structure shared between main window and dialog box
typedef struct
{
    int                iDevice, iMapMode ;
    BOOL               fMatchAspect ;
    BOOL               fAdvGraphics ;
    LOGFONT            lf ;
    TEXTMETRIC         tm ;
    TCHAR              szFaceName [LF_FULLFACESIZE] ;
}
DLGPARAMS ;

// Formatting for BCHAR fields of TEXTMETRIC structure
#ifdef UNICODE
#define BCHARFORM TEXT ("0x%04X")
#else
#define BCHARFORM TEXT ("0x%02X")
#endif

// Global variables
HWND hdlg ;
TCHAR szAppName[] = TEXT ("PickFont") ;

// Forward declarations of functions
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
void SetLogFontFromFields (HWND hdlg, DLGPARAMS * pdp) ;
void SetFieldsFromTextMetric (HWND hdlg, DLGPARAMS * pdp) ;
void MySetMapMode(HDC hdc, int iMapMode) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;

```

```

    wndclass.lpszMenuName      = szAppName ;
    wndclass.lpszClassName     = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("PickFont: Create Logical Font"),
                          WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        if (hdlg == 0 || !IsDialogMessage (hdlg, &msg))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static DLGPARAMS dp ;
    static TCHAR      szText[] = TEXT ("\x41\x42\x43\x44\x45 ")
        TEXT ("\x61\x62\x63\x64\x65 ")

        TEXT ("\xC0\xC1\xC2\xC3\xC4\xC5 ")
        TEXT ("\xE0\xE1\xE2\xE3\xE4\xE5 ")

#ifdef UNICODE
        TEXT ("\x0390\x0391\x0392\x0393\x0394\x0395 ")
        TEXT ("\x03B0\x03B1\x03B2\x03B3\x03B4\x03B5 ")
        TEXT ("\x0410\x0411\x0412\x0413\x0414\x0415 ")
        TEXT ("\x0430\x0431\x0432\x0433\x0434\x0435 ")
        TEXT ("\x5000\x5001\x5002\x5003\x5004")
#endif

    ;

    HDC
                                hdc ;

```

```

PAINTSTRUCT                                ps ;
RECT                                        rect ;

switch (message)
{
case WM_CREATE:
    dp.iDevice = IDM_DEVICE_SCREEN ;
    hdlg = CreateDialogParam ((LPCTSTR) lParam)->hInstance,
        szAppName, hwnd, DlgProc, (LPARAM) &dp) ;
        return 0 ;
case WM_SETFOCUS:
    SetFocus (hdlg) ;
    return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
case IDM_DEVICE_SCREEN:
case IDM_DEVICE_PRINTER:
        CheckMenuItem (GetMenu (hwnd), dp.iDevice, MF_UNCHECKED) ;
        dp.iDevice = LOWORD (wParam) ;
        CheckMenuItem (GetMenu (hwnd), dp.iDevice, MF_CHECKED) ;
        SendMessage (hwnd, WM_COMMAND, IDOK, 0) ;
        return 0 ;
    }
    break ;

case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        // Set graphics mode so escapement works in Windows NT

        SetGraphicsMode (hdc, dp.fAdvGraphics ? GM_ADVANCED : GM_COMPATIBLE) ;

        // Set the mapping mode and the mapper flag

        MySetMapMode (hdc, dp.iMapMode) ;
        SetMapperFlags (hdc, dp.fMatchAspect) ;

        // Find the point to begin drawing text

        GetClientRect (hdlg, &rect) ;
        rect.bottom += 1 ;
        DPtoLP (hdc, (PPOINT) &rect, 2) ;

        // Create and select the font; display the text

        SelectObject (hdc, CreateFontIndirect (&dp.lf)) ;

```

```

        TextOut (hdc, rect.left, rect.bottom, szText, lstrlen
(szText)) ;

        DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK DlgProc (        HWND hdlg, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static DLGPARAMS          *    pdp ;
    static PRINTDLG            pd = { sizeof (PRINTDLG) } ;
    HDC                        hdcDevice ;
    HFONT                      hFont ;

    switch (message)
    {
    case WM_INITDIALOG:
        // Save pointer to dialog-parameters structure in WndProc

        pdp = (DLGPARAMS *) lParam ;

        SendDlgItemMessage (hdlg, IDC_LF_FACENAME,
            EM_LIMITTEXT, LF_FACESIZE - 1, 0) ;
        CheckRadioButton (hdlg, IDC_OUT_DEFAULT, IDC_OUT_OUTLINE,
            IDC_OUT_DEFAULT) ;
        CheckRadioButton (hdlg, IDC_DEFAULT_QUALITY, IDC_PROOF_QUALITY,
            IDC_DEFAULT_QUALITY) ;
        CheckRadioButton (hdlg, IDC_DEFAULT_PITCH, IDC_VARIABLE_PITCH,
            IDC_DEFAULT_PITCH) ;
        CheckRadioButton (hdlg, IDC_FF_DONTCARE, IDC_FF_DECORATIVE,
            IDC_FF_DONTCARE) ;
        CheckRadioButton (hdlg, IDC_MM_TEXT, IDC_MM_LOGTWIPS,
            IDC_MM_TEXT) ;
        SendMessage (hdlg, WM_COMMAND, IDOK, 0) ;

                                                // fall through

    case WM_SETFOCUS:
        SetFocus (GetDlgItem (hdlg, IDC_LF_HEIGHT)) ;
        return FALSE ;
    }
}

```

```

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_CHARSET_HELP:
        MessageBox (    hdlg,
TEXT    ("0          =    Ansi\n")
TEXT    ("1          =    Default\n")
TEXT    ("2          =    Symbol\n")
TEXT    ("128        =    Shift JIS (Japanese)\n")
TEXT    ("129        =    Hangul (Korean)\n")
TEXT    ("130        =    Johab (Korean)\n")
TEXT    ("134        =    GB 2312 (Simplified Chinese)\n")
TEXT    ("136        =    Chinese Big 5 (Traditional Chinese)\n")
TEXT    ("177        =    Hebrew\n")
TEXT    ("178        =    Arabic\n")
TEXT    ("161        =    Greek\n")
TEXT    ("162        =    Turkish\n")
TEXT    ("163        =    Vietnamese\n")
TEXT    ("204        =    Russian\n")
TEXT    ("222        =    Thai\n")
TEXT    ("238        =    East European\n")
TEXT    ("255        =    OEM"),
        szAppName, MB_OK | MB_ICONINFORMATION) ;
        return TRUE ;

        // These radio buttons set the lfOutPrecision field

    case IDC_OUT_DEFAULT:
        pdp->lf.lfOutPrecision = OUT_DEFAULT_PRECIS ;
        return TRUE ;

    case IDC_OUT_STRING:
        pdp->lf.lfOutPrecision = OUT_STRING_PRECIS ;
        return TRUE ;

    case IDC_OUT_CHARACTER:
        pdp->lf.lfOutPrecision = OUT_CHARACTER_PRECIS ;
        return TRUE ;

    case IDC_OUT_STROKE:
        pdp->lf.lfOutPrecision = OUT_STROKE_PRECIS ;
        return TRUE ;

    case IDC_OUT_TT:
        pdp->lf.lfOutPrecision = OUT_TT_PRECIS ;
        return TRUE ;

    case IDC_OUT_DEVICE:

```

```

        pdp->lf.lfOutPrecision = OUT_DEVICE_PRECIS ;
        return TRUE ;

    case IDC_OUT_RASTER:
        pdp->lf.lfOutPrecision = OUT_RASTER_PRECIS ;
        return TRUE ;

    case IDC_OUT_TT_ONLY:
        pdp->lf.lfOutPrecision = OUT_TT_ONLY_PRECIS ;
        return TRUE ;

    case IDC_OUT_OUTLINE:
        pdp->lf.lfOutPrecision = OUT_OUTLINE_PRECIS ;
        return TRUE ;

    // These three radio buttons set the lfQuality field

    case IDC_DEFAULT_QUALITY:
        pdp->lf.lfQuality = DEFAULT_QUALITY ;
        return TRUE ;

    case IDC_DRAFT_QUALITY:
        pdp->lf.lfQuality = DRAFT_QUALITY ;
        return TRUE ;

    case IDC_PROOF_QUALITY:
        pdp->lf.lfQuality = PROOF_QUALITY ;
        return TRUE ;

    // These three radio buttons set the lower nibble
    //   of the lfPitchAndFamily field

    case IDC_DEFAULT_PITCH:
        pdp->lf.lfPitchAndFamily = (0xF0 &
pdp->lf.lfPitchAndFamily) | DEFAULT_PITCH ;
        return TRUE ;

    case IDC_FIXED_PITCH:
        pdp->lf.lfPitchAndFamily = (0xF0 &
pdp->lf.lfPitchAndFamily) | FIXED_PITCH ;
        return TRUE ;

    case IDC_VARIABLE_PITCH:
        pdp->lf.lfPitchAndFamily = (0xF0 &
pdp->lf.lfPitchAndFamily) | VARIABLE_PITCH ;
        return TRUE ;

    // These six radio buttons set the upper nibble

```

```

// of the lfPitchAndFamily field

        case IDC_FF_DONTCARE:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_DONTCARE ;
            return TRUE ;

        case IDC_FF_ROMAN:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_ROMAN ;
            return TRUE ;

        case IDC_FF_SWISS:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_SWISS ;
            return TRUE ;

        case IDC_FF_MODERN:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_MODERN ;
            return TRUE ;

        case IDC_FF_SCRIPT:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_SCRIPT ;
            return TRUE ;

        case IDC_FF_DECORATIVE:
            pdp->lf.lfPitchAndFamily = (0x0F &
pdp->lf.lfPitchAndFamily) | FF_DECORATIVE ;
            return TRUE ;

// Mapping mode:

        case IDC_MM_TEXT:
        case IDC_MM_LOMETRIC:
        case IDC_MM_HIMETRIC:
        case IDC_MM_LOENGLISH:
        case IDC_MM_HIENGLISH:
        case IDC_MM_TWIPS:
        case IDC_MM_LOGTWIPS:
            pdp->iMapMode = LOWORD (wParam) ;
            return TRUE ;

// OK button pressed
// -----

        case IDOK:

```

```
// Get LOGFONT structure

SetLogFontFromFields (hdlg, pdp) ;

// Set Match-Aspect and Advanced Graphics flags

pdp->fMatchAspect = IsDlgButtonChecked (hdlg,
IDC_MATCH_ASPECT) ;
pdp->fAdvGraphics = IsDlgButtonChecked (hdlg,
IDC_ADV_GRAPHICS) ;

// Get Information Context

if (pdp->iDevice == IDM_DEVICE_SCREEN)
{
    hdcDevice = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
}
else
{
    pd.hwndOwner = hdlg ;
    pd.Flags = PD_RETURNDEFAULT | PD_RETURNIC ;
    pd.hDevNames = NULL ;
    pd.hDevMode = NULL ;

    PrintDlg (&pd) ;

    hdcDevice = pd.hDC ;
}

// Set the mapping mode and the mapper flag

MySetMapMode (hdcDevice, pdp->iMapMode) ;
SetMapperFlags (hdcDevice, pdp->fMatchAspect) ;

// Create font and select it into IC

hFont = CreateFontIndirect (&pdp->lf) ;
SelectObject (hdcDevice, hFont) ;

// Get the text metrics and face name

GetTextMetrics (hdcDevice, &pdp->tm) ;
GetTextFace (hdcDevice, LF_FULLFACESIZE, pdp->szFaceName) ;
DeleteDC (hdcDevice) ;
DeleteObject (hFont) ;

// Update dialog fields and invalidate main window

SetFieldsFromTextMetric (hdlg, pdp) ;
```



```

        InvalidateRect (GetParent (hdlg), NULL, TRUE) ;
        return TRUE ;
    }
    break ;
}
return FALSE ;
}

void SetLogFontFromFields (HWND hdlg, DLGPARAMS * pdp)
{
    pdp->lf.lfHeight = GetDlgItemInt (hdlg, IDC_LF_HEIGHT, NULL, TRUE) ;
    pdp->lf.lfWidth = GetDlgItemInt (hdlg, IDC_LF_WIDTH, NULL, TRUE) ;
    pdp->lf.lfEscapement=GetDlgItemInt (hdlg, IDC_LF_ESCAPE, NULL, TRUE) ;
    pdp->lf.lfOrientation=GetDlgItemInt (hdlg, IDC_LF_ORIENT, NULL, TRUE) ;
    pdp->lf.lfWeight =GetDlgItemInt (hdlg, IDC_LF_WEIGHT, NULL, TRUE) ;
    pdp->lf.lfCharSet =GetDlgItemInt (hdlg, IDC_LF_CHARSET, NULL, FALSE) ;
    pdp->lf.lfItalic =IsDlgButtonChecked(hdlg, IDC_LF_ITALIC) ==
BST_CHECKED ;
    pdp->lf.lfUnderline =IsDlgButtonChecked (hdlg, IDC_LF_UNDER) ==
BST_CHECKED ;
    pdp->lf.lfStrikeOut =IsDlgButtonChecked (hdlg, IDC_LF_STRIKE) ==
BST_CHECKED ;
    GetDlgItemText (hdlg, IDC_LF_FACENAME, pdp->lf.lfFaceName, LF_FACESIZE) ;
}

void SetFieldsFromTextMetric (HWND hdlg, DLGPARAMS * pdp)
{
    TCHAR          szBuffer [10] ;
    TCHAR *        szYes      = TEXT ("Yes") ;
    TCHAR *        szNo   = TEXT ("No") ;
    TCHAR *        szFamily [] = { TEXT ("Don't Know"),
TEXT ("Roman"),
        TEXT ("Swiss"), TEXT ("Modern"),
        TEXT ("Script"), TEXT ("Decorative"),
        TEXT ("Undefined") } ;

    SetDlgItemInt (hdlg, IDC_TM_HEIGHT, pdp->tm.tmHeight, TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_ASCENT, pdp->tm.tmAscent, TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_DESCENT, pdp->tm.tmDescent, TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_INTLEAD, pdp->tm.tmInternalLeading,
        TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_EXTLEAD, pdp->tm.tmExternalLeading,
TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_AVECHAR, pdp->tm.tmAveCharWidth,
        TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_MAXCHAR, pdp->tm.tmMaxCharWidth,
        TRUE) ;
    SetDlgItemInt (hdlg, IDC_TM_WEIGHT, pdp->tm.tmWeight,
        TRUE) ;

```

```

SetDlgItemInt (hdlg, IDC_TM_OVERHANG,    pdp->tm.tmOverhang,
              TRUE) ;
SetDlgItemInt (hdlg, IDC_TM_DIGASPX,    pdp->tm.tmDigitizedAspectX,
              TRUE) ;
SetDlgItemInt (hdlg, IDC_TM_DIGASPY,    pdp->tm.tmDigitizedAspectY,
              TRUE) ;

wsprintf (szBuffer, BCHARFORM, pdp->tm.tmFirstChar) ;
SetDlgItemText (hdlg, IDC_TM_FIRSTCHAR, szBuffer) ;

wsprintf (szBuffer, BCHARFORM, pdp->tm.tmLastChar) ;
SetDlgItemText (hdlg, IDC_TM_LASTCHAR, szBuffer) ;

wsprintf (szBuffer, BCHARFORM, pdp->tm.tmDefaultChar) ;
SetDlgItemText (hdlg, IDC_TM_DEFCHAR, szBuffer) ;

wsprintf (szBuffer, BCHARFORM, pdp->tm.tmBreakChar) ;
SetDlgItemText (hdlg, IDC_TM_BREAKCHAR, szBuffer) ;

SetDlgItemText (hdlg, IDC_TM_ITALIC,    pdp->tm.tmItalic           ?
szYes : szNo) ;
SetDlgItemText (hdlg, IDC_TM_UNDER,    pdp->tm.tmUnderlined       ?
szYes : szNo) ;
SetDlgItemText (hdlg, IDC_TM_STRUCK,    pdp->tm.tmStruckOut        ?
szYes : szNo) ;

SetDlgItemText (hdlg, IDC_TM_VARIABLE,
                TMPF_FIXED_PITCH & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

SetDlgItemText (hdlg, IDC_TM_VECTOR,
                TMPF_VECTOR & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

SetDlgItemText (hdlg, IDC_TM_TRUETYPE,
                TMPF_TRUETYPE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

SetDlgItemText (hdlg, IDC_TM_DEVICE,
                TMPF_DEVICE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

SetDlgItemText (hdlg, IDC_TM_FAMILY,
                szFamily [min (6, pdp->tm.tmPitchAndFamily >> 4)]) ;

SetDlgItemInt (hdlg, IDC_TM_CHARSET, pdp->tm.tmCharSet, FALSE) ;
SetDlgItemText (hdlg, IDC_TM_FACENAME, pdp->szFaceName) ;
}

void MySetMapMode (HDC hdc, int iMapMode)
{
    switch (iMapMode)

```

```

{
case IDC_MM_TEXT:          SetMapMode (hdc, MM_TEXT) ;          break ;
case IDC_MM_LOMETRIC: SetMapMode (hdc, MM_LOMETRIC) ;    break ;
case IDC_MM_HIMETRIC: SetMapMode (hdc, MM_HIMETRIC) ;    break ;
case IDC_MM_LOENGLISH:    SetMapMode (hdc, MM_LOENGLISH) ;    break ;
case IDC_MM_HIENGLISH:    SetMapMode (hdc, MM_HIENGLISH) ;    break ;
case IDC_MM_TWIPS:        SetMapMode (hdc, MM_TWIPS) ;
break ;
case IDC_MM_LOGTWIPS:
        SetMapMode (hdc, MM_ANISOTROPIC) ;
        SetWindowExtEx (hdc, 1440, 1440, NULL) ;
        SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
        GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
        break ;
}
}
PICKFONT.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
PICKFONT DIALOG DISCARDABLE 0, 0, 348, 308
STYLE WS_CHILD | WS_VISIBLE | WS_BORDER
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT                                "&Height:", IDC_STATIC, 8, 10, 44, 8
    EDITTEXT                            IDC_LF_HEIGHT, 64, 8, 24, 12, ES_AUTOHSCROLL
    LTEXT                                "&Width", IDC_STATIC, 8, 26, 44, 8
    EDITTEXT                            IDC_LF_WIDTH, 64, 24, 24, 12, ES_AUTOHSCROLL
    LTEXT                                "Escapement:", IDC_STATIC, 8, 42, 44, 8
    EDITTEXT                            IDC_LF_ESCAPE, 64, 40, 24, 12, ES_AUTOHSCROLL
    LTEXT                                "Orientation:", IDC_STATIC, 8, 58, 44, 8
    EDITTEXT                            IDC_LF_ORIENT, 64, 56, 24, 12, ES_AUTOHSCROLL
    LTEXT                                "Weight:", IDC_STATIC, 8, 74, 44, 8
    EDITTEXT                            IDC_LF_WEIGHT, 64, 74, 24, 12, ES_AUTOHSCROLL
    GROUPBOX                            "Mapping
Mode", IDC_STATIC, 97, 3, 96, 90, WS_GROUP
    CONTROL
        "Text", IDC_MM_TEXT, "Button", BS_AUTORADIOBUTTON, 104, 13, 56,
            8
    CONTROL                            "Low
Metric", IDC_MM_LOMETRIC, "Button", BS_AUTORADIOBUTTON,
            104, 24, 56, 8
    CONTROL                            High Metric", IDC_MM_HIMETRIC, "Button",
            BS_AUTORADIOBUTTON, 104, 35, 56, 8

```

```

CONTROL                                "Low English", IDC_MM_LOENGLISH, "Button",
                                         BS_AUTORADIOBUTTON, 104, 46, 56, 8
CONTROL                                "High English", IDC_MM_HIENGLISH, "Button",
                                         BS_AUTORADIOBUTTON, 104, 57, 56, 8
CONTROL
"Twips", IDC_MM_TWIPS, "Button", BS_AUTORADIOBUTTON, 104, 68,
                                         56, 8
CONTROL                                "Logical Twips", IDC_MM_LOGTWIPS, "Button",
                                         BS_AUTORADIOBUTTON, 104, 79, 64, 8
CONTROL                                "Italic", IDC_LF_ITALIC, "Button", BS_AUTOCHECKBOX
|
                                         WS_TABSTOP, 8, 90, 48, 12
CONTROL
"Underline", IDC_LF_UNDER, "Button", BS_AUTOCHECKBOX |
                                         WS_TABSTOP, 8, 104, 48, 12
CONTROL                                "Strike
Out", IDC_LF_STRIKE, "Button", BS_AUTOCHECKBOX |
                                         WS_TABSTOP, 8, 118, 48, 12
CONTROL                                "Match
Aspect", IDC_MATCH_ASPECT, "Button", BS_AUTOCHECKBOX |
                                         WS_TABSTOP, 60, 104, 62, 8
CONTROL                                "Adv Grfx Mode", IDC_ADV_GRAPHICS, "Button",
                                         BS_AUTOCHECKBOX
WS_TABSTOP, 60, 118, 62, 8
LTEXT                                "Character
Set:", IDC_STATIC, 8, 137, 46, 8
EDITTEXT                                IDC_LF_CHARSET, 58, 135, 24, 12, ES_AUTOHSCROLL
PUSHBUTTON                            "?", IDC_CHARSET_HELP, 90, 135, 14, 14
GROUPBOX                                "Quality", IDC_STATIC, 132, 98, 62, 48, WS_GROUP
CONTROL                                "Default", IDC_DEFAULT_QUALITY, "Button",
BS_AUTORADIOBUTTON, 136, 110, 40, 8
CONTROL
"Draft", IDC_DRAFT_QUALITY, "Button", BS_AUTORADIOBUTTON,
                                         136, 122, 40, 8
CONTROL
"Proof", IDC_PROOF_QUALITY, "Button", BS_AUTORADIOBUTTON,
                                         136, 134, 40, 8
LTEXT                                "Face Name:", IDC_STATIC, 8, 154, 44, 8
EDITTEXT
IDC_LF_FACENAME, 58, 152, 136, 12, ES_AUTOHSCROLL
GROUPBOX                                "Output
Precision", IDC_STATIC, 8, 166, 118, 133, WS_GROUP
CONTROL
"OUT_DEFAULT_PRECIS", IDC_OUT_DEFAULT, "Button",
BS_AUTORADIOBUTTON, 12, 178, 112, 8
CONTROL

```

```

"OUT_STRING_PRECIS", IDC_OUT_STRING, "Button",

BS_AUTORADIOBUTTON, 12, 191, 112, 8
CONTROL
"OUT_CHARACTER_PRECIS", IDC_OUT_CHARACTER, "Button",

BS_AUTORADIOBUTTON, 12, 204, 112, 8
CONTROL
"OUT_STROKE_PRECIS", IDC_OUT_STROKE, "Button",

BS_AUTORADIOBUTTON, 12, 217, 112, 8
CONTROL
"OUT_TT_PRECIS", IDC_OUT_TT, "Button", BS_AUTORADIOBUTTON,
                                12, 230, 112, 8
CONTROL
"OUT_DEVICE_PRECIS", IDC_OUT_DEVICE, "Button",

BS_AUTORADIOBUTTON, 12, 243, 112, 8
CONTROL
"OUT_RASTER_PRECIS", IDC_OUT_RASTER, "Button",

BS_AUTORADIOBUTTON, 12, 256, 112, 8
CONTROL
"OUT_TT_ONLY_PRECIS", IDC_OUT_TT_ONLY, "Button",

BS_AUTORADIOBUTTON, 12, 269, 112, 8
CONTROL
"OUT_OUTLINE_PRECIS", IDC_OUT_OUTLINE, "Button",

BS_AUTORADIOBUTTON, 12, 282, 112, 8
GROUPBOX                                "Pitch", IDC_STATIC, 132, 166, 62, 50, WS_GROUP
CONTROL
"Default", IDC_DEFAULT_PITCH, "Button", BS_AUTORADIOBUTTON,
                                137, 176, 52, 8
CONTROL
"Fixed", IDC_FIXED_PITCH, "Button", BS_AUTORADIOBUTTON, 137,
                                189, 52, 8
CONTROL                                "Variable", IDC_VARIABLE_PITCH, "Button",

BS_AUTORADIOBUTTON, 137, 203, 52, 8
GROUPBOX                                "Family", IDC_STATIC, 132, 218, 62, 82, WS_GROUP
CONTROL                                "Don't
Care", IDC_FF_DONTCARE, "Button", BS_AUTORADIOBUTTON,
                                137, 229, 52, 8
CONTROL
"Roman", IDC_FF_ROMAN, "Button", BS_AUTORADIOBUTTON, 137, 241,
                                52, 8
CONTROL

```

```

"Swiss", IDC_FF_SWISS, "Button", BS_AUTORADIOBUTTON, 137, 253,
                                52, 8
CONTROL
"Modern", IDC_FF_MODERN, "Button", BS_AUTORADIOBUTTON, 137,
                                265, 52, 8
CONTROL
"Script", IDC_FF_SCRIPT, "Button", BS_AUTORADIOBUTTON, 137,
                                277, 52, 8
CONTROL
"Decorative", IDC_FF_DECORATIVE, "Button",
BS_AUTORADIOBUTTON, 137, 289, 52, 8
DEFPUSHBUTTON          "OK", IDOK, 247, 286, 50, 14
GROUPBOX                "Text
Metrics", IDC_STATIC, 201, 2, 140, 272, WS_GROUP
LTEXT
"Height:", IDC_STATIC, 207, 12, 64, 8
LTEXT                                "0", IDC_TM_HEIGHT, 281, 12, 44, 8
LTEXT
"Ascent:", IDC_STATIC, 207, 22, 64, 8
LTEXT                                "0", IDC_TM_ASCENT, 281, 22, 44, 8
LTEXT
"Descent:", IDC_STATIC, 207, 32, 64, 8
LTEXT                                "0", IDC_TM_DESCENT, 281, 32, 44, 8
LTEXT                                "Internal
Leading:", IDC_STATIC, 207, 42, 64, 8
LTEXT                                "0", IDC_TM_INTLEAD, 281, 42, 44, 8
LTEXT                                "External
Leading:", IDC_STATIC, 207, 52, 64, 8
LTEXT                                "0", IDC_TM_EXTLEAD, 281, 52, 44, 8
LTEXT                                "Ave                      Char
Width:", IDC_STATIC, 207, 62, 64, 8
LTEXT                                "0", IDC_TM_AVECHAR, 281, 62, 44, 8
LTEXT                                "Max                      Char
Width:", IDC_STATIC, 207, 72, 64, 8
LTEXT                                "0", IDC_TM_MAXCHAR, 281, 72, 44, 8
LTEXT
"Weight:", IDC_STATIC, 207, 82, 64, 8
LTEXT                                "0", IDC_TM_WEIGHT, 281, 82, 44, 8
LTEXT
"Overhang:", IDC_STATIC, 207, 92, 64, 8
LTEXT                                "0", IDC_TM_OVERHANG, 281, 92, 44, 8
LTEXT                                "Digitized                      Aspect
X:", IDC_STATIC, 207, 102, 64, 8
LTEXT                                "0", IDC_TM_DIGASPX, 281, 102, 44, 8
LTEXT                                "Digitized                      Aspect

```

```

Y:", IDC_STATIC, 207, 112, 64, 8
    LTEXT
    "0", IDC_TM_DIGASPY, 281, 112, 44, 8
    LTEXT                                "First
Char:", IDC_STATIC, 207, 122, 64, 8
    LTEXT
    "0", IDC_TM_FIRSTCHAR, 281, 122, 44, 8
    LTEXT                                "Last
Char:", IDC_STATIC, 207, 132, 64, 8
    LTEXT
    "0", IDC_TM_LASTCHAR, 281, 132, 44, 8
    LTEXT                                "Default
Char:", IDC_STATIC, 207, 142, 64, 8
    LTEXT
    "0", IDC_TM_DEFCHAR, 281, 142, 44, 8
    LTEXT                                "Break
Char:", IDC_STATIC, 207, 152, 64, 8
    LTEXT
    "0", IDC_TM_BREAKCHAR, 281, 152, 44, 8
    LTEXT
    "Italic?", IDC_STATIC, 207, 162, 64, 8
    LTEXT                                "0", IDC_TM_ITALIC, 281, 162, 44, 8
    LTEXT
    "Underlined?", IDC_STATIC, 207, 172, 64, 8
    LTEXT                                "0", IDC_TM_UNDER, 281, 172, 44, 8
    LTEXT                                "Struck Out?", IDC_STATIC, 207, 182, 64, 8
    LTEXT                                "0", IDC_TM_STRUCK, 281, 182, 44, 8
    LTEXT                                "Variable
Pitch?", IDC_STATIC, 207, 192, 64, 8
    LTEXT                                "0", IDC_TM_VARIABLE, 281, 192, 44, 8
    LTEXT                                "Vector
Font?", IDC_STATIC, 207, 202, 64, 8
    LTEXT                                "0", IDC_TM_VECTOR, 281, 202, 44, 8
    LTEXT                                "TrueType
Font?", IDC_STATIC, 207, 212, 64, 8
    LTEXT                                "0", IDC_TM_TRUETYPE, 281, 212, 44, 8
    LTEXT                                "Device
Font?", IDC_STATIC, 207, 222, 64, 8
    LTEXT                                "0", IDC_TM_DEVICE, 281, 222, 44, 8
    LTEXT                                "Family:", IDC_STATIC, 207, 232, 64, 8
    LTEXT                                "0", IDC_TM_FAMILY, 281, 232, 44, 8
    LTEXT                                "Character
Set:", IDC_STATIC, 207, 242, 64, 8
    LTEXT                                "0", IDC_TM_CHARSET, 281, 242, 44, 8
    LTEXT                                "0", IDC_TM_FACENAME, 207, 262, 128, 8
END

```

```

////////////////////////////////////

```

```
/
// Menu
PICKFONT MENU DISCARDABLE
BEGIN
    POPUP "&Device"
    BEGIN
        MENUITEM "&Screen",          IDM_DEVICE_SCREEN, CHECKED
        MENUITEM "&Printer",        IDM_DEVICE_PRINTER
    END
END

RESOURCE.H
// Microsoft Developer Studio generated include file.
// Used by PickFont.rc

#define IDC_LF_HEIGHT      1000
#define IDC_LF_WIDTH      1001
#define IDC_LF_ESCAPE     1002
#define IDC_LF_ORIENT     1003
#define IDC_LF_WEIGHT     1004
#define IDC_MM_TEXT       1005
#define IDC_MM_LOMETRIC   1006
#define IDC_MM_HIMETRIC   1007
#define IDC_MM_LOENGLISH  1008
#define IDC_MM_HIENGLISH  1009
#define IDC_MM_TWIPS      1010
#define IDC_MM_LOGTWIPS   1011
#define IDC_LF_ITALIC     1012
#define IDC_LF_UNDER      1013
#define IDC_LF_STRIKE     1014
#define IDC_MATCH_ASPECT  1015
#define IDC_ADV_GRAPHICS  1016
#define IDC_LF_CHARSET    1017
#define IDC_CHARSET_HELP  1018
#define IDC_DEFAULT_QUALITY 1019
#define IDC_DRAFT_QUALITY  1020
#define IDC_PROOF_QUALITY  1021
#define IDC_LF_FACENAME    1022
#define IDC_OUT_DEFAULT    1023
#define IDC_OUT_STRING     1024
#define IDC_OUT_CHARACTER  1025
#define IDC_OUT_STROKE     1026
#define IDC_OUT_TT         1027
#define IDC_OUT_DEVICE     1028
#define IDC_OUT_RASTER     1029
#define IDC_OUT_TT_ONLY    1030
#define IDC_OUT_OUTLINE    1031
#define IDC_DEFAULT_PITCH  1032
#define IDC_FIXED_PITCH    1033
```



```

#define IDC_VARIABLE_PITCH 1034
#define IDC_FF_DONTCARE 1035
#define IDC_FF_ROMAN 1036
#define IDC_FF_SWISS 1037
#define IDC_FF_MODERN 1038
#define IDC_FF_SCRIPT 1039
#define IDC_FF_DECORATIVE 1040
#define IDC_TM_HEIGHT 1041
#define IDC_TM_ASCENT 1042
#define IDC_TM_DESCENT 1043
#define IDC_TM_INTLEAD 1044
#define IDC_TM_EXTLEAD 1045
#define IDC_TM_AVECHAR 1046
#define IDC_TM_MAXCHAR 1047
#define IDC_TM_WEIGHT 1048
#define IDC_TM_OVERHANG 1049
#define IDC_TM_DIGASPX 1050
#define IDC_TM_DIGASPY 1051
#define IDC_TM_FIRSTCHAR 1052
#define IDC_TM_LASTCHAR 1053
#define IDC_TM_DEFCHAR 1054
#define IDC_TM_BREAKCHAR 1055
#define IDC_TM_ITALIC 1056
#define IDC_TM_UNDER 1057
#define IDC_TM_STRUCK 1058
#define IDC_TM_VARIABLE 1059
#define IDC_TM_VECTOR 1060
#define IDC_TM_TRUETYPE 1061
#define IDC_TM_DEVICE 1062
#define IDC_TM_FAMILY 1063
#define IDC_TM_CHARSET 1064
#define IDC_TM_FACENAME 1065
#define IDM_DEVICE_SCREEN 40001
#define IDM_DEVICE_PRINTER 40002

```

图 17-1 显示了典型的 PICKFONT 萤幕显示。PICKFONT 左半部分显示了一个非模态对话方块，透过它，您可以选择逻辑字体结构的大部分栏位。对话方块的右半部分显示了字体选入装置内容後 GetTextMetrics 的结果。对话方块的下部，程式使用这种字体显示一个字串。因为非模态对话方块非常大，所以最好在 1024 768 或更大的显示大小下执行这个程式。

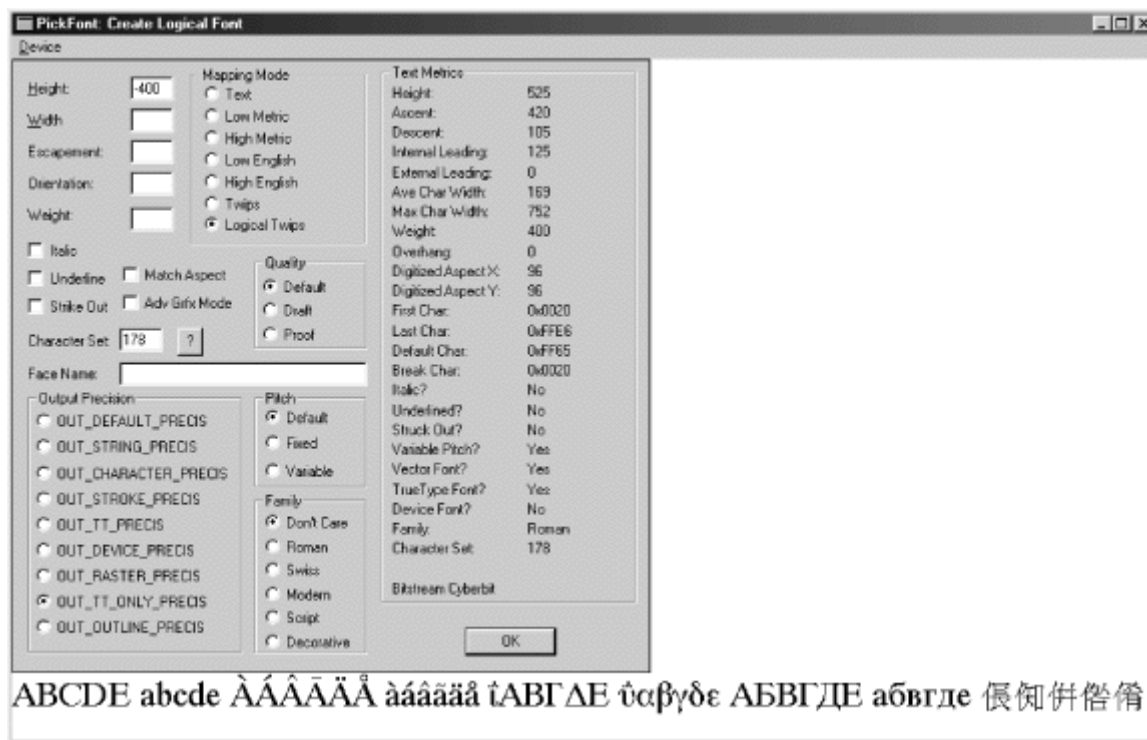


图 17-1 典型的 PICKFONT 萤幕显示 (Windows NT 下的 Unicode 版本)

非模态对话方块还包含一些非逻辑字体结构的选项，它们是包括「Logical Twips」方式的映射方式、「Match Aspect」选项（更改 Windows 将逻辑字体与真实字体匹配的方式）和「Adv Grtx Mode」（设定 Windows NT 中的高级图形模式）。稍後我将对这些作详细讨论。

从「Device」功能表中，可以选择内定印表机而不是视讯显示器。在这种情况下，PICKFONT 将逻辑字体选入印表机装置内容中，并从印表机显示 TEXTMETRIC 结构。然後，程式将逻辑字体选入视窗装置内容中，以显示样本字符串。因此，程式显示的文字可能会使用与 TEXTMETRIC 栏位所描述的字体（印表机字体）不同的字体（萤幕字体）。

PICKFONT 程式的大部分逻辑都在处理对话方块的必要动作，因此我不会详细讨论该程式的工作方式，只解释建立和选择逻辑字体的原理。

逻辑字体结构

您可以呼叫 CreateFont 来建立逻辑字体，它是具有 14 个参数的函式。一般，定义一个 LOGFONT 型态的结构

```
LOGFONT lf ;
```

然後再定义该结构的栏位会更容易一些。完成後，可以使用指向该结构的指标呼叫 CreateFontIndirect：

```
hFont = CreateFontIndirect (&lf) ;
```

您不必设定 LOGFONT 结构的每个栏位。如果逻辑字体结构定义为静态变数，

那么所有的栏位都会初始化为 0，0 一般是预设值。然後，可以不用更改而直接使用这个结构，CreateFontIndirect 会传回字体的代号。当您将该字体选入装置内容时，会得到一个合理的内定字体。您可以根据自己的需要，明确或模糊地填充 LOGFONT 结构，Windows 会用一种真实字体与您的要求相匹配。

在我讨论 LOGFONT 结构中每个栏位时，您可能想用 PICKFONT 程式来测试它们。当您希望程式使用您输入的任何栏位时，别忘了按下 Enter 或「OK」按钮。

LOGFONT 结构的前两个栏位是逻辑单位，因此它们依赖於映射方式的目前设定：

lfHeight 这是以逻辑单位表示的希望的字元高度。您可以将 lfHeight 设定 0，以使用内定大小，或者根据栏位代表的含义将其设定为正数或负数。如果将 lfHeight 设定为正数，就表示您希望该值表示含有内部间隔（不是外部间隔）的高度。实际上，所要求的字体行距为 lfHeight。如果将 lfHeight 设定为负值，则 Windows 会将其绝对值作为与点值一致的字体高度。这是一个很重要的区别：如果想要特定点值的字体，可将点值转换为逻辑单位，并将 lfHeight 栏位设定为该值的负数。如果 lfHeight 是正值，则 TEXTMETRIC 结构的 tmHeight 栏位近似为该值（有时有微小的偏差，可能由於舍入误差所引起）。如果 lfHeight 是负值，则它粗略地与不包括 tmInternalLeading 栏位的 TEXTMETRIC 结构的 tmHeight 栏位相匹配。

lfWidth 是逻辑单位的字元期望宽度。在多数情况下，可以将此值设定为 0，让 Windows 仅根据高度选择字体。使用非零值对点阵字体并不会起太大作用，但對於 TrueType 字体，您能轻松地用它来获得比正常字元更宽或更窄的字体。这个栏位对应於 TEXTMETRIC 结构的 tmAveCharWidth 栏位。要正确使用 lfWidth 栏位，首先把带有 lfWidth 栏位的 LOGFONT 结构设定为 0，建立逻辑字体，将它选入装置内容，然後呼叫 GetTextMetrics。得到 tmAveCharWidth 栏位，可按比例调节其值的大小，然後使用所调节的 lfWidth 的 tmAveCharWidth 值建立第二种字体。

下两个栏位指定文字的「移位角度」和「方向」。理论上，lfEscapement 使字符串能够以一定的角度书写（但每个字元的基准线仍与水平轴平行），而 lfOrientation 使单个字元倾斜。但是这两个栏位并不是那么有效，即使现在它们只有在下面的情况下才能很好地起作用：使用 TrueType 字体、执行 Windows NT 以及首先用 CM_ADVANCED 旗标设定呼叫 SetGraphicsMode。通过选中「Adv Grfx Mode」核取方块，您能够完成 PICKFONT 中的最终需要。

在验证 PICKFONT 中的这些栏位时，要注意单位是十分之一度，逆时针方向旋转。它很容易输入一个值使范例字符串消失！因此，请使用 0 到 -600 或 3000 到

3600 之间的值。

lfEscapement 这是从水平方向上逆时针测量的十分之几的角度。它指定在书写文字时字串的连续字元放置的方式。表 17-1 提供了几个例子：

表 17-1

值	字元的放置
0	从左向右（内定）
900	向上
1800	从右向左
2700	向下

在 Windows 98 中，这个值设定了 TrueType 文字的移位角度和方向。在 Windows NT 中，这个值通常也是这样设定，除了用 GM_ADVANCED 参数呼叫 SetGraphicsMode 时，它按文件中说明的那样工作。

lfOrientation 这是从水平方向逆时针测量的十分之几的角度，它影响单个字元的外观。表 17-2 提供了几个例子：

表 17-2

值	字元外观
0	正常（内定）
900	向右倾斜 90 度
1800	颠倒
2700	向左倾斜 90 度

这个栏位一般不起作用，除非在 Windows NT 下使用 TrueType 字体，并把图像模式设定为 GM_ADVANCED，在这种情况下它按文件中说明的那样工作。

其余 10 个栏位如下：

lfWeight 这个栏位使您能够指定粗体。WINGDI.H 表头档案定义了可用於这个栏位的一组值（参见表 17-3）。

表 17-3

值	识别字
0	FW_DONTCARE
100	FW_THIN
200	FW_EXTRALIGHT 或 FW_ULTRALIGHT
300	FW_LIGHT
400	FW_NORMAL 或 FW_REGULAR
500	FW_MEDIUM

600	FW_SEMIBOLD 或 FW_DEMIBOLD
700	FW_BOLD
800	FW_EXTRABOLD 或 FW_ULTRABOLD
900	FW_HEAVY 或 FW_BLACK

事实上,它比以前用过的任何一组值都完善。您可以对标准字使用 0 或 400,对粗体使用 700。

lfItalic 在非零值时,它指定斜体。Windows 能在 GDI 点阵字体上合成斜体。亦即,Windows 仅仅移动若干行字元点阵图来模仿斜体。对于 TrueType 字体,Windows 使用真正的斜体或字体的倾斜版本。

lfUnderline 在非零值时,它指定底线,这项属性在 GDI 字体上都是用合成的。也就是说,Windows GDI 只是在包括空格的每个字元底线。

lfStrikeOut 在非零值时,它指定字体上应该有一条线穿过。这也是由 GDI 字体合成的。

lfCharSet 这是指定字体字元集的一个位元组的值。我会在下一节「字元集和 Unicode」中更详细地讨论这个栏位。在 PICKFONT 中,您可以按下带有问号的按钮来取得能够使用的字元集列表。

注意 lfCharSet 栏位是唯一不用零表示预设值的栏位。零值相当于 ANSI_CHARSET,ANSI 字元在美国和西欧使用。DEFAULT_CHARSET 代码等于 1,表示程式执行的机器上内定的字元集。

lfOutPrecision 它指定了 Windows 用实际的字体匹配期望的字体大小和特徵的方式。这是一个复杂的栏位,一般很少使用。请查看关于 LOGFONT 结构的文件以得到更详细的资讯。注意,可以使用 OUT_TT_ONLY_PRECIS 旗标来确保得到的是 TrueType 字体。

lfClipPrecision 这个栏位指定了当字元的一部分位于剪裁区以外时,剪裁字元的方式。这个栏位不经常使用,PICKFONT 程式也没有使用它。

lfQuality 这是一个给 Windows 的指令,有关于期望字体与实际字体相匹配的指令。它实际只对点阵字体有意义,并不影响 TrueType 字体。DRAFT_QUALITY 旗标指出需要 GDI 缩放点阵字体以得到想要的大小;PROOF_QUALITY 旗标指出不需缩放。PROOF_QUALITY 字体最漂亮,但它们可能比所希望的要小一些。这个栏位中也可以使用 DEFAULT_QUALITY (或 0)。

lfPitchAndFamily 这个位元组由两部分组成。您可以使用位元或运算符结合用于此栏位的两个识别字。最低的两位元指定字体是定宽(即所有字元的宽度相等)还是变宽(参见表 17-4)。

表 17-4

值	识别字
0	DEFAULT_PITCH
1	FIXED_PITCH
2	VARIABLE_PITCH

位元组的上半部分指定字体系列（参见表 17-5）。

表 17-5

值	识别字
0x00	FW_DONTCARE
0x10	FF_ROMAN（变宽，serifs）
0x20	FF_SWISS（变宽，非 serifs）
0x30	FF_MODERN（定宽）
0x40	FF_SCRIPT（模仿手写）
0x50	FF_DECORATIVE

lfFaceName 这是关于字样（如 Courier、Arial 或 Times New Roman）的实际文字名称。这个栏位是宽度为 LF_FACESIZE（或 32 个字元）的位元组阵列。如果要得到 TrueType 的斜体或粗体字体，有两种方法。在 lfFaceName 栏位中使用完整的字体名称（如 Times New Roman Italic），或者可以使用基本名称（即 Times New Roman），并设定 lfItalic 栏位。

字体映射演算法

在设定了逻辑字体结构后，呼叫 CreateFontIndirect 来得到逻辑字体代号。当呼叫 SelectObject 把逻辑字体选入装置内容时，Windows 寻找与所需字体最接近匹配的实际字体。它使用「字体映射演算法」。结构的某些栏位要比其他栏位更重要一些。

了解字体映射的最好方式是花一些时间试验 PICKFONT。以下是几条指南：

- lfCharSet（字元集）栏位是非常重要的。如果您指定了 OEM_CHARSET(255)，会得到某种笔划字体或终端机字体，因为它们是唯一使用 OEM 字元集的字体。然而，随著 TrueType「Big Fonts」的出现（在第六章〈TrueType 和大字体〉一节讨论过），单一的 TrueType 字体能映射到包括 OEM 字元集等不同的字元集。您需要使用 SYMBOL_CHARSET(2) 来得到 Symbol 字体或 Wingdings 字体。
- lfPitchAndFamily 栏位的 FIXED_PITCH 间距值很重要，因为您实际上告诉 Windows 不想处理变宽字体。

- `lfFaceName` 栏位很重要，因为您指定了所需字体的字样。如果让 `lfFaceName` 设定为 `NULL`，并在 `lfPitchAndFamily` 栏位中将组值设定为 `FF_DONTCARE` 以外的值，因为指定了字体系列，所以该栏位也很重要。
- 对于点阵字体，Windows 会试图配合 `lfHeight` 值，即使需要增加较小字体的大小。实际字体的高度总是小于或等于所需的字体，除非没有更小的字体满足您的要求。对于笔划或 TrueType 字体，Windows 仅简单地将字体缩放到需要的高度。
- 可以通过将 `lfQuality` 设定为 `PROOF_QUALITY` 来防止 Windows 缩放点阵字体。这么做可以告诉 Windows 所需的字体高度没有字体外观重要。
- 如果指明了对于显示器的特定纵横比不协调的 `lfHeight` 和 `lfWeight` 值，Windows 能映射到为显示器或其他不同纵横比的设备设计的点阵字体。这是得到细或粗字体的技巧（当然，对于 TrueType 字体是不必要的）。一般而言，您可能想避免为另一种设备挑配字体。您可以通过单击标有「Match Aspect」的核取方块，在 `PICKFONT` 中完成。如果选中了核取方块，`PICKFONT` 会使用 `TRUE` 参数呼叫 `SetMapperFlags`。

取得字体资讯

在 `PICKFONT` 中非模态对话方块的右侧是字体选入装置内容后从 `GetTextMetrics` 函式中获得的资讯（注意，可以使用 `PICKFONT` 的「Device」功能表指出装置内容是萤幕还是内定印表机。因为在印表机上有效的字体可能不同，所以结果也可能不同）。在 `PICKFONT` 中列表的底部是从 `GetTextFace` 得到的有效字体名称。

除了数值化的纵横比以外，Windows 复制到 `TEXTMETRIC` 结构的所有大小值都以逻辑单位表示。`TEXTMETRIC` 结构的栏位如下：

tmHeight 逻辑单位的字元高度。它近似等于 `LOGFONT` 结构中指定的 `lfHeight` 栏位的值，如果该值为正，它就代表行距，而非点值。如果 `LOGFONT` 结构的 `lfHeight` 栏位为负，则 `tmHeight` 栏位减 `tmInternalLeading` 栏位应近似等于 `lfHeight` 栏位的绝对值。

tmAscent 逻辑单位的基准线以上的字元垂直大小。

tmDescent 逻辑单位的基准线以下的字元垂直大小。

tmInternalLeading 包含在 `tmHeight` 值内的垂直大小，通常被一些大写字母上注音符占据。同样，可以用 `tmHeight` 值减 `tmInternalLeading` 值来计算字体的点值。

tmExternalLeading **tmHeight** 以外的行距附加量，字体的设计者推荐用

於隔开文字的连续行。

tmAveCharWidth 字体中小写字母的平均宽度。

tmMaxCharWidth 逻辑单位的字元最大宽度。对于定宽字体，这个值与 tmAveCharWidth 相同。

tmWeight 字体重量，范围从 0 到 999。实际上，这个栏位为 400 时是标准字体，700 时是粗体。

tmOverhang Windows 在合成斜体或粗体时添加到点阵字体字元的额外宽度量（逻辑单位）。当点阵字体斜体化时，tmAveCharWidth 值保持不变，因为斜体化的字串与相同的正常字串的总宽度相等。要为字体加粗，Windows 必须稍微增加每个字元的宽度。对于粗体，tmAveCharWidth 值小于 tmOverhang 值，等於没有加粗的相同字体的 tmAveCharWidth 值。

tmDigitizedAspectX 和 **tmDigitizedAspectY** 字体合适的纵横比。它们与使用 LOGPIXELSX 和 LOGPIXELSY 识别字从 GetDeviceCaps 得到的值相同。

tmFirstChar 字体中第一个字元的字元代码。

tmLastChar 字体中最后一个字元的字元代码。如果 TEXTMETRIC 结构通过呼叫 GetTextMetricsW（函式的宽字元版本）获得，那么这个值可能大于 255。

tmDefaultChar Windows 用于显示不在字体中的字元的字元代码，通常是矩形。

tmBreakChar 在调整文字时，Windows 和您的程式用于确定单字断开的字元。如果您不用一些奇怪的东西（例如 EBCDIC 字体），它就是 32——空白字元。

tmItalic 对于斜体字为非零值。

tmUnderlined 对于底线字体为非零值。

tmStruckOut 对于删除线字体为非零值。

tmPitchAndFamily 低四位元是表示字体某些特徵的旗标，由在 WINGDI.H 中定义的识别字指出（参见表 17-6）。

表 17-6

值	识别字
0x01	TMPF_FIXED_PITCH
0x02	TMPF_VECTOR
0x04	TMPF_TRUETYPE
0x08	TMPF_DEVICE

不管 TMPF_FIXED_PITCH 旗标的名称是什么，如果字体字元是变宽的，则最低位元为 1。第二最低位元（TMPF_VECTOR）对于 TrueType 字体和使用其他可缩放的轮廓技术的字体（如 PostScript 的字体）为 1。TMPF_DEVICE 旗标表示设

备字体（即印表机内置的字体），而不是依据 GDI 的字体。

这个栏位的第四高的位元表示字体系列，并且与 LOGFONT 的 lfPitchAndFamily 栏位中所用的值相同。

`tmCharSet` 字元集识别字。

字元集和 Unicode

我在第六章讨论了 Windows 字元集的概念，在那里我们必须处理涉及键盘的国际化问题。在 LOGFONT 和 TEXTMETRIC 结构中，所需字体（或实际字体）的字元集由 0 至 255 之间的单个位元组的数值表示。定义在 WINGDI.H 中的字元集识别字如下所示：

#define	ANSI_CHARSET	0
#define	DEFAULT_CHARSET	1
#define	SYMBOL_CHARSET	2
#define	MAC_CHARSET	77
#define	SHIFTJIS_CHARSET	128
#define	HANGEUL_CHARSET	129
#define	HANGUL_CHARSET	129
#define	JOHAB_CHARSET	130
#define	GB2312_CHARSET	134
#define	CHINESEBIG5_CHARSET	136
#define	GREEK_CHARSET	161
#define	TURKISH_CHARSET	162
#define	VIETNAMESE_CHARSET	163
#define	HEBREW_CHARSET	177
#define	ARABIC_CHARSET	178
#define	BALTIC_CHARSET	186
#define	RUSSIAN_CHARSET	204
#define	THAI_CHARSET	222
#define	EASTEUROPE_CHARSET	238
#define	OEM_CHARSET	255

字元集与页码表的概念类似，但是字元集特定於 Windows，且通常小於或等於 255。

与本书的所有程式一样，您可以带有定义的 UNICODE 识别字编译 PICKFONT，也可以不帶 UNICODE 识别字编译它。和往常一样，本书内附光碟上的程式的两个版本分别位於 DEBUG 和 RELEASE 目录中。

注意，在程式的 Unicode 版本中 PICKFONT 在其视窗底部显示的字串要更长一些。在两个版本中，字串的字元代码由 0x40 到 0x45、0x60 到 0x65。不管您选择了哪种字元集（除了 SYMBOL_CHARSET），这些字元代码都显示拉丁字母表的前五个大写和小写字母（即 A 到 E 和 a 到 e）。

当执行 PICKFONT 程式的非 Unicode 版本时，接下来的 12 个字元——字元

代码 0xC0 到 0xC5 以及 0xE0 到 0xE5——将依赖于所选择的字元集。对于 ANSI_CHARSET, 这个字元代码对应于大写和小写字母 A 的加重音版本。对于 GREEK_CHARSET, 这些代码对应于希腊字母表的字母。对于 RUSSIAN_CHARSET, 对应于斯拉夫字母表的字母。注意, 当您选择一种字元集时, 字体可能会改变, 这是因为点阵字体可能没有这些字元, 但 TrueType 字体可能有。您可能回忆起大多数 TrueType 字体是「Big fonts」并且包含几种不同字元集的字母。如果您执行 Windows 的远东版本, 这些字元会被解释为双位元组字元, 并且会按方块字显示, 而不是按字母显示。

在 Windows NT 下执行 PICKFONT 的 Unicode 版本时, 代码 0xC0 到 0xC5 以及 0xE0 到 0xE5 通常是大写和小写字母 A 的加重音版本(除了 SYMBOL_CHARSET), 因为 Unicode 中定义了这些代码。程式也显示 0x0390 到 0x0395 以及 0x03B0 到 0x03B5 的字元代码。由于它们在 Unicode 中有定义, 这些代码总是对应于希腊字母表的字母。同样地, 程式显示 0x0410 到 0x0415 以及 0x0430 到 0x0435 的字元代码, 它们对应于斯拉夫字母表的字母。然而, 这些字元不可能存在于内定字体中, 您必须选择 GREEK_CHARSET 或 RUSSIAN_CHARSET 来得到它们。在这种情况下, LOGFONT 结构中的字元集 ID 不更改实际的字元集; 字元集总是 Unicode。而字元集 ID 指出来自所需字元集的字元。

现在选择 HEBREW_CHARSET (代码 177)。希伯来字母表不包括在 Windows 通常的 Big Fonts 中, 因此作业系统选择 Lucida Sans Unicode, 这一点您可以在非模态对话方块的右下角中验证。

PICKFONT 也显示 0x5000 到 0x5004 的字元代码, 它们对应于汉语、日语和朝鲜语象形文字的一部分。如果您执行 Windows 的远东版本, 或者下载了比 Lucida Sans Unicode 范围更广的免费 Unicode 字体, 就可以看到这些。Bitstream CyberBit 字体就是这样的一种字体, 您可以从 <http://www.bitstream.com/products/world/cyberbits> 中找到。(Lucida Sans Unicode 大约有 300K, 而 Bitstream CyberBit 大约有 13M)。如果您安装了这种字体, 当需要一种 Lucida Sans Unicode 不支援的字体时, Windows 会选择它, 这些字体如: SHIFTJIS_CHARSET (日语)、HANGUL_CHARSET (朝鲜语)、JOHAB_CHARSET (朝鲜语)、GB2312_CHARSET (简体中文) 或 CHINESEBIG5_CHARSET (繁体中文)。

本章的后面有一个程式可让您查看 Unicode 字体的所有字母。

EZFONT 系统

TrueType 字体系统 (以传统的排版为基础) 为 Windows 以不同的方式显示

文字提供了牢固的基础。但是一些 Windows 的字体选择函式依据较旧技术，使得画面上的点阵字体必须趋近印表机设备字体的样子。下一节将讲到列举字体的做法，它能够使程式获得显示器或印表机上全部有效字体的列表。不过，「ChooseFont」对话方块（稍後讨论）确实大幅度消除了程式列举字体的必要性。

因为标准 TrueType 字体可以在任何系统上使用，且这些字体可以用于显示器以及印表机，如此一来，程式在选择 TrueType 字体或在缺乏资讯的情况下取得某种相似的字体时，就没有必要列举字体了。程式只需简单并明确地选择系统中存在的 TrueType 字体（当然，除非使用者故意删除它们）。这种方法与指定字体名称（可能是第十七章中〈TrueType 字体〉一节中列出的 13 种字体中的一种）和字体大小一样简单。我把这种方法称做 EZFONT（「简便字体」），程式 17-2 列出了它的两个档案。

程式 17-2 EZFONT

```
EZFONT.H
/*-----
-
EZFONT.H header file
-----
*/

HFONT EzCreateFont (   HDC hdc, TCHAR * szFaceName, int iDeciPtHeight,
                      int iDeciPtWidth, int iAttributes, BOOL fLogRes) ;

#define      EZ_ATTR_BOLD                1
#define      EZ_ATTR_ITALIC              2
#define      EZ_ATTR_UNDERLINE           4
#define      EZ_ATTR_STRIKEOUT           8
EZFONT.C
/*-----
--
EZFONT.C --      Easy Font Creation
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include <math.h>
#include "ezfont.h"

HFONT EzCreateFont (   HDC hdc, TCHAR * szFaceName, int iDeciPtHeight,
                      int iDeciPtWidth, int iAttributes, BOOL fLogRes)
{
    FLOAT
        cxDpi, cyDpi ;
```

```

HFONT                hFont ;
LOGFONT              lf ;
POINT                pt ;
TEXTMETRIC           tm ;

SaveDC (hdc) ;
SetGraphicsMode (hdc, GM_ADVANCED) ;
ModifyWorldTransform (hdc, NULL, MWT_IDENTITY) ;
SetViewportOrgEx (hdc, 0, 0, NULL) ;
SetWindowOrgEx (hdc, 0, 0, NULL) ;

if (fLogRes)
{
    cxDpi = (FLOAT) GetDeviceCaps (hdc, LOGPIXELSX) ;
    cyDpi = (FLOAT) GetDeviceCaps (hdc, LOGPIXELSY) ;
}
else
{
    cxDpi = (FLOAT) (25.4 * GetDeviceCaps (hdc, HORZRES) /
        GetDeviceCaps (hdc, HORZSIZE)) ;
    cyDpi = (FLOAT) (25.4 * GetDeviceCaps (hdc, VERTRES) /
        GetDeviceCaps (hdc, VERTSIZE)) ;
}

pt.x = (int) (iDeciPtWidth * cxDpi / 72) ;
pt.y = (int) (iDeciPtHeight * cyDpi / 72) ;

DPtoLP (hdc, &pt, 1) ;
lf.lfHeight = - (int) (fabs (pt.y) / 10.0 + 0.5) ;
lf.lfWidth = 0 ;
lf.lfEscapement = 0 ;
lf.lfOrientation = 0 ;
lf.lfWeight = iAttributes & EZ_ATTR_BOLD ? 700 : 0 ;
lf.lfItalic = iAttributes & EZ_ATTR_ITALIC ? 1 : 0 ;
lf.lfUnderline = iAttributes & EZ_ATTR_UNDERLINE ? 1 : 0 ;
lf.lfStrikeOut = iAttributes & EZ_ATTR_STRIKEOUT ? 1 : 0 ;
lf.lfCharSet = DEFAULT_CHARSET ;
lf.lfOutPrecision = 0 ;
lf.lfClipPrecision = 0 ;
lf.lfQuality = 0 ;
lf.lfPitchAndFamily = 0 ;

lstrcpy (lf.lfFaceName, szFaceName) ;

hFont = CreateFontIndirect (&lf) ;

if (iDeciPtWidth != 0)
{

```

```

        hFont = (HFONT) SelectObject (hdc, hFont) ;
        GetTextMetrics (hdc, &tm) ;
        DeleteObject (SelectObject (hdc, hFont)) ;
        lf.lfWidth = (int) (tm.tmAveCharWidth *
        fabs (pt.x) / fabs (pt.y) + 0.5) ;
        hFont = CreateFontIndirect (&lf) ;
    }

    RestoreDC (hdc, -1) ;
    return hFont ;
}

```

EZFONT.C 只有一个函式，称为 EzCreateFont，如下所示：

```

hFont = EzCreateFont ( hdc,      szFaceName,      iDeciPtHeight,      iDeciPtWidth,
                    iAttributes, fLogRes) ;

```

函式传回字体代号。可通过呼叫 SelectObject 将该字体选入装置内容，然後呼叫 GetTextMetrics 或 GetOutlineTextMetrics 以确定字体尺寸在逻辑座标中的实际大小。在程式终止前，应该呼叫 DeleteObject 删除任何建立的字体。

szFaceName 参数可以是任何 TrueType 字体名称。您选择的字体越接近标准字体，则该字体在系统中存在的机率就越大。

第三个参数指出所需的点值，但是它的单位是十分之一。因而，如果所需要的点值为十二又二分之一，则值应为 125。

第四个参数通常应设定为零或与第三个参数相同。然而，通过将此栏位设定为不同值可以建立更宽或更窄的 TrueType 字体。它以点为单位描述了字体的宽度，有时称之为字体的「全宽 (em-width)」。不要将它与字体字元的平均宽度或其他类似的东西相混淆。在过去的排版技术中，大写字母 M 的宽度与高度是相等的。於是，「完全正方形 (em-square)」的概念产生了，这是全宽测量的起源。当字体的全宽等於字体的全高（字体的点值）时，字元宽度是字体设计者设定的宽度。宽或窄的全宽值可以产生更细或更宽的字元。

您可以将 iAttributes 参数设定为以下定义在 EZFONT.H 中的值：

```

EZ_ATTR_BOLD
EZ_ATTR_ITALIC
EZ_ATTR_UNDERLINE
EZ_ATTR_STRIKEOUT

```

可以使用 EZ_ATTR_BOLD 或 EZ_ATTR_ITALIC 或者将样式作为完整 TrueType 字体名称的一部分。

最後，我们将参数 fLogRes 设定为逻辑值 TRUE，以表示字体点值与设备的「逻辑解析度」相吻合，其中「逻辑解析度」是 GetDeviceCaps 函式使用 LOGPIXELSX 和 LOGPIXELSY 参数的传回值。另外，依据解析度的字体大小是从 HORZRES、HORZSIZE、VERTRES 和 VERTSIZE 计算出来的。这仅对於 Windows NT

下的视讯显示器才有所不同。

EzCreateFont 函式开始只进行一些用於 Windows NT 的调整。即呼叫 SetGraphicsMode 和 ModifyWorldTransform 函式，它们在 Windows 98 下不起作用。因为 Windows NT 的全球转换应该有修改字体可视大小的作用，因此在计算字体大小之前，全球转换设定为预设值——无转换。

EzCreateFont 基本上设定 LOGFONT 结构的栏位并呼叫 CreateFontIndirect，CreateFontIndirect 传回字体的代号。EzCreateFont 函式的主要任务是将字体的点值转换为 LOGFONT 结构的 lfHeight 栏位所要求的逻辑单位。其实是首先将点值转换为装置单位（图素），然後再转换为逻辑单位。为完成第一步，函式使用 GetDeviceCaps。从图素到逻辑单位的转换似乎只需简单地呼叫 DPtoLP（「从装置点到逻辑点」）函式。但是为了使 DPtoLP 转换正常工作，在以後使用建立的字体显示文字时，相同的映射方式必须有效。这就意味著应该在呼叫 EzCreateFont 函式前设定映射方式。在大多数情况下，只使用一种映射方式在视窗的特定区域绘制，因此这种要求不是什么问题。

程式 17-3 所示的 EZTEST 程式不很严格地考验了 EZFONT 档案。此程式使用上面的 EZTEST 档案，还包括了本书後面程式要使用的 FONTDEMO 档案。

程式 17-3 EZTEST

```
EZTEST.C
/*-----
EZTEST.C -- Test of EZFONT
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "ezfont.h"

TCHAR szAppName [] = TEXT ("EZTest") ;
TCHAR szTitle [] = TEXT ("EZTest: Test of EZFONT") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    HFONT          hFont ;
    int            y, iPointSize ;
    LOGFONT        lf ;
    TCHAR          szBuffer [100] ;
    TEXTMETRIC     tm ;

    // Set Logical Twips mapping mode

    SetMapMode (hdc, MM_ANISOTROPIC) ;
```

```

SetWindowExtEx (hdc, 1440, 1440, NULL) ;
SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
    GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;

    // Try some fonts

y = 0 ;
for (iPointSize = 80 ; iPointSize <= 120 ; iPointSize++)
{
    hFont = EzCreateFont ( hdc, TEXT ("Times New Roman"),
        iPointSize, 0, 0, TRUE) ;

    GetObject (hFont, sizeof (LOGFONT), &lf) ;

    SelectObject (hdc, hFont) ;
    GetTextMetrics (hdc, &tm) ;
    TextOut (hdc, 0, y, szBuffer,
        wsprintf ( szBuffer,
TEXT ("Times New Roman font of %i.%i points, ")
TEXT ("lf.lfHeight = %i, tm.tmHeight = %i"),
    iPointSize / 10, iPointSize % 10,
    lf.lfHeight, tm.tmHeight)) ;

    DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;
    y += tm.tmHeight ;
}
}

FONTDEMO.C
/*-----
-
    FONTDEMO.C --          Font Demonstration Shell Program
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "..\EZTest\EzFont.h"
#include "..\EZTest\resource.h"

extern void PaintRoutine (HWND, HDC, int, int) ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

HINSTANCE hInst ;

extern TCHAR szAppName [] ;
extern TCHAR szTitle [] ;

```

```
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR szCmdLine, int
iCmdShow)
{
    TCHAR                szResource [] = TEXT ("FontDemo") ;
    HWND                  hwnd ;
    MSG                   msg ;
    WNDCLASS              wndclass ;

    hInst = hInstance ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = szResource ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, szTitle,
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
```



```

static DOCINFO      di = { sizeof (DOCINFO), TEXT ("Font Demo:
Printing") } ;
static int          cxClient, cyClient ;
static PRINTDLG     pd = { sizeof (PRINTDLG) } ;
BOOL                fSuccess ;
HDC                 hdc, hdcPrn ;
int                 cxPage, cyPage ;
PAINTSTRUCT         ps ;

switch (message)
{
case WM_COMMAND:
    switch (wParam)
    {
        case IDM_PRINT:

            // Get printer DC

            pd.hwndOwner      = hwnd ;
            pd.Flags          = PD_RETURNDC |
PD_NOPAGENUMS | PD_NOSELECTION ;

            if (! PrintDlg (&pd))
                return 0 ;

            if (NULL == (hdcPrn = pd.hDC))
            {
                MessageBox( hwnd, TEXT ("Cannot obtain Printer DC"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                return 0 ;
            }
            // Get size of printable area of page

            cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
            cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

            fSuccess = FALSE ;

            // Do the printer page

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
            {
                PaintRoutine (hwnd, hdcPrn, cxPage, cyPage) ;

                if (EndPage (hdcPrn) > 0)

```

```

        {
            fSuccess = TRUE ;
            EndDoc (hdcPrn) ;
        }
    }
    DeleteDC (hdcPrn) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (!fSuccess)
        MessageBox (hwnd,
            TEXT ("Error encountered during printing"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

    case  IDM_ABOUT:
        MessageBox (    hwnd,        TEXT        ("Font
Demonstration Program\n")
            TEXT ("(c) Charles Petzold, 1998"),
            szAppName, MB_ICONINFORMATION | MB_OK) ;
        return 0 ;
    }
    break ;

    case  WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case  WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        PaintRoutine (hwnd, hdc, cxClient, cyClient) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case  WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

FONTDEMO.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

```

```

////////////////////////////////////
/
// Menu
FONTDEMO MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Print...",
        IDM_PRINT
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About...",
        IDM_ABOUT
    END
END
RESOURCE.H
// Microsoft Developer Studio generated include file.
// Used by FontDemo.rc

#define IDM_PRINT      40001
#define IDM_ABOUT      40002

```

EZTEST.C 中的 PaintRoutine 函式将映射方式设定为 Logical Twips, 然後建立字体范围从 8 点到 12 点 (间隔为 0.1 点) 的 Times New Roman 字体。第一次执行此程式时, 它的输出可能会使您困惑。许多行文字使用大小明显相同的字体, 并且 TEXTMETRIC 函式也报告这些字体具有相同的高度。这一切都是点阵处理的结果。显示器上的图素是不连续的, 它不能显示每一个可能的字体大小。但是, FONTDEMO 外壳程式使列印输出的字体是不同的。这里您会发现字体大小区分得更加精确。

字体的旋转

您在 PICKFONT 中可能已经实验过了, LOGFONT 结构的 lfOrientation 和 lfEscapement 栏位可以旋转 TrueType 文字。如果仔细考虑一下, 这对 GDI 不会造成多大困难, 因为围绕原点旋转坐标点的公式是公开的。

虽然 EzCreateFont 不能指定字体的旋转角度, 但是如 FONTR0T (「字体旋转」) 程式展示的那样, 在呼叫函式後, 进行调整是非常容易的。程式 17-4 显示了 FONTR0T.C 档案, 该程式也需要上面显示的 EZFONT 档案和 FONTDEMO 档案。

程式 17-4 FONTR0T

```

FONTR0T.C
/*-----

```

```

--
      FONTROT.C --      Rotated Fonts
                                   (c) Charles Petzold, 1998
-----
-*/
#include <windows.h>
#include "..\eztest\ezfont.h"

TCHAR szAppName [] = TEXT ("FontRot") ;
TCHAR szTitle [] = TEXT ("FontRot: Rotated Fonts") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    static TCHAR      szString [] = TEXT ("  Rotation") ;
    HFONT             hFont ;
    int               i ;
    LOGFONT           lf ;

    hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 540, 0, 0, TRUE) ;
    GetObject (hFont, sizeof (LOGFONT), &lf) ;
    DeleteObject (hFont) ;

    SetBkMode (hdc, TRANSPARENT) ;
    SetTextAlign (hdc, TA_BASELINE) ;
    SetViewportOrgEx (hdc, cxArea / 2, cyArea / 2, NULL) ;

    for (i = 0 ; i < 12 ; i ++)
    {
        lf.lfEscapement = lf.lfOrientation = i * 300 ;
        SelectObject (hdc, CreateFontIndirect (&lf)) ;

        TextOut (hdc, 0, 0, szString, lstrlen (szString)) ;
        DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;
    }
}

```

FONTR0T 呼叫 EzCreateFont 只是为了获得与 54 点 Times New Roman 字体相关的 LOGFONT 结构。然後，程式删除该字体。在 for 回圈中，对於每隔 30 度的角度，建立新字体并显示文字。结果如图 17-2 所示。

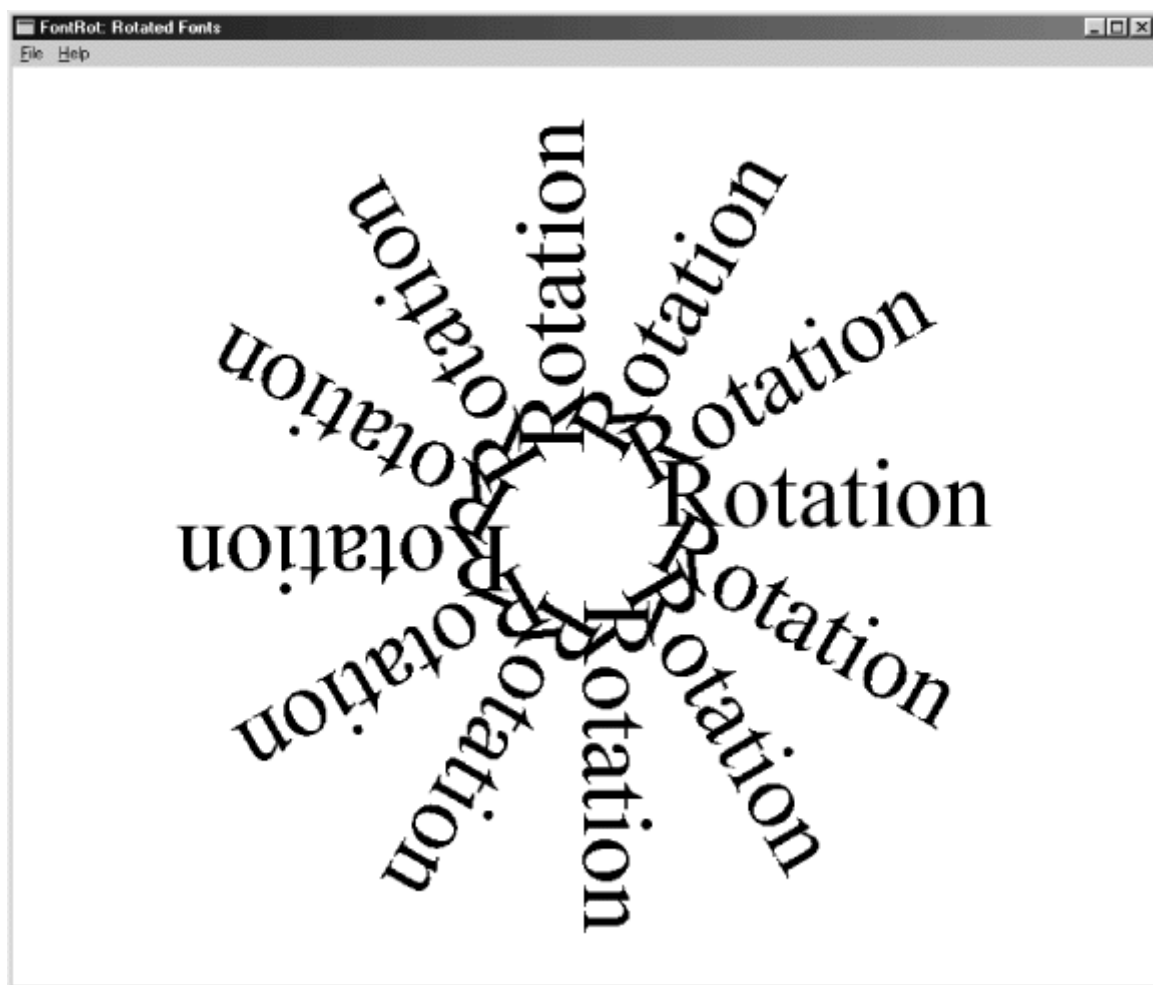


图 17-2 FONTR0T 的萤幕显示

如果您对图形旋转和其他线性转换的更专业方法感兴趣，并且知道您的程式在 Windows NT 下执行将受到限制，您可以使用 XFORM 矩阵和座标转换函式数。

字体列举

字体列举是从 GDI 中取得设备的全部有效字体列表的程式。程式可以选择其中一种字体，或将它们显示在对话方块中供使用者选择。我先简单地介绍一下列举函式，然後显示使用 ChooseFont 函式的方法，ChooseFont 降低了应用程式中进行字体列举的必要性。

列举函式

在 Windows 的早期，字体列举需要使用 EnumFonts 函式：

```
EnumFonts (hdc, szTypeFace, EnumProc, pData) ;
```

程式可以列举所有的字体（将第二个参数设定为 NULL）或只列出特定的字样。第三个参数是列举 callback 函式；第四个参数是传递给该函式的可选资料。GDI 为系统中的每种字体呼叫 callback 函式，将定义字体的 LOGFONT 和 TEXTMETRIC 结构以及一些表示字体形态的旗标传递给它。

EnumFontFamilies 函式是 Windows 3.1 下列举 TrueType 字体的函式：

```
EnumFontFamilies (hdc, szFaceName, EnumProc, pData) ;
```

通常第一次呼叫 EnumFontFamilies 时，第二个参数设定为 NULL。为每个字体系列（例如 Times New Roman）呼叫一次 EnumProc callback 函式。然後，应用程式使用该字体名称和不同的 callback 函式再次呼叫 EnumFontFamilies。GDI 为字体系列中的每种字体（例如 Times New Roman Italic）呼叫第二个 callback 函式。对于非 TrueType 字体，向 callback 函式传递 ENUMLOGFONT 结构（它是由 LOGFONT 结构加上「全名」栏位和「型态」栏位构成，「型态」栏位如文字名称「Italic」或「Bold」）和 TEXTMETRIC 结构，对于 TrueType 字体传递 NEWTEXTMETRIC 结构。NEWTEXTMETRIC 结构相对于 TEXTMETRIC 结构中的资讯添加了四个栏位。

EnumFontFamiliesEx 函式被推荐在 Windows 的 32 位元的版本下使用：

```
EnumFontFamiliesEx (hdc, &logfont, EnumProc, pData, dwFlags) ;
```

第二个参数是指向 LOGFONT 结构的指标，其中 lfCharSet 和 lfFaceName 栏位指出了所要列举的字体资讯。Callback 函式在 ENUMLOGFONTEX 和 NEWTEXTMETRICEX 结构中得到每种字体的资讯。

「ChooseFont」对话方块

在第十一章稍微介绍了 ChooseFont 的通用对话方块。现在，我们讨论字体列举，需要详细了解一下 ChooseFont 函式的内部工作原理。ChooseFont 函式得到指向 CHOOSEFONT 结构的指标以此作为它的唯一参数，并显示列出所有字体的对话方块。利用从 ChooseFont 中的传回值，LOGFONT 结构（CHOOSEFONT 结构的一部分）能够建立逻辑字体。

程式 17-5 所示的 CHOSFONT 程式展示了使用 ChooseFont 函式的方法，并显示了函式定义的 LOGFONT 结构的栏位。程式也显示了在 PICKFONT 中显示的相同字串。

程式 17-5 CHOSFONT

```
CHOSFONT.C
/*-----
---
      CHOSFONT.C --      ChooseFont Demo
                                   (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include "resource.h"
```

```

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("ChosFont") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName     = szAppName ;
    wndclass.lpszClassName   = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("ChooseFont"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam, LPARAM
lParam)
{
    static CHOOSEFONT      cf ;

```

```

static int          cyChar ;
static LOGFONT      lf ;
static TCHAR  szText[] = TEXT ("\x41\x42\x43\x44\x45 ")
                        TEXT ("\x61\x62\x63\x64\x65 ")
                        TEXT ("\xC0\xC1\xC2\xC3\xC4\xC5 ")
                        TEXT ("\xE0\xE1\xE2\xE3\xE4\xE5 ")

#ifdef UNICODE
    TEXT ("\x0390\x0391\x0392\x0393\x0394\x0395 ")
    TEXT ("\x03B0\x03B1\x03B2\x03B3\x03B4\x03B5 ")
    TEXT ("\x0410\x0411\x0412\x0413\x0414\x0415 ")
    TEXT ("\x0430\x0431\x0432\x0433\x0434\x0435 ")
    TEXT ("\x5000\x5001\x5002\x5003\x5004")
#endif

;

HDC          hdc ;
int          y ;
PAINTSTRUCT  ps ;
TCHAR        szBuffer [64] ;
TEXTMETRIC   tm ;

switch (message)
{
case WM_CREATE:

    // Get text height
    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    // Initialize the LOGFONT structure
    GetObject (GetStockObject (SYSTEM_FONT), sizeof (lf),
&lf) ;

    // Initialize the CHOOSEFONT structure
    cf.lStructSize      = sizeof (CHOOSEFONT) ;
    cf.hwndOwner        = hwnd ;
    cf.hDC              = NULL ;
    cf.lpLogFont        = &lf ;
    cf.iPointSize       = 0 ;
    cf.Flags            = CF_INITTOLOGFONTSTRUCT |
                        CF_SCREENFONTS | CF_EFFECTS ;
    cf.rgbColors        = 0 ;
    cf.lCustData        = 0 ;
    cf.lpfHook          = NULL ;
    cf.lpTemplateName  = NULL ;
    cf.hInstance        = NULL ;
    cf.lpszStyle        = NULL ;
    cf.nFontType        = 0 ;
    cf.nSizeMin         = 0 ;
    cf.nSizeMax         = 0 ;
    return 0 ;

case WM_COMMAND:

```



```

        switch (LOWORD (wParam))
        {
        case  IDM_FONT:
                if (ChooseFont (&cf))
                        InvalidateRect (hwnd, NULL, TRUE) ;
                return 0 ;
        }
        return 0 ;

case  WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

                                // Display sample text using selected
font
        SelectObject (hdc, CreateFontIndirect (&lf)) ;
        GetTextMetrics (hdc, &tm) ;
        SetTextColor (hdc, cf.rgbColors) ;
        TextOut (hdc, 0, y = tm.tmExternalLeading, szText, lstrlen (szText)) ;

                                // Display LOGFONT structure fields using system font
        DeleteObject          (SelectObject          (hdc,
GetStockObject (SYSTEM_FONT))) ;
        SetTextColor          (hdc, 0) ;

        TextOut (hdc, 0, y += tm.tmHeight, szBuffer,
wsprintf (szBuffer, TEXT ("lfHeight = %i"), lf.lfHeight)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfWidth = %i"), lf.lfWidth)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfEscapement = %i"),
        lf.lfEscapement)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfOrientation = %i"),
        lf.lfOrientation)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfWeight = %i"),lf.lfWeight)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfItalic = %i"),lf.lfItalic)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfUnderline = %i"),lf.lfUnderline)) ;

```

```

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfStrikeOut = %i"),lf.lfStrikeOut)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfCharSet = %i"),lf.lfCharSet)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfOutPrecision = %i"),
        lf.lfOutPrecision)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfClipPrecision = %i"),
        lf.lfClipPrecision)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfQuality = %i"),lf.lfQuality)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfPitchAndFamily = 0x%02X"),
        lf.lfPitchAndFamily)) ;

        TextOut (hdc, 0, y += cyChar, szBuffer,
wsprintf (szBuffer, TEXT ("lfFaceName = %s"),lf.lfFaceName)) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;
case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

CHOSFONT.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
CHOSFONT MENU DISCARDABLE
BEGIN
        MENUITEM "&Font!",
        IDM_FONT
END

RESOURCE.H
// Microsoft Developer Studio generated include file.

```

```
// Used by ChosFont.rc
#define IDM_FONT      40001
```

与一般的对话方块一样，CHOOSEFONT 结构的 Flags 栏位列出了许多选项。CHOSFONT 指定的 CF_INITLOGFONTSTRUCT 旗标使 Windows 根据传递给 ChooseFont 结构的 LOGFONT 结构对对话方块的选择进行初始化。您可以使用旗标来指定只要列出 TrueType 字体 (CF_TTONLY) 或只要列出定宽字体 (CF_FIXEDPITCHONLY) 或无符号字体 (CF_SCRIPTONLY)。也可以显示萤幕字体 (CF_SCREENFONTS)、列印字体 (CF_PRINTERFONTS) 或者两者都显示 (CF_BOTH)。在後两种情况下，CHOOSEFONT 结构的 hDC 栏位必须是印表机装置内容。CHOSFONT 程式使用 CF_SCREENFONTS 旗标。

CF_EFFECTS 旗标 (CHOSFONT 程式使用的第三个旗标) 强迫对话方块包括用於底线和删除线的核取方块并且允许选择文字的颜色。在程式码中变换文字颜色不难，您可以试一试。

注意「Font」对话方块中由 ChooseFont 显示的「Script」栏位。它让使用者选择用於特殊字体的字元集，适当的字元集 ID 在 LOGFONT 结构中传回。

ChooseFont 函式使用逻辑英寸从点值中计算 lfHeight 栏位。例如，假定您从「显示属性」对话方块中安装了「小字体」。这意味著带有视讯显示装置内容的 GetDeviceCaps 和参数 LOGPIXELSY 传回 96。如果使用 ChooseFont 选择 72 点的 Times Roman 字体，实际上是想要 1 英寸高的字体。当 ChooseFont 传回後，LOGFONT 结构的 lfHeight 栏位等於-96（注意负号），这是指字体的点值等於 96 图素，或者 1 逻辑英寸。

以上大概是我们想要知道的。但请记住以下几点：

- 如果在 Windows NT 下设定了度量映射方式，则逻辑座标与字体的实际大小不一致。例如，如果在依据度量映射方式的文字旁画一把尺，会发现它与字体不搭调。应该使用上面描述的 Logical Twips 映射方式来绘制图形，才能与字体大小一致。
- 如果要使用任何非 MM_TEXT 映射方式，请确保在把字体选入装置内容和显示文字时，没有设定映射方式。否则，GDI 会认为 LOGFONT 结构的 lfHeight 栏位是逻辑座标。
- 由 ChooseFont 设定的 LOGFONT 结构的 lfHeight 栏位总是图素值，并且它只适用於视讯显示器。当您为印表机装置内容建立字体时，必须调整 lfHeight 值。ChooseFont 函式使用 CHOOSEFONT 结构的 hDC 栏位只为获得列在对话方块中的印表机字体。此装置内容代号不影响 lfHeight 值。幸运的是，CHOOSEFONT 结构包括一个 iPointSize 栏位，它提供以十分之一

点为单位的所选字体的大小。无论是什么装置内容和映射方式，都能把这个栏位转化为逻辑大小并用于 lfHeight 栏位。在 EZFONT.C 中找到合适的程式码，您可以根据需要简化它。

另一个使用 ChooseFont 的程式是 UNICHARS，这个程式让您查看一种字体的所有字元，对于研究 Lucida Sans Unicode 字体（内定的显示字体）或 Bitstream CyberBit 字体尤其有用。UNICHARS 总是使用 TextOutW 函式来显示字体的字元，因此可以在 Windows NT 或 Windows 98 下执行它。

程式 17-6 UNICHARS

```

UNICHARS.C
/*-----
-
    UNICHARS.C --      Displays 16-bit character codes
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("UniChars") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
    = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
    = WndProc ;
    wndclass.cbClsExtra
    = 0 ;
    wndclass.cbWndExtra
    = 0 ;
    wndclass.hInstance
    = hInstance ;
    wndclass.hIcon
    = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
    = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
    = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
    = szAppName ;
    wndclass.lpszClassName
    = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requies Windows
NT!"),
                                szAppName,

```

```

MB_ICONERROR) ;

        return 0 ;

    }

    hwnd = CreateWindow (  szAppName, TEXT ("Unicode Characters"),
        WS_OVERLAPPEDWINDOW | WS_VSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    static CHOOSEFONT          cf ;
    static int                  iPage ;
    static LOGFONT              lf ;
    HDC                          hdc ;
    int                          cxChar, cyChar, x, y, i, cxLabels ;
    PAINTSTRUCT                  ps ;
    SIZE                         size ;
    TCHAR                        szBuffer [8] ;
    TEXTMETRIC                  tm ;
    WCHAR                        ch ;

    switch (message)
    {
    case  WM_CREATE:
        hdc = GetDC (hwnd) ;
        lf.lfHeight = - GetDeviceCaps (hdc, LOGPIXELSY) / 6 ; //
12 points

        lstrcpy (lf.lfFaceName, TEXT ("Lucida Sans Unicode")) ;
        ReleaseDC (hwnd, hdc) ;

        cf.lStructSize  = sizeof (CHOOSEFONT) ;
        cf.hwndOwner    = hwnd ;
        cf.lpLogFont     = &lf ;
        cf.Flags        = CF_INITTLOGFONTSTRUCT | CF_SCREENFONTS ;

```

```
        SetScrollRange          (hwnd, SB_VERT, 0, 255, FALSE) ;
        SetScrollPos            (hwnd, SB_VERT, iPage, TRUE) ;
        return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
        case IDM_FONT:
            if ( ChooseFont (&cf))
                InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;
    }
    return 0 ;

case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
        case SB_LINEUP:    iPage -= 1      ; break ;
        case SB_LINEDOWN:  iPage += 1      ; break ;
        case SB_PAGEUP:    iPage -= 16     ; break ;
        case SB_PAGEDOWN:  iPage += 16     ; break ;
        case SB_THUMBPOSITION: iPage= HIWORD (wParam); break ;
        default:
            return 0 ;
    }

    iPage = max (0, min (iPage, 255)) ;

    SetScrollPos (hwnd, SB_VERT, iPage, TRUE) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectObject (hdc, CreateFontIndirect (&lf)) ;

    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmMaxCharWidth ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;

    cxLabels = 0 ;

    for (i = 0 ; i < 16 ; i++)
    {
        wsprintf (szBuffer, TEXT (" 000%1X: "), i) ;
        GetTextExtentPoint (hdc, szBuffer, 7, &size) ;

        cxLabels = max (cxLabels, size.cx) ;
    }
}
```

```

        }

        for (y = 0 ; y < 16 ; y++)
        {
            wsprintf (szBuffer, TEXT (" %03X_:"), 16 * iPage + y) ;
            TextOut (hdc, 0, y * cyChar, szBuffer, 7) ;

            for (x = 0 ; x < 16 ; x++)
            {
                ch = (WCHAR) (256 * iPage + 16 * y + x) ;
                TextOutW (hdc, x * cxChar + cxLabels,
                        y * cyChar, &ch, 1) ;
            }
        }

        DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

UNICHARS.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
UNICHARS MENU DISCARDABLE
BEGIN
    MENUITEM "&Font!",
        IDM_FONT
END

RESOURCE.H
// Microsoft Developer Studio generated include file.
// Used by Unichars.rc

#define IDM_FONT            40001

```

段落格式

具有选择并建立逻辑字体的能力後，就可以处理文字格式了。这个程序包

括以四种方式之一来把文字的每一行放在页边距内：左对齐、向右对齐、居中或分散对齐——即从页边距的一端到另一端，文字间距相等。对于前三种方式，可以使用带有 DT_WORDBREAK 参数的 DrawText 函式，但这种方法有局限性。例如，您无法确定 DrawText 会把文字的哪个部分恰好放在矩形内。DrawText 对于一些简单任务是很方便的，但对更复杂的格式化任务，则可能要用到 TextOut。

简单文字格式

对文字的最有用的一个函式是 GetTextExtentPoint32（这个函式的名称显示了 Windows 早期版本的一些变化）。该函式根据装置内容中选入的目前字体得出字串的宽度和高度：

```
GetTextExtentPoint32 (hdc, pString, iCount, &size) ;
```

逻辑单位的文字宽度和高度在 SIZE 结构的 cx 和 cy 栏位中传回。我使用一行文字的例子，假定您把一种字体选入装置内容，现在要写入文字：

```
TCHAR * szText [] = TEXT ("Hello, how are you?") ;
```

您希望文字从垂直座标 yStart 开始，页边距由座标 xLeft 和 xRight 设定。您的任务就是计算文字开始处的水平座标的 xStart 值。

如果文字以定宽字体显示，那么这项任务就相当容易，但通常不是这样的。首先您得到字串的文字宽度：

```
GetTextExtentPoint32 (hdc, szText, lstrlen (szText), &size) ;
```

如果 size.cx 比 (xRight - xLeft) 大，这一行就太长了，不能放在页边距内。我们假定它能放进去。

要向左对齐文字，只要把 xStart 设定为与 xLeft 相等，然后写入文字：

```
TextOut (hdc, xStart, yStart, szText, lstrlen (szText)) ;
```

这很容易。现在可以把 size.cy 加到 yStart 中写下一行文字了。

要向右对齐文字，用以下公式计算 xStart：

```
xStart = xRight - size.cx ;
```

居中文字用以下公式：

```
xStart = (xLeft + xRight - size.cx) / 2 ;
```

现在开始艰巨的任务——在左右页边距内分散对齐文字。页边距之间的距离是 (xRight - xLeft)。如不调整，文字宽度就是 size.cx。两者之差

```
xRight - xLeft - size.cx
```

必须在字串的三个空格字元处平均配置。这听起来很讨厌，但还不是太糟。可以呼叫

```
SetTextJustification (hdc, xRight - xLeft - size.cx, 3)
```

来完成。第二个参数是字串内空格字元中需要分配的空间量。第三个参数是空格字元的数量，这里为 3。现在把 xStart 设定与 xLeft 相等，用 TextOut

写入文字：

```
TextOut (hdc, xStart, yStart, szText, lstrlen (szText)) ;
```

文字会在 xLeft 和 xRight 页边距之间分散对齐。

无论何时呼叫 SetTextJustification，如果空间量不能在空格字元中平均分配，它就会累积一个错误值。这将影响后面的 GetTextExtentPoint32 呼叫。每次开始新的一行，都必须通过呼叫

```
SetTextJustification (hdc, 0, 0) ;
```

来清除错误值。

使用段落

如果您处理整个段落，就必须从头开始并扫描字符串来寻找空格字元。每当碰到一个空格（或其他能用于断开一行的字元），需呼叫 GetTextExtentPoint32 来确定文字是否能放入左右页边距之间。当文字超出允许的空间时，就要退回上一个空白。现在，您已经能够确定一行的字符串了。如果想要分散对齐该行，呼叫 SetTextJustification 和 TextOut，清除错误值，并继续下一行。

显示在程式 17-7 中的 JUSTIFY1 对 Mark Twain 的《The Adventures of Huckleberry Finn》中的第一段做了这样的处理。您可以从对话方块中选择想要的字体，也可以使用功能表选项来更改对齐方式（左对齐、向右对齐、居中或分散对齐）。图 17-3 是典型的 JUSTIFY1 萤幕显示。

程式 17-7 JUSTIFY1

```
JUSTIFY1.C
/*-----
JUSTIFY1.C -- Justified Type Program #1
(c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Justify1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;
```

```
wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc          = WndProc ;
wndclass.cbClsExtra           = 0 ;
wndclass.cbWndExtra           = 0 ;
wndclass.hInstance           = hInstance ;
wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName         = szAppName ;
wndclass.lpszClassName        = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Justified Type #1"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void DrawRuler (HDC hdc, RECT * prc)
{
    static int iRuleSize [16] = {360, 72,144, 72,216, 72,144,72,
288, 72,144, 72,216, 72,144,72 } ;
    int i, j ;
    POINT ptClient ;

    SaveDC (hdc) ;

    // Set Logical Twips mapping mode
    SetMapMode (hdc, MM_ANISOTROPIC) ;
    SetWindowExtEx (hdc, 1440, 1440, NULL) ;
    SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
                     GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
```

```

        // Move the origin to a half inch from upper left

SetWindowOrgEx (hdc, -720, -720, NULL) ;
        // Find the right margin (quarter inch from right)
ptClient.x = prc->right ;
ptClient.y = prc->bottom ;
DPTtoLP (hdc, &ptClient, 1) ;
ptClient.x -= 360 ;

        // Draw the rulers
MoveToEx      (hdc, 0,          -360, NULL) ;
LineTo        (hdc, ptClient.x, -360) ;
MoveToEx      (hdc, -360,        0, NULL) ;
LineTo        (hdc, -360, ptClient.y) ;

for (i = 0, j = 0 ; i <= ptClient.x ; i += 1440 / 16, j++)
{
    MoveToEx      (hdc, i, -360, NULL) ;
    LineTo        (hdc, i, -360 - iRuleSize [j % 16]) ;
}

for (i = 0, j = 0 ; i <= ptClient.y ; i += 1440 / 16, j++)
{
    MoveToEx      (hdc, -360, i, NULL) ;
    LineTo        (hdc, -360 - iRuleSize [j % 16], i) ;
}

RestoreDC (hdc, -1) ;
}
void Justify (HDC hdc, PTSTR pText, RECT * prc, int iAlign)
{
    int          xStart, yStart, cSpaceChars ;
    PTSTR        pBegin, pEnd ;
    SIZE         size ;

    yStart = prc->top ;
    do
        // for each text line
    {
        cSpaceChars = 0 ;          // initialize number of spaces in line

        while (* pText == ' ')    // skip over leading spaces
            pText++ ;

        pBegin = pText ;          // set pointer to char at beginning of line

        do
            // until the line is known
        {

```

```
pEnd =pText ;    // set pointer to char at end of line

    // skip to next space

    while (*pText != '\0' && *pText++ != ' ' ) ;

    if (*pText == '\0')
        break ;

    // after each space encountered, calculate extents

    cSpaceChars++ ;
    GetTextExtentPoint32 (hdc, pBegin,
pText - pBegin - 1, &size) ;
    }

    while (size.cx < (prc->right - prc->left)) ;
    cSpaceChars-- ;    // discount last space at end of line

    while (*(pEnd - 1) == ' ') // eliminate trailing spaces
    {
        pEnd-- ;
        cSpaceChars-- ;
    }

    // if end of text and no space characters, set pEnd to end
    if (* pText == '\0' || cSpaceChars <= 0)
        pEnd = pText ;

    GetTextExtentPoint32 (hdc, pBegin, pEnd - pBegin, &size) ;

    switch (iAlign)    // use alignment for xStart
    {
    case IDM_ALIGN_LEFT:
        xStart = prc->left ;
        break ;

    case IDM_ALIGN_RIGHT:
        xStart = prc->right - size.cx ;
        break ;

    case IDM_ALIGN_CENTER:
        xStart = (prc->right + prc->left - size.cx) / 2 ;
        break ;

    case IDM_ALIGN_JUSTIFIED:
        if (* pText != '\0' && cSpaceChars > 0)
            SetTextJustification (hdc, prc->right-prc->left - size.cx, cSpaceChars) ;
        xStart = prc->left ;
```

```

        break ;
    }

    // display the text

    TextOut (hdc, xStart, yStart, pBegin, pEnd - pBegin) ;

    // prepare for next line

    SetTextJustification (hdc, 0, 0) ;
    yStart += size.cy ;
    pText = pEnd ;
}
while (*pText && yStart < prc->bottom - size.cy) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static CHOOSEFONTcf ;
    static DOCINFO          di = { sizeof (DOCINFO), TEXT ("Justify1:
Printing") } ;
    static int              iAlign = IDM_ALIGN_LEFT ;
    static LOGFONT          lf ;
    static PRINTDLG         pd ;
    static TCHAR            szText[] = {
TEXT ("You don't know about me, without you ")
TEXT ("have read a book by the name of \"The ")
TEXT ("Adventures of Tom Sawyer,\" but that ")
TEXT ("ain't no matter. That book was made by ")
TEXT ("Mr. Mark Twain, and he told the truth, ")
TEXT ("mainly. There was things which he ")
TEXT ("stretched, but mainly he told the truth. ")
TEXT ("That is nothing. I never seen anybody ")
TEXT ("but lied, one time or another, without ")
TEXT ("it was Aunt Polly, or the widow, or ")
TEXT ("maybe Mary. Aunt Polly -- Tom's Aunt ")
TEXT ("Polly, she is -- and Mary, and the Widow ")
TEXT ("Douglas, is all told about in that book ")
TEXT ("-- which is mostly a true book; with ")
TEXT ("some stretchers, as I said before.") } ;
    BOOL                    fSuccess ;
    HDC                     hdc, hdcPrn ;
    HMENU                   hMenu ;
    int                     iSavePointSize ;
    PAINTSTRUCT             ps ;
    RECT                    rect ;

    switch (message)

```

```

{
case WM_CREATE:

                                // Initialize the CHOOSEFONT structure

    GetObject (GetStockObject (SYSTEM_FONT), sizeof (lf),
&lf) ;

    cf.lStructSize      = sizeof (CHOOSEFONT) ;
    cf.hwndOwner        = hwnd ;
    cf.hDC              = NULL ;
    cf.lpLogFont        = &lf ;
    cf.iPointSize       = 0 ;
    cf.Flags = CF_INITTTOLOGFONTSTRUCT | CF_SCREENFONTS | CF_EFFECTS ;
    cf.rgbColors        = 0 ;
    cf.lCustData        = 0 ;
    cf.lpfHook          = NULL ;
    cf.lpTemplateName  = NULL ;
    cf.hInstance        = NULL ;
    cf.lpszStyle        = NULL ;
    cf.nFontType        = 0 ;
    cf.nSizeMin         = 0 ;
    cf.nSizeMax         = 0 ;

    return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_PRINT:
        // Get printer DC

        pd.lStructSize      = sizeof (PRINTDLG) ;
        pd.hwndOwner        = hwnd ;
        pd.Flags            = PD_RETURNDC |
PD_NOPAGENUMS | PD_NOSELECTION ;

        if (!PrintDlg (&pd))
            return 0 ;

        if (NULL == (hdcPrn = pd.hDC))
        {
            MessageBox (hwnd, TEXT ("Cannot obtain Printer DC"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;
        }

        // Set margins of 1 inch

```

```
rect.left = GetDeviceCaps (hdcPrn, LOGPIXELSX) -
            GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;

rect.top = GetDeviceCaps (hdcPrn, LOGPIXELSY)
           GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;
rect.right = GetDeviceCaps (hdcPrn, PHYSICALWIDTH) -
             GetDeviceCaps (hdcPrn, LOGPIXELSX) -
             GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;
rect.bottom= GetDeviceCaps (hdcPrn, PHYSICALHEIGHT) -
             GetDeviceCaps (hdcPrn, LOGPIXELSY) -
             GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

// Display text on printer

SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
ShowCursor (TRUE) ;

fSuccess = FALSE ;
if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
{
    // Select font using adjusted lfHeight

    iSavePointSize = lf.lfHeight ;
    lf.lfHeight = -(GetDeviceCaps (hdcPrn, LOGPIXELSY) *
                   cf.iPointSize) / 720 ;

    SelectObject (hdcPrn, CreateFontIndirect (&lf)) ;
    lf.lfHeight = iSavePointSize ;

    // Set text color

    SetTextColor (hdcPrn, cf.rgbColors) ;

    // Display text

    Justify (hdcPrn, szText, &rect, iAlign) ;

    if (EndPage (hdcPrn) > 0)
    {
        fSuccess = TRUE ;
        EndDoc (hdcPrn) ;
    }
}
ShowCursor (FALSE) ;
SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

DeleteDC (hdcPrn) ;
```

```

        if (!fSuccess)
            MessageBox (hwnd, TEXT ("Could not print text"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;

    case IDM_FONT:
        if (ChooseFont (&cf))
            InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case IDM_ALIGN_LEFT:
    case IDM_ALIGN_RIGHT:
    case IDM_ALIGN_CENTER:
    case IDM_ALIGN_JUSTIFIED:
        CheckMenuItem (hMenu, iAlign, MF_UNCHECKED) ;
        iAlign = LOWORD (wParam) ;
        CheckMenuItem (hMenu, iAlign, MF_CHECKED) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;
    }
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;
    DrawRuler (hdc, &rect) ;

    rect.left += GetDeviceCaps (hdc, LOGPIXELSX) / 2 ;
    rect.top += GetDeviceCaps (hdc, LOGPIXELSY) / 2 ;
    rect.right -= GetDeviceCaps (hdc, LOGPIXELSX) / 4 ;

    SelectObject (hdc, CreateFontIndirect (&lf)) ;
    SetTextColor (hdc, cf.rgbColors) ;

    Justify (hdc, szText, &rect, iAlign) ;

    DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;

```



```

}
JUSTIFY1.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
JUSTIFY1 MENU DISCARDABLE BEGIN POPUP "&File"
    BEGIN
        MENUITEM "&Print",
        IDM_FILE_PRINT
    END
    POPUP "&Font"
    BEGIN
        MENUITEM "&Font...",      IDM_FONT
    END
    POPUP "&Align"
    BEGIN
        MENUITEM "&Left",          IDM_ALIGN_LEFT, CHECKED
        MENUITEM "&Right",         IDM_ALIGN_RIGHT
        MENUITEM "&Centered",      IDM_ALIGN_CENTER
        MENUITEM "&Justified",     IDM_ALIGN_JUSTIFIED
    END
END

RESOURCE.H
// Microsoft Developer Studio generated include file.
// Used by Justify1.rc

#define IDM_FILE_PRINT 40001
#define IDM_FONT 40002
#define IDM_ALIGN_LEFT 40003
#define IDM_ALIGN_RIGHT 40004
#define IDM_ALIGN_CENTER 40005
#define IDM_ALIGN_JUSTIFIED 40006

```

JUSTIFY1 在显示区域的上部和左侧显示了尺规（当然单位是逻辑英寸）。尺规由 DrawRuler 函式画出。一个矩形结构定义了分散对齐文字的区域。

涉及对文字进行格式处理的大量工作由 Justify 函式实作。函式搜寻文字开始的空白，并使用 GetTextExtentPoint32 测量每一行。当行的长度超过显示区域的宽度，JUSTIFY1 传回先前的空格并使该行到达 linefeed 处。根据 iAlign 常数的值，行的对齐方式有：同左对齐、向右对齐、居中或分散对齐。

JUSTIFY1 并不完美。例如，它没有处理连字元的问题。此外，当每行少于两个字时，分散对齐的做法会失效。即使我们解决了这个不是特别难的问题，

当一个单字太长在左右边距间放不下时，程式仍不能正常运作。当然，当我们在程式中对同一行使用多种字体（如同 Windows 文书处理程式轻松做出的那样）时，情况会更复杂。还没有人声称这种处理容易，它只是比我们亲自做所有的工作容易一些。

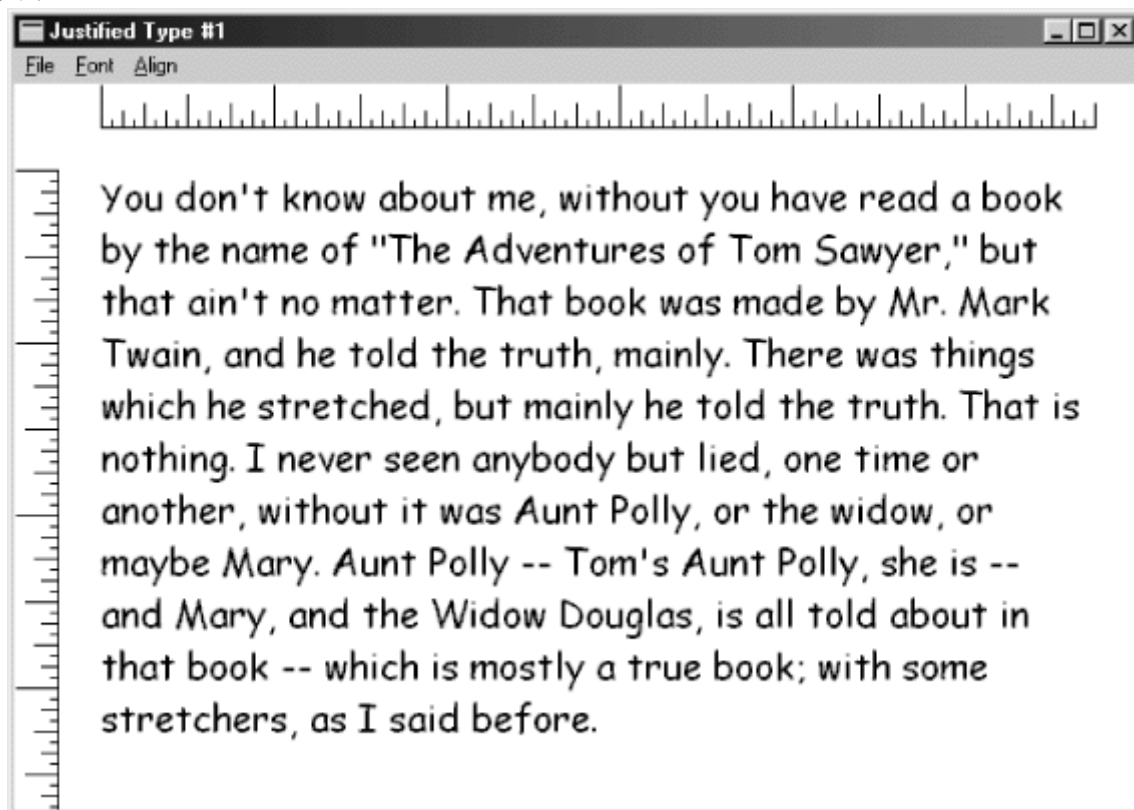


图 17-3 典型的 JUSTIFY1 萤幕显示

列印输出预览

有些字体不是为了在萤幕上查看用的，这些字体是用於列印的。通常在这种情况下，文字的萤幕预览必须与列印输出的格式精确配合。显示同样的字体、大小和字元格式是不够的。使用 TrueType 是个捷径。另外还需要将段落中的每一行在同样位置断开。这是 WYSIWYG 中的难点。

JUSTIFY1 包含一个「Print」选项，但该选项仅在页面的上、左和右边设定 1 英寸的边距。这样，格式化完全与萤幕显示器无关。这里有一个有趣的练习：在 JUSTIFY1 中更改几行程式码，使萤幕和印表机逻辑依据一个 6 英寸的格式化矩形。方法就是在 WM_PAINT 和「Print」命令处理程式中更改 rect.right 的定义。在 WM_PAINT 处理程式中，相对应叙述为：

```
rect.right = rect.left + 6 * GetDeviceCaps (hdc, LOGPIXELSX) ;
```

在「Print」命令处理程式中，相对应叙述为：

```
rect.right = rect.left + 6 * GetDeviceCaps (hdcPrn, LOGPIXELSX) ;
```

如果选择了一种 TrueType 字体，萤幕上的 linefeed 情况应与印表机的输

出相同。

但实际情况并不是这样。即使两种设备使用同样点值的相同字体，并将文字显示在同样的格式化矩形中，不同的显示解析度及凑整误差也会使 linefeed 出现在不同地方。显然，需要一种更高明的方法进行萤幕上的列印输出预览。

程式 17-8 所示的 JUSTIFY2 示范了这种方法的一个尝试。JUSTIFY2 中的程式码是依据 Microsoft 的 David Weise 所写的 TTJUST (「TrueType Justify」) 程式，而该程式又是依据本书前面的一个版本中的 JUSTIFY1 程式。为表现出这一程式中所增加的复杂性，用 Herman Melville 的《Moby-Dick》中的第一章代替了 Mark Twain 小说的摘录。

程式 17-8 JUSTIFY2

```
JUSTIFY2.C
/*-----
---
JUSTIFY2.C --          Justified Type Program #2
                        (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "resource.h"

#define OUTWIDTH 6          // Width of formatted output in inches
#define LASTCHAR 127       // Last character code used in text

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Justify2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName  = szAppName ;
    if (!RegisterClass (&wndclass))
```

```
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow ( szAppName, TEXT ("Justified Type #2"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void DrawRuler (HDC hdc, RECT * prc)
{
    static int iRuleSize [16] = {360,72,144, 72,216,72,144,72,288,72,144,
72,216,72,144, 72 } ;
    int i, j ;
    POINT ptClient ;

    SaveDC (hdc) ;
    // Set Logical Twips mapping mode
    SetMapMode (hdc, MM_ANISOTROPIC) ;
    SetWindowExtEx (hdc, 1440, 1440, NULL) ;
    SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
                    GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;

    // Move the origin to a half inch from upper left

    SetWindowOrgEx (hdc, -720, -720, NULL) ;
    // Find the right margin (quarter inch from right)
    ptClient.x = prc->right ;
    ptClient.y = prc->bottom ;
    DPTOLP (hdc, &ptClient, 1) ;
    ptClient.x -= 360 ;

    // Draw the rulers
    MoveToEx (hdc, 0, -36 0, NULL) ;
```

```

LineTo          (hdc, OUTWIDTH * 1440, -360  0) ;
MoveToEx        (hdc, -360,  0, NULL) ;
LineTo          (hdc, -360,      ptClient.y) ;

for (i = 0, j = 0 ;    i <= ptClient.x && i <= OUTWIDTH * 1440 ;
    i += 1440 / 16, j++)
{
    MoveToEx      (hdc, i, -360, NULL) ;
    LineTo        (hdc, i, -360 - iRuleSize [j % 16]) ;
}

for (i = 0, j = 0 ; i <= ptClient.y ; i += 1440 / 16, j++)
{
    MoveToEx      (hdc, -360, i, NULL) ;
    LineTo        (hdc, -360 - iRuleSize [j % 16], i) ;
}

RestoreDC (hdc, -1) ;
}

/*-----
-
    GetCharDesignWidths:  Gets character widths for font as large as the
                           original
design size
-----
*/

UINT GetCharDesignWidths (HDC hdc, UINT uFirst, UINT uLast, int * piWidths)
{
    HFONT          hFont, hFontDesign ;
    LOGFONT        lf ;
    OUTLINETEXTMETRIC otm ;

    hFont = GetCurrentObject (hdc, OBJ_FONT) ;
    GetObject (hFont, sizeof (LOGFONT), &lf) ;

    // Get outline text metrics (we'll only be using a field that is
    // independent of the DC the font is selected into)

    otm.otmSize = sizeof (OUTLINETEXTMETRIC) ;
    GetOutlineTextMetrics (hdc, sizeof (OUTLINETEXTMETRIC), &otm) ;

    // Create a new font based on the design size
    lf.lfHeight      = - (int) otm.otmEMSquare ;
    lf.lfWidth       = 0 ;
    hFontDesign      = CreateFontIndirect (&lf) ;

```

```

        // Select the font into the DC and get the character widths

SaveDC (hdc) ;
SetMapMode (hdc, MM_TEXT) ;
SelectObject (hdc, hFontDesign) ;

GetCharWidth (hdc, uFirst, uLast, piWidths) ;
SelectObject (hdc, hFont) ;
RestoreDC (hdc, -1) ;

        // Clean up
DeleteObject (hFontDesign) ;
return otm.otmEMSSquare ;
}

/*-----
   GetScaledWidths: Gets floating point character widths for selected
                        font size
-----*/

void GetScaledWidths (HDC hdc, double * pdWidths)
{
    double          dScale ;
    HFONT           hFont ;
    int              aiDesignWidths [LASTCHAR + 1] ;
    int              i ;
    LOGFONT          lf ;
    UINT             uEMSSquare ;

        // Call function above

    uEMSSquare = GetCharDesignWidths (hdc, 0, LASTCHAR, aiDesignWidths) ;
        // Get LOGFONT for current font in device context
    hFont = GetCurrentObject (hdc, OBJ_FONT) ;
    GetObject (hFont, sizeof (LOGFONT), &lf) ;
        // Scale the widths and store as floating point values
    dScale = (double) -lf.lfHeight / (double) uEMSSquare ;
    for ( i = 0 ; i <= LASTCHAR ; i++)
        pdWidths[i] = dScale * aiDesignWidths[i] ;
}

/*-----
--
   GetTextExtentFloat: Calculates text width in floating point
-----*/

```

```

double GetTextExtentFloat (double * pdWidths, PTSTR psText, int iCount)
{
    double      dWidth = 0 ;
    int         i ;

    for ( i = 0 ; i < iCount ; i++)
        dWidth += pdWidths [psText[i]] ;

    return dWidth ;
}

/*-----
--
Justify: Based on design units for screen/printer compatibility
-----
-*/

void Justify (HDC hdc, PTSTR pText, RECT * prc, int iAlign)
{
    double      dWidth, adWidths[LASTCHAR + 1] ;
    int         xStart, yStart, cSpaceChars ;
    PTSTR       pBegin, pEnd ;
    SIZE        size ;

    // Fill the adWidths array with floating point character widths

    GetScaledWidths (hdc, adWidths) ;
    yStart = prc->top ;
    do
        // for each text line
    {
        cSpaceChars = 0 ;      // initialize number of spaces in line

        while (*pText == ' ') // skip over leading spaces
            pText++ ;

        pBegin = pText ;      // set pointer to char at beginning of line

        do
            // until the line is known
        {
            pEnd = pText ;    // set pointer to char at end of line

            // skip to next space

            while (*pText != '\0' && *pText++ != ' ') ;

            if (*pText == '\0')
                break ;
        }
    }
}

```

```

        // after each space encountered, calculate extents

        cSpaceChars++ ;
        dWidth = GetTextExtentFloat (adWidths,  pBegin,
                                         pText - pBegin - 1) ;
    }
    while (dWidth < (double) (prc->right - prc->left)) ;

    cSpaceChars-- ;          // discount last space at end of line

    while (*(pEnd - 1) == ' ') // eliminate trailing spaces
    {
        pEnd-- ;
        cSpaceChars-- ;
    }

    // if end of text and no space characters, set pEnd to end
    if (*pText == '\\0' || cSpaceChars <= 0)
        pEnd = pText ;

        // Now get integer extents

    GetTextExtentPoint32(hdc, pBegin, pEnd - pBegin, &size) ;

    switch (iAlign)          // use alignment for xStart
    {
    case  IDM_ALIGN_LEFT:
        xStart = prc->left ;
        break ;

    case  IDM_ALIGN_RIGHT:
        xStart = prc->right - size.cx ;
        break ;

    case  IDM_ALIGN_CENTER:
        xStart = (prc->right + prc->left - size.cx) / 2 ;
        break ;

    case  IDM_ALIGN_JUSTIFIED:
        if (*pText != '\\0' && cSpaceChars > 0)
            SetTextJustification (hdc,
                prc->right - prc->left - size.cx,
                cSpaceChars) ;
        xStart = prc->left ;
        break ;
    }

    // display the text

```



```

        TextOut (hdc, xStart, yStart, pBegin, pEnd - pBegin) ;

        // prepare for next line

        SetTextJustification (hdc, 0, 0) ;
        yStart += size.cy ;
        pText = pEnd ;
    }
    while (*pText && yStart < prc->bottom - size.cy) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam, LPARAM
lParam)
{
    static      CHOOSEFONT cf ;
    static      DOCINFO      di = { sizeof (DOCINFO), TEXT
("Justify2: Printing") } ;
    static      int          iAlign = IDM_ALIGN_LEFT ;
    static      LOGFONT      lf ;
    static      PRINTDLG      pd ;
    static      TCHAR        szText[] = {
TEXT ("Call me Ishmael. Some years ago -- never ")
TEXT ("mind how long precisely -- having little ")
TEXT ("or no money in my purse, and nothing ")
TEXT ("particular to interest me on shore, I ")
TEXT ("thought I would sail about a little and ")
TEXT ("see the watery part of the world. It is ")
TEXT ("a way I have of driving off the spleen, ")
TEXT ("and regulating the circulation. Whenever ")
TEXT ("I find myself growing grim about the ")
TEXT ("mouth; whenever it is a damp, drizzly ")
TEXT ("November in my soul; whenever I find ")
TEXT ("myself involuntarily pausing before ")
TEXT ("coffin warehouses, and bringing up the ")
TEXT ("rear of every funeral I meet; and ")
TEXT ("especially whenever my hypos get such an ")
TEXT ("upper hand of me, that it requires a ")
TEXT ("strong moral principle to prevent me ")
TEXT ("from deliberately stepping into the ")
TEXT ("street, and methodically knocking ")
TEXT ("people's hats off -- then, I account it ")
TEXT ("high time to get to sea as soon as I ")
TEXT ("can. This is my substitute for pistol ")
TEXT ("and ball. With a philosophical flourish ")
TEXT ("Cato throws himself upon his sword; I ")
TEXT ("quietly take to the ship. There is ")
TEXT ("nothing surprising in this. If they but ")
TEXT ("knew it, almost all men in their degree, ")

```

```

TEXT ("some time or other, cherish very nearly ")
TEXT ("the same feelings towards the ocean with ")
TEXT ("me.") } ;

    BOOL                fSuccess ;
    HDC                 hdc, hdcPrn ;
    HMENU               hMenu ;
    int                 iSavePointSize ;
    PAINTSTRUCT         ps ;
    RECT                rect ;

switch (message)
{
case WM_CREATE:

                                // Initialize the CHOOSEFONT structure

    hdc = GetDC (hwnd) ;
    lf.lfHeight = - GetDeviceCaps (hdc, LOGPIXELSY) / 6 ;
    lf.lfOutPrecision = OUT_TT_ONLY_PRECIS ;
    lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;
    ReleaseDC (hwnd, hdc) ;

    cf.lStructSize          = sizeof (CHOOSEFONT) ;
    cf.hwndOwner            = hwnd ;
    cf.hDC                  = NULL ;
    cf.lpLogFont            = &lf ;
    cf.iPointSize           = 120 ;

                                // Set flags for TrueType only!

    cf.Flags                =    CF_INITTOLOGFONTSTRUCT

CF_SCREENFONTS |

                                CF_TTONLY | CF_EFFECTS ;
    cf.rgbColors            = 0 ;
    cf.lCustData            = 0 ;
    cf.lpfHook             = NULL ;
    cf.lpTemplateName      = NULL ;
    cf.hInstance            = NULL ;
    cf.lpszStyle            = NULL ;
    cf.nFontType            = 0 ;
    cf.nSizeMin             = 0 ;
    cf.nSizeMax             = 0 ;

    return 0 ;

case WM_COMMAND:

    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))

```

```

        {
        case IDM_FILE_PRINT:
            // Get printer DC

            pd.lStructSize = sizeof (PRINTDLG) ;
            pd.hwndOwner   = hwnd ;
pd.Flags      = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

            if (!PrintDlg (&pd))
                return 0 ;

            if (NULL == (hdcPrn = pd.hDC))
            {
MessageBox      (hwnd, TEXT ("Cannot obtain Printer DC"),
                 szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                return 0 ;
            }
            // Set margins for OUTWIDTH inches wide

            rect.left = (GetDeviceCaps (hdcPrn, PHYSICALWIDTH)

-
GetDeviceCaps (hdcPrn, LOGPIXELSX) * OUTWIDTH) / 2
-
            GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;

            rect.right =      rect.left +
GetDeviceCaps (hdcPrn, LOGPIXELSX) * OUTWIDTH ;

            // Set margins of 1 inch at top and bottom

            rect.top  = GetDeviceCaps (hdcPrn, LOGPIXELSY) -
GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

            rect.bottom = GetDeviceCaps (hdcPrn, PHYSICALHEIGHT)

-

GetDeviceCaps (hdcPrn, LOGPIXELSY) -
GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

            // Display text on printer

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            fSuccess = FALSE ;

            if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
            {
                // Select font using adjusted lfHeight

```

```

        iSavePointSize = lf.lfHeight ;
        lf.lfHeight = -(GetDeviceCaps (hdcPrn, LOGPIXELSY) *
                        cf.iPointSize) / 720 ;

        SelectObject (hdcPrn, CreateFontIndirect (&lf)) ;
                        lf.lfHeight = iSavePointSize ;

                                // Set text color

        SetTextColor (hdcPrn, cf.rgbColors) ;

                                // Display text

        Justify (hdcPrn, szText, &rect, iAlign) ;

                                if (EndPage (hdcPrn) > 0)
                                {
                                                fSuccess = TRUE ;
                                                EndDoc (hdcPrn) ;
                                }
        }

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        DeleteDC (hdcPrn) ;

        if (!fSuccess)
        MessageBox (hwnd, TEXT ("Could not print text"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;

    case  IDM_FONT:

        if (ChooseFont (&cf))
            InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case  IDM_ALIGN_LEFT:
    case  IDM_ALIGN_RIGHT:
    case  IDM_ALIGN_CENTER:
    case  IDM_ALIGN_JUSTIFIED:
        CheckMenuItem (hMenu, iAlign, MF_UNCHECKED) ;
        iAlign = LOWORD (wParam) ;
        CheckMenuItem (hMenu, iAlign, MF_CHECKED) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    }
    return 0 ;

case  WM_PAINT:

```

```

        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;
        DrawRuler (hdc, &rect) ;

        rect.left  += GetDeviceCaps (hdc, LOGPIXELSX) / 2 ;
        rect.top   += GetDeviceCaps (hdc, LOGPIXELSY) / 2 ;
        rect.right = rect.left + OUTWIDTH * GetDeviceCaps (hdc,
LOGPIXELSX) ;

        SelectObject (hdc, CreateFontIndirect (&lf)) ;
        SetTextColor (hdc, cf.rgbColors) ;

        Justify (hdc, szText, &rect, iAlign) ;

        DeleteObject (SelectObject (hdc, GetStockObject
(SYSTEM_FONT))) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

JUSTIFY2.RC
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
JUSTIFY2 MENU DISCARDABLE BEGIN POPUP "&File"
    BEGIN
        MENUITEM "&Print",          IDM_FILE_PRINT
    END
    POPUP "&Font"
    BEGIN
        MENUITEM "&Font...",      IDM_FONT
    END
    POPUP "&Align"
    BEGIN
        MENUITEM "&Left",          IDM_ALIGN_LEFT, CHECKED
        MENUITEM "&Right",         IDM_ALIGN_RIGHT
        MENUITEM "&Centered",       IDM_ALIGN_CENTER
        MENUITEM "&Justified",      IDM_ALIGN_JUSTIFIED

```

```

        END
END
RESOURCE.H
// Microsoft Developer Studio generated include file.
// Used by Justify2.rc

#define IDM_FILE_PRINT 40001
#define IDM_FONT 40002
#define IDM_ALIGN_LEFT 40003
#define IDM_ALIGN_RIGHT 40004
#define IDM_ALIGN_CENTER 40005
#define IDM_ALIGN_JUSTIFIED 40006

```

JUSTIFY2 仅使用 TrueType 字体。在它的 GetCharDesignWidths 函式中，程式使用 GetOutlineTextMetrics 函式取得一个表面上似乎不重要的资讯，即 OUTLINETEXTMETRIC 的 otmEMSsquare 栏位。

TrueType 字体在全方 (em-square) 的网格上设计 (如我说过「em」是指一种方块型态的宽度，M 在宽度上等於字体点值的大小)。任何特定 TrueType 字体的所有字元都是在同样的网格上设计的，虽然这些字元通常有不同的宽度。OUTLINETEXTMETRIC 结构的 otmEMSsquare 栏位给出了任意特定字体的这种全方形式的大小。您会发现：对于大多数 TrueType 字体，otmEMSsquare 栏位等於 2048，这意味著字体是在 2048 2048 的网格上设计的。

关键在於：可以为想要使用的特定 TrueType 字体名称设定一个 LOGFONT 结构，其 lfHeight 栏位等於 otmEMSsquare 值的负数。在建立字体并将其选入装置内容後，可呼叫 GetCharWidth。该函式以逻辑单位提供字体中单个字元的宽度。通常，因为这些字元被缩放为不同的字体大小，所以字元宽度并不准确。但使用依据 otmEMSsquare 大小的字体，这些宽度总是与任何装置内容无关的精确整数。

GetCharDesignWidths 函式以这种方式获得原始的字元设计宽度，并将它们储存在整数阵列中。JUSTIFY2 程式在自己的文字中仅使用 ASCII 字元，因此，这个阵列不需要很大。GetScaledWidths 函式将这些整数型态宽度转变为依据设备逻辑座标中字体的实际点值的浮点宽度。GetTextExtentFloat 函式使用这些浮点宽度计算整个字串的宽度。这是新的 Justify 函式用以计算文字行宽度的操作。

有趣的东西

根据外形轮廓表示字体字元提供了将字体与其他图形技术相结合的可能性。前面我们讨论了旋转字体的方式。这里讲述一些其他技巧。继续之前，先

了解两个重要的预备知识：绘图路径和扩展画笔。

GDI 绘图路径

绘图路径是储存在 GDI 内的直线和曲线的集合。绘图路径是在 Windows 的 32 位元版本中发表的。绘图路径看上去类似於区域，我们确实可以将绘图路径转换为区域，并使用绘图路径进行剪裁。但随后我们会发现两者的不同。

要定义绘图路径，可先简单呼叫

```
BeginPath (hdc) ;
```

进行该呼叫之後，所画的任何线（例如，直线、弧及贝塞尔曲线）将作为绘图路径储存在 GDI 内部，不被显示到装置内容上。绘图路径经常由连结起来的线组成。要制作连结线，应使用 LineTo、PolylineTo 和 BezierTo 函式，这些函式都以目前位置为起点划线。如果使用 MoveToEx 改变了目前位置，或呼叫其他的画线函式，或者呼叫了会导致目前位置改变的视窗/视埠函式，您就在整个绘图路径中建立了一个新的子绘图路径。因此，绘图路径包含一或多个子绘图路径，每一个子绘图路径是一系列连结的线段。

绘图路径中的每个子绘图路径可以是敞开的或封闭的。封闭子绘图路径之第一条连结线的第一个点与最後一条连结线的最後一点相同，并且子绘图路径通过呼叫 CloseFigure 结束。如果必要的话，CloseFigure 将用一条直线封闭子绘图路径。随後的画线函式将开始一个新的子绘图路径。最後，通过下面的呼叫结束绘图路径定义：

```
EndPath (hdc) ;
```

这时，接著呼叫下列五个函式之一：

```
StrokePath (hdc) ;  
FillPath (hdc) ;  
StrokeAndFillPath (hdc) ;  
hRgn = PathToRegion (hdc) ;  
SelectClipPath (hdc, iCombine) ;
```

这些函式中的每一个都会在绘图路径定义完成後，将其清除。

StrokePath 使用目前画笔绘制绘图路径。您可能会好奇：绘图路径上的点有哪些？为什么不能跳过这些绘图路径片段正常地画线？稍後我会告诉您原因。

另外四个函式用直线关闭任何敞开的绘图路径。FillPath 依照目前的多边填充模式使用目前画刷填充绘图路径。StrokeAndFillPath 一次完成这两项工作。也可将绘图路径转换为区域，或者将绘图路径用於某个剪裁区域。iCombine 参数是 CombineRgn 函式使用的 RGN_ 系列常数之一，它指出了绘图路径与目前剪裁区域的结合方式。

用於填充或剪取时，绘图路径比绘图区域更灵活，这是因为绘图区域仅能由矩形、椭圆及多边形的组合定义；绘图路径可由贝塞尔曲线定义，至少在 Windows NT 中还可由弧线组成。在 GDI 中，绘图路径和区域的储存也完全不同。绘图路径是直线及曲线定义的集合；而绘图区域（通常意义上）是扫描线的集合。

扩展画笔

在呼叫 `StrokePath` 时，使用目前画笔绘制绘图路径。在第四章讨论了用以建立画笔物件的 `CreatePen` 函式。伴随绘图路径的发表，Windows 也支援一个称为 `ExtCreatePen` 的扩展画笔函式呼叫。该函式揭示了其建立绘图路径以及使用绘图路径要比不使用绘图路径画线有用。`ExtCreatePen` 函式如下所示：

```
hPen = ExtCreatePen (iStyle, iWidth, &lBrush, 0, NULL) ;
```

您可以使用该函式正常地绘制线段，但在这种情况下 Windows 98 不支援一些功能。甚至用以显示绘图路径时，Windows 98 仍不支援一些功能，这就是上面函式的最後两个参数被设定为 0 及 `NULL` 的原因。

對於 `ExtCreatePen` 的第一个参数，可使用第四章中所讨论的用在 `CreatePen` 上的所有样式。您可使用 `PS_GEOMETRIC` 另外组合这些样式（其中 `iWidth` 参数以逻辑单位表示线宽并能够转换），或者使用 `PS_COSMETIC`（其中 `iWidth` 参数必须是 1）。Windows 98 中，虚线或点画线样式的画笔必须是 `PS_COSMETIC`，在 Windows NT 中取消了这个限制。

`CreatePen` 的一个参数表示颜色；`ExtCreatePen` 的相应参数不只表示颜色，它还使用画刷给 `PS_GEOMETRIC` 画笔内部着色。该画刷甚至能透过点阵图定义。

在绘制宽线段时，我们可能要关注线段端点的外观。在连结直线或曲线时，可能还要关注线段间连结点的外观。画笔由 `CreatePen` 建立时，这些端点及连结点通常是圆形的；使用 `ExtCreatePen` 建立画笔时我们可以选择。（实际上，在 Windows 98 中，只有在使用画笔实作绘图路径时我们可以选择；在 Windows NT 中要更加灵活）。宽线段的端点可以使用 `ExtCreatePen` 中的下列画笔样式定义：

```
PS_ENDCAP_ROUND  
PS_ENDCAP_SQUARE  
PS_ENDCAP_FLAT
```

「square」样式与「flat」样式的不同点是：前者将线伸展到一半宽。与端点类似，绘图路径中线段间的连结点可通过如下样式设定：

```
PS_JOIN_ROUND  
PS_JOIN_BEVEL  
PS_JOIN_MITER
```

「bevel」样式将连结点切断；「miter」样式将连结点变为箭头。程式 17-9

所示的 ENDJOIN 是对此的一个较好的说明。

程式 17-9 ENDJOIN

```

ENDJOIN.C
/*-----
-
        ENDJOIN.C --      Ends and Joins Demo
                                (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("EndJoin") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Ends and Joins Demo"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;

```

```

UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static          int          iEnd[]          =
    {PS_ENDCAP_ROUND,PS_ENDCAP_SQUARE,PS_ENDCAP_FLAT } ;
    static int iJoin[]={PS_JOIN_ROUND,   PS_JOIN_BEVEL,PS_JOIN_MITER } ;
    static int  cxClient, cyClient ;
    HDC          hdc ;
    int           i ;
    LOGBRUSH      ib ;
    PAINTSTRUCT   ps ;

    switch (iMsg)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        SetMapMode (hdc, MM_ANISOTROPIC) ;
        SetWindowExtEx (hdc, 100, 100, NULL) ;
        SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

        lb.lbStyle = BS_SOLID ;
        lb.lbColor = RGB (128, 128, 128) ;
        lb.lbHatch = 0 ;

        for (i = 0 ; i < 3 ; i++)
        {
            SelectObject (hdc, ExtCreatePen (PS_SOLID | PS_GEOMETRIC
|
                iEnd [i] | iJoin [i], 10, &lb, 0, NULL)) ;
                BeginPath (hdc) ;
                MoveToEx          (hdc, 10 + 30 * i, 25, NULL) ;
                LineTo            (hdc, 20 + 30 * i, 75) ;
                LineTo            (hdc, 30 + 30 * i, 25) ;

```

```

        EndPath (hdc) ;
        StrokePath (hdc) ;

        DeleteObject (
            SelectObject (hdc, GetStockObject (BLACK_PEN))) ;

        MoveToEx (hdc, 10 + 30 * i, 25, NULL) ;
        LineTo (hdc, 20 + 30 * i, 75) ;
        LineTo (hdc, 30 + 30 * i, 25) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

```

程式使用上述端点和连结点样式画了三条 V 形的宽线段。程式也使用备用黑色画笔画了三条同样的线。这样就将宽线与通常的细线做了比较。结果如图 17-4 所示。

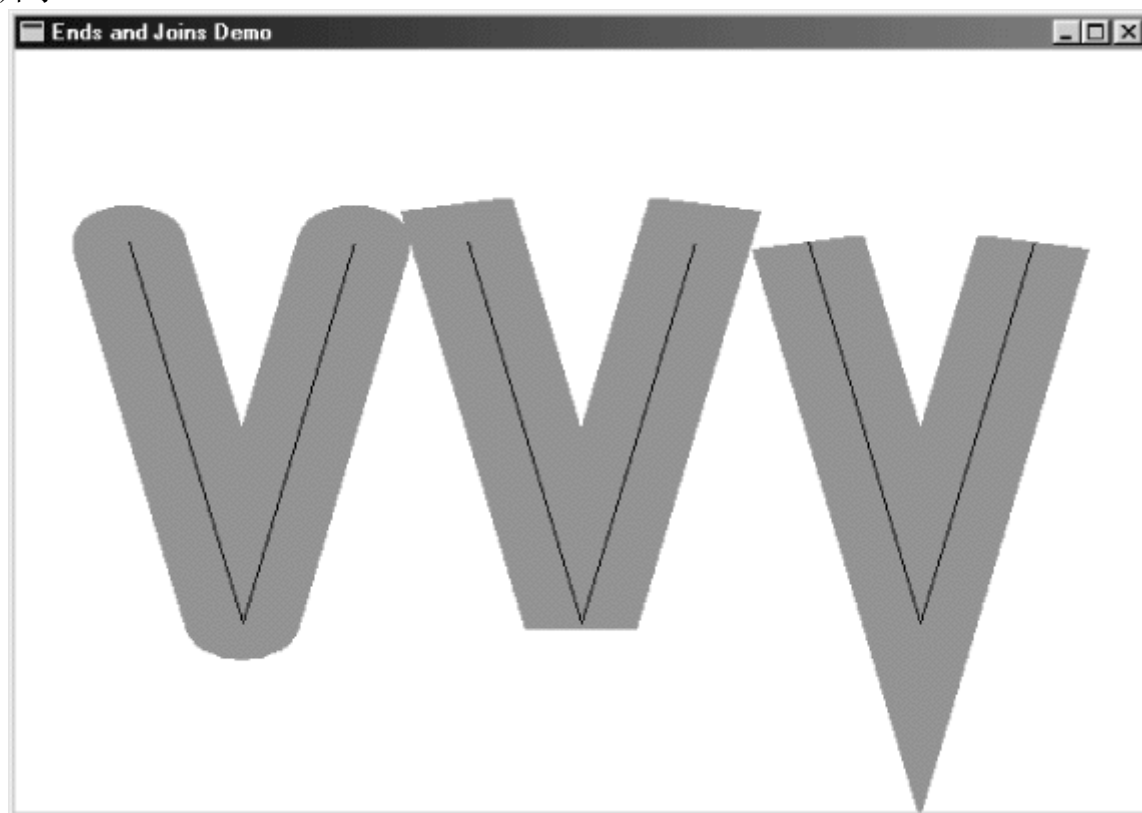


图 17-4 ENDJOIN 的萤幕显示

现在大家该明白为什么 Windows 支援 StrokePath 函式了：如果分别画两条直线，GDI 不得不在每一条线上使用端点。只有在绘图路径定义中，GDI 知道线

段是连结的并使用线段的连结点。

四个范例程式

这究竟有什么好处呢？仔细考虑一下：轮廓字体的字元由一系列座标值定义，这些座标定义了直线和转折线。因而，直线及曲线能成为绘图路径定义的一部分。

确实可以！程式 17-10 所示的 FONTOUT1 程式对此做了展示。

程式 17-10 FONTOUT1

```
FONTOUT1.C
/*-----
---
      FONTOUT1.C --          Using Path to Outline Font
                              (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include "..\eztest\ezfont.h"

TCHAR szAppName [] = TEXT ("FontOut1") ;
TCHAR szTitle [] = TEXT ("FontOut1: Using Path to Outline Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    static TCHAR      szString [] = TEXT ("Outline") ;
    HFONT              hFont ;
    SIZE               size ;

    hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;
    SelectObject (hdc, hFont) ;
    GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
    BeginPath (hdc) ;
    TextOut (hdc, ( cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
                                                    szString,          lstrlen
(szString)) ;
    EndPath (hdc) ;
    StrokePath (hdc) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
    DeleteObject (hFont) ;
}
```

此程式和本章後面的程式都使用了前面所示的 EZFONT 和 FONTDEMO 档案。

程式建立了 144 点的 TrueType 字体并呼叫 GetTextExtentPoint32 函式取得文字方块的大小。然後，呼叫绘图路径定义中的 TextOut 函式使文字在显示

区域视窗中处于中心的位置。因为对 TextOut 函式的呼叫是被绘图路径设定命令所包围的（即 BeginPath 和 EndPath 呼叫之间）程式中进行的，GDI 不立即显示文字。相反，程式将字元轮廓储存在绘图路径定义中。

在绘图路径定义结束后，FONTOUT1 呼叫 StrokePath。因为装置内容中未选入指定的画笔，所以 GDI 仅仅使用内定画笔绘制字元轮廓，如图 17-5 所示。

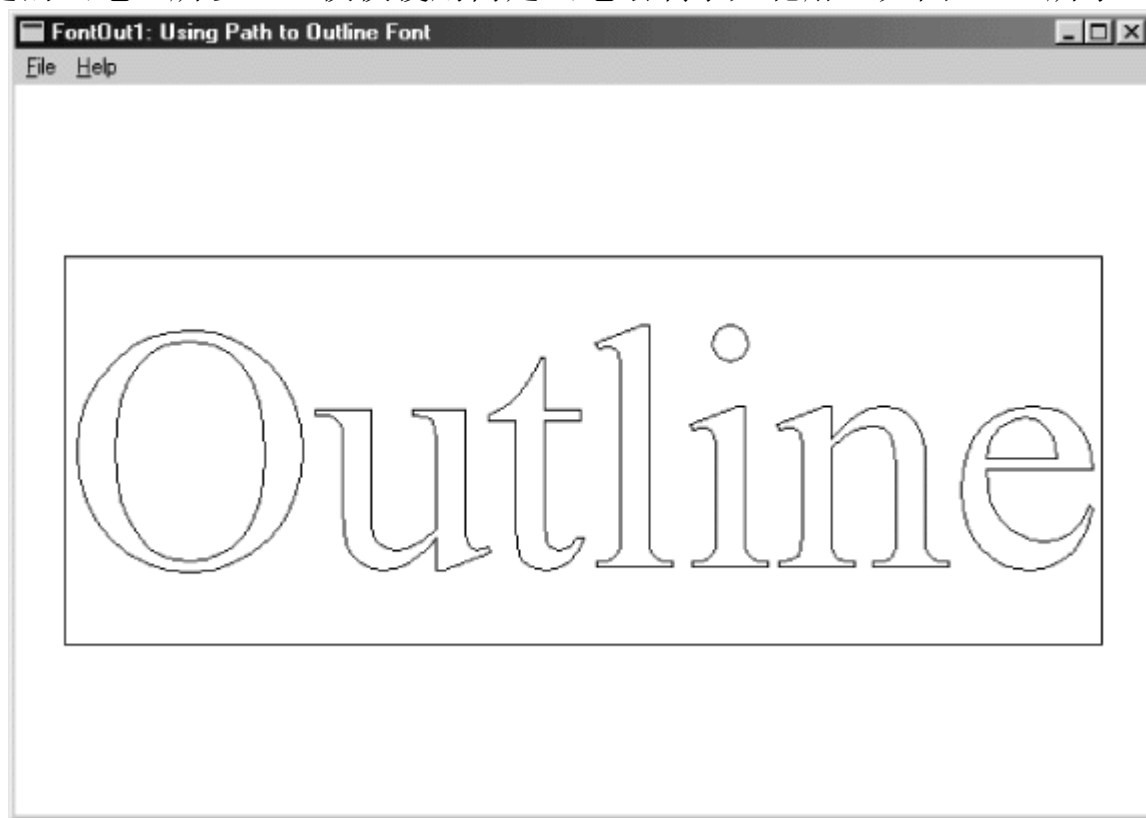


图 17-5 FONTOUT1 的萤幕显示

现在我们都得到什么呢？我们已经获得了所期望的轮廓字元，但是字符串外面为什么会围绕著矩形呢？

回想一下，文字背景模式使用内定的 OPAQUE，而不是 TRANSPARENT。该矩形就是文字方块的轮廓。这清晰地展示了在内定的 OPAQUE 模式下 GDI 绘制文字时所使用的两个步骤：首先绘制一个填充的矩形，接著绘制字元。文字方块矩形的轮廓也因此成为绘图路径的一部分。

使用 ExtCreatePen 函式就能够使用内定画笔以外的东西绘制字体字元的轮廓。程式 17-11 所示的 FONTOUT2 对此做了展示。

程式 17-11 FONTOUT2

```
FONTOUT2.C
/*-----
---
FONTOUT2.C --          Using Path to Outline Font
                                   (c) Charles Petzold, 1998
-----
*/
```

```
#include <windows.h>
#include "..\eztest\ezfont.h"

TCHAR szAppName [] = TEXT ("FontOut2") ;
TCHAR szTitle [] = TEXT ("FontOut2: Using Path to Outline Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    static TCHAR      szString [] = TEXT ("Outline") ;
    HFONT             hFont ;
    LOGBRUSH          lb ;
    SIZE               size ;

    hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;
    SelectObject (hdc, hFont) ;
    SetBkMode (hdc, TRANSPARENT) ;

    GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
    BeginPath (hdc) ;
    TextOut (hdc, ( cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
             szString, lstrlen (szString)) ;
    EndPath (hdc) ;
    lb.lbStyle = BS_SOLID ;
    lb.lbColor = RGB (255, 0, 0) ;
    lb.lbHatch = 0 ;

    SelectObject (hdc, ExtCreatePen (PS_GEOMETRIC | PS_DOT,
                                     GetDeviceCaps (hdc, LOGPIXELSX) / 24, &lb, 0, NULL)) ;
    StrokePath (hdc) ;
    DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
    DeleteObject (hFont) ;
}
```

此程式呼叫 StrokePath 之前建立 (并选入装置内容) 一个 3 点 (1/24 英寸) 宽的红色点线笔。程式在 Windows NT 下执行时, 结果如图 17-6 所示。Windows 98 不支援超过 1 图素宽的非实心笔, 因此 Windows 98 将以实心的红色笔绘制。

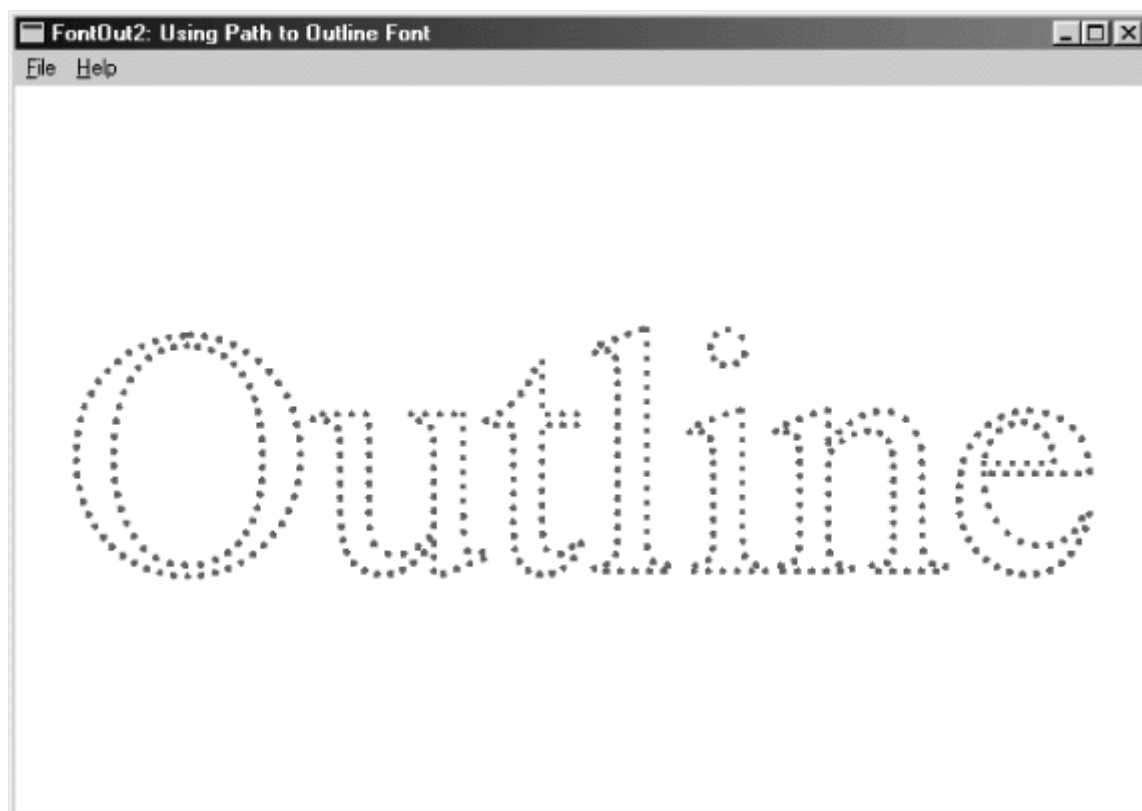


图 17-6 FONTOUT2 的萤幕显示

您也可以使用绘图路径定义填充区域。请用前面两个程式所示的方法建立绘图路径，选择一种填充图案，然後呼叫 FillPath。能呼叫的另一个函式是 StrokeAndFillPath，它绘制绘图路径的轮廓并用一个函式呼叫将其填充。

StrokeAndFillPath 函式如程式 17-12 FONTFILL 所展示。

程式 17-12 FONTFILL

```

FONTFILL.C
/*-----
--
--      FONTFILL.C --      Using Path to Fill Font
--                                     (c) Charles Petzold, 1998
--
--*/

#include <windows.h>
#include "..\eztest\ezfont.h"

TCHAR szAppName [] = TEXT ("FontFill") ;
TCHAR szTitle [] = TEXT ("FontFill: Using Path to Fill Font") ;
void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    static TCHAR szString [] = TEXT ("Filling") ;
    HFONT          hFont ;
    SIZE           size ;

    hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;

```

```
SelectObject (hdc, hFont) ;
SetBkMode (hdc, TRANSPARENT) ;

GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
BeginPath (hdc) ;
TextOut (hdc, ( cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
                                                szString,          lstrlen
(szString)) ;
EndPath (hdc) ;
SelectObject (hdc, CreateHatchBrush (HS_DIAGCROSS, RGB (255, 0, 0))) ;
SetBkColor (hdc, RGB (0, 0, 255)) ;
SetBkMode (hdc, OPAQUE) ;

StrokeAndFillPath (hdc) ;
DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
DeleteObject (hFont) ;
}
```

FONTFILL 使用内定画笔绘制绘图路径的轮廓,但使用 HS_DIAGCROSS 样式建立红色的阴影画刷。注意程式在建立绘图路径时将背景模式设定为 TRANSPARENT,在填充绘图路径时又将其重设为 OPAQUE,这样它能够为区域图案使用蓝色的背景颜色。结果如图 17-7 所示。

您可能想在本程式中尝试几个变更,观察变更的影响。首先,如果您将第一个 SetBkMode 呼叫变为注解,将得到由图案而不是字元本身所覆盖的文字方块背景。这通常不是我们实际所需要的,但确实可这样做。

此外,填充字元及将它们用做剪裁时,您可能想有效地放弃内定的 ALTERNATE 多边填充模式。我的经验表示:如果使用 WINDING 填充模式,则构建 TrueType 字体以避免出现奇怪的现象(例如「0」的内部被填充),但使用 ALTERNATE 模式更安全。



图 17-7 FONTFILL 的萤幕显示

最後，可使用一个绘图路径，因此也是一个 TrueType 字体，来定义剪裁区域。如程式 17-13 FONTCLIP 所示。

程式 17-13 FONTCLIP

```
FONTCLIP.C
/*-----
-
FONTCLIP.C --          Using Path for Clipping on Font
                        (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "..\eztest\ezfont.h"

TCHAR szAppName [] = TEXT ("FontClip") ;
TCHAR szTitle    [] = TEXT ("FontClip: Using Path for Clipping on Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    static TCHAR    szString [] = TEXT ("Clipping") ;
    HFONT          hFont ;
    int             y, iOffset ;
    POINT           pt [4] ;
    SIZE            size ;
```

```

hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1200, 0, 0, TRUE) ;
SelectObject (hdc, hFont) ;

GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
BeginPath (hdc) ;
TextOut (hdc, ( cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
                                                szString, lstrlen (szString)) ;

EndPath (hdc) ;

                                // Set clipping area
SelectClipPath (hdc, RGN_COPY) ;
                                // Draw Bezier splines
iOffset = (cxArea + cyArea) / 4 ;
for (y = -iOffset ; y < cyArea + iOffset ; y++)
{
    pt[0].x = 0 ;
    pt[0].y = y ;

    pt[1].x = cxArea / 3 ;
    pt[1].y = y + iOffset ;

    pt[2].x = 2 * cxArea / 3 ;
    pt[2].y = y - iOffset ;

    pt[3].x = cxArea ;
    pt[3].y = y ;

    SelectObject (hdc, CreatePen (PS_SOLID, 1,
                                RGB (rand () % 256, rand () % 256, rand () % 256))) ;
    PolyBezier (hdc, pt, 4) ;
    DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
DeleteObject (hFont) ;
}

```

程式中故意不使用 SetBkMode 呼叫以实作不同的效果。程式在绘图路径支架中绘制一些文字，然後呼叫 SelectClipPath。接著使用随机颜色绘制一系列贝塞尔曲线。

如果 FONTCLIP 程式使用 TRANSPARENT 选项呼叫 SetBkMode，贝塞尔曲线将被限制在字元轮廓的内部。在内定 OPAQUE 选项的背景模式下，剪裁区域被限制在文字方块内部而不是文字内部。如图 17-8 所示。

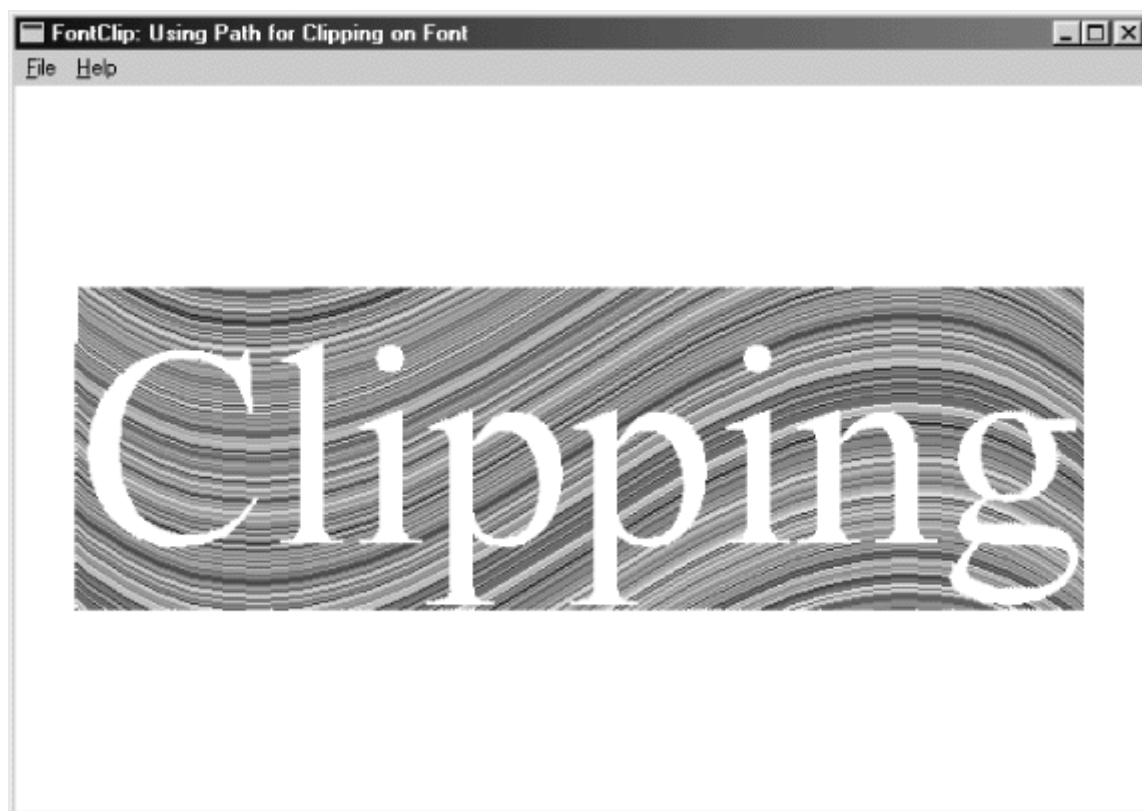


图 17-8 FONTCLIP 得萤幕显示

您或许会想在 FONTCLIP 中插入 SetBkMode 呼叫来观察 TRANSPARENT 选项的变化。

FONTDEMO 外壳程式允许您列印并显示这些效果，甚至允许您尝试自己的一些特殊效果。

第十八章 Metafile

Metafile 和向量图形的关系，就像点阵图和位元映射图形的关系一样。点阵图通常来自实际的图像，而 metafile 则大多是通过电脑程式人为建立的。Metafile 由一系列与图形函式呼叫相同的二进位记录组成，这些记录一般用於绘制直线、曲线、填入的区域和文字等。

「画图 (paint)」程式建立点阵图，而「绘图 (draw)」程式建立 metafile。在优秀的绘图程式中，能轻易地「抓住」某个独立的图形物件（例如一条直线）并将它移动到其他位置。这是因为组成图形的每个成员都是以单独的记录储存的。在画图程式中，这是不可能的——您通常都会局限於删除或插入点阵图矩形块。

由於 metafile 以图形绘制命令描述图像，因此可以对图像进行缩放而不会失真。点阵图则不然，如果以二倍大小来显示点阵图，您却无法得到二倍的解析度，而只是在水平和垂直方向上重复点阵图的位元。

Metafile 可以转换为点阵图，但是会丢失一些资讯：组成 metafile 的图形物件将不再是独立的，而是被合并进大的图像。将点阵图转换为 metafile 要艰难得多，一般仅限於非常简单的图像，而且它需要大量处理来分析边界和轮廓。而 metafile 可以包含绘制点阵图的命令。

虽然 metafile 可以作为图片剪辑储存在磁片上，但是它们大多用於程式通过剪贴簿共用图片的情况。由於 metafile 将图片描述为图像函式呼叫的集合，因而它们既比点阵图占用更少的空间，又比点阵图更与装置无关。

Microsoft Windows 支援两种 metafile 格式和支援这些格式的两组函式。我首先讨论从 Windows 1.0 到目前的 32 位元 Windows 版本都支援的 metafile 函式，然後讨论为 32 位元 Windows 系统开发的「增强型 metafile」。增强型 metafile 在原有 metafile 的基础上有了一些改进，应该尽可能地加以利用。

旧的 metafile 格式

Metafile 既能够暂时储存在记忆体中，也能够以档案的形式储存在磁片上。对应用程式来说，两者区别不大，尤其是由 Windows 来处理磁片上储存和载入 metafile 资料的档案 I/O 时，更是如此。

记忆体 metafile 的简单利用

如果呼叫 CreateMetaFile 函式来建立 metafile 装置内容，Windows 就会以

早期的格式建立一个 metafile，然後您可以使用大部分 GDI 绘图函式在该 metafile 装置内容上进行绘图。这些 GDI 呼叫并不在任何具体的装置上绘图，相反地，它们被储存在 metafile 中。当关闭 metafile 装置内容时，会得到 metafile 的代号。这时就可以在某个具体的装置内容上「播放」这个 metafile，这与直接执行 metafile 中 GDI 函式的效果等同。

CreateMetaFile 只有一个参数，它可以是 NULL 或档案名称。如果是 NULL，则 metafile 储存在记忆体中。如果是档案名称(以 .WMF 作为「Windows Metafile」的副档名)，则 metafile 储存在磁片档案中。

程式 18-1 中的 METAFILE 显示了在 WM_CREATE 讯息处理期间建立记忆体 metafile 的方法，并在 WM_PAINT 讯息处理期间将图像显示 100 遍。

程式 18-1 METAFILE

```

METAFILE.C
/*-----
    METAFILE.C --          Metafile Demonstration Program
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR szAppName [] = TEXT ("Metafile") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon      (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows
NT!"),

```

```

szAppName,
MB_ICONERROR) ;

    return 0 ;

}

hwnd = CreateWindow (  szAppName, TEXT ("Metafile Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
    static HMETAFILE      hmf ;
    static int             cxClient, cyClient ;
    static HBRUSH          hBrush ;
    static HDC             hdc, hdcMeta ;
    static int             x, y ;
    static PAINTSTRUCT      ps ;

    switch (message)
    {
    case WM_CREATE:
        hdcMeta      =      CreateMetaFile (NULL) ;
        hBrush       =      CreateSolidBrush (RGB (0, 0, 255)) ;
        Rectangle    (hdcMeta, 0, 0, 100, 100) ;

        MoveToEx     (hdcMeta, 0, 0, NULL) ;
        LineTo       (hdcMeta, 100, 100) ;
        MoveToEx     (hdcMeta, 0, 100, NULL) ;
        LineTo       (hdcMeta, 100, 0) ;

        SelectObject (hdcMeta, hBrush) ;
        Ellipse      (hdcMeta, 20, 20, 80, 80) ;

        hmf = CloseMetaFile (hdcMeta) ;
    }
}

```

```
        DeleteObject (hBrush) ;
        return 0 ;

case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        SetMapMode (hdc, MM_ANISOTROPIC) ;
        SetWindowExtEx (hdc, 1000, 1000, NULL) ;
        SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

        for (x = 0 ; x < 10 ; x++)
        for (y = 0 ; y < 10 ; y++)
        {
                SetWindowOrgEx (hdc, -100 * x, -100 * y, NULL) ;
                PlayMetaFile (hdc, hmf) ;
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY:
        DeleteMetaFile (hmf) ;
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个程式展示了在使用记忆体 metafile 时所涉及的 4 个 metafile 函式的用法。第一个是 CreateMetaFile。在 WM_CREATE 讯息处理期间用 NULL 参数呼叫该函式,并传回 metafile 装置内容的代号。然後, METAFILE 利用这个 metafileDC 来绘制两条直线和一个蓝色椭圆。这些函式呼叫以二进位形式储存在 metafile 中。CloseMetaFile 函式传回 metafile 的代号。因为以後还要用到该 metafile 代号,所以把它储存在静态变数。

该 metafile 包含 GDI 函式呼叫的二进位表示码,它们是两个 MoveToEx 呼叫、两个 LineTo 呼叫、一个 SelectObject 呼叫(指定蓝色画刷)和一个 Ellipse 呼叫。座标没有指定任何映射方式或转换,它们只是作为数值资料被储存在 metafile 中。

在 WM_PAINT 讯息处理期间, METAFILE 设定一种映射方式并呼叫

PlayMetaFile 在视窗中绘制物件 100 次。Metafile 中函式呼叫的座标按照目的装置内容的目前变换方式加以解释。在呼叫 PlayMetaFile 时，事实上是在重复地呼叫最初在 WM_CREATE 讯息处理期间建立 metafile 时，在 CreateMetaFile 和 CloseMetaFile 之间所做的所有呼叫。

和任何 GDI 物件一样，metafile 物件也应该在程式终止前被删除。这是在 WM_DESTROY 讯息处理期间用 DeleteMetaFile 函式处理的工作。

METAFILE 程式的结果如图 18-1 所示。

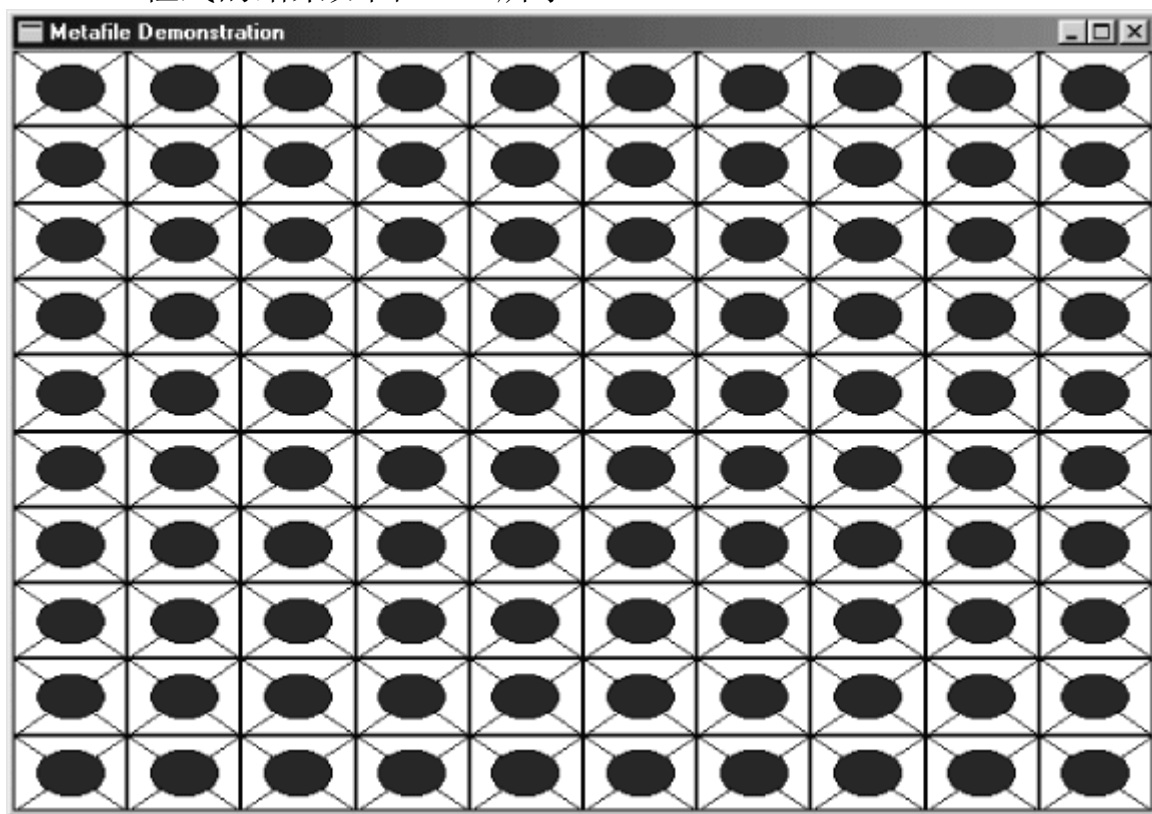


图 18-1 METAFILE 程式执行结果显示

将 metafile 储存在磁碟上

在上面的例子中，CreateMetaFile 的 NULL 参数表示要建立储存在记忆体中的 metafile。我们也可以建立作为档案储存在磁碟上的 metafile，这种方法对于大的 metafile 比较合适，因为可以节省记忆体空间。而另一方面，每次使用磁片上的 metafile 时，就需要存取磁片。

要把 METAFILE 转换为使用 metafile 磁片档案的程式，必须把 CreateMetaFile 的 NULL 参数替换为档案名称。在 WM_CREATE 处理结束时，可以用 metafile 代号来呼叫 DeleteMetaFile，这样代号被删除，但是磁片档案仍然被储存著。

在处理 WM_PAINT 讯息处理期间，可以通过呼叫 GetMetaFile 来取得此磁碟档案的 metafile 代号：


```
hmf = GetMetaFile (szFileName) ;
```

现在就可以像前面那样显示这个 metafile。在 WM_PAINT 讯息处理结束时，可以用下面的叙述删除该 metafile 代号：

```
DeleteMetaFile (hmf) ;
```

在开始处理 WM_DESTROY 讯息时，不必删除 metafile，因为它已经在 WM_CREATE 讯息和每个 WM_PAINT 讯息结束时被删除了，但是仍然需要删除磁碟档案：

```
DeleteFile (szFileName) ;
```

当然，除非您想储存该档案。

正如在第十章讨论过的，metafile 也可以作为使用者自订资源。您可以简单地把它当作资料块载入。如果您有一块包含 metafile 内容的资料，那么您可以使用

```
hmf = SetMetaFileBitsEx (iSize, pData) ;
```

来建立 metafile。SetMetaFileBitsEx 有一个对应的函式——GetMetaFileBitsEx，此函式将 metafile 的内容复制到记忆体块中。

老式 metafile 与剪贴簿

老式 metafile 有个讨厌的缺陷。如果您具有老式 metafile 的代号，那么，当您在显示 metafile 时如何确定它的大小呢？除非您深入分析 metafile 的内部结构，否则无法得知。

此外，当程式从剪贴簿取得老式 metafile 时，如果 metafile 被定义为在 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式下显示，则此程式在使用该 metafile 时具有最大程度的灵活性。程式收到该 metafile 後，就可以在显示它之前简单地通过设定视埠的范围来缩放图像。然而，如果 metafile 内的映射方式被设定为 MM_ISOTROPIC 或 MM_ANISOTROPIC，则收到该 metafile 的程式将无法继续执行。程式仅能在显示 metafile 之前或之後进行 GDI 呼叫，不允许在显示 metafile 当中进行 GDI 呼叫。

为了解决这些问题，老式 metafile 代号不直接放入剪贴簿供其他程式取得，而是作为「metafile 图片」（METAFILEPICT 结构型态）的一部分。此结构使得从剪贴簿上取得 metafile 图片的程式能够在显示 metafile 之前设定映射方式和视埠范围。

METAFILEPICT 结构的长度为 16 个位元组，定义如下：

```
typedef struct tagMETAFILEPICT
{
    LONG mm ;                // mapping mode
    LONG xExt ;              // width of the metafile image
    LONG yExt ;              // height of the metafile image
```

```

    LONG hMF ;                                // handle to the metafile
}
METAFILEPICT ;

```

對於 MM_ISOTROPIC 和 MM_ANISOTROPIC 以外的所有映射方式，图像大小用 xExt 和 yExt 值表示，其单位是由 mm 给出的映射方式的单位。利用这些资讯，从剪贴簿复制 metafile 图片结构的程式就能够确定在显示 metafile 时所需的显示空间。建立该 metafile 的程式可以将这些值设定为输入 metafile 的 GDI 绘制函式中所使用的最大的 x 座标和 y 座标值。

在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式下，xExt 和 yExt 栏位有不同的功能。我们在第五章中曾介绍过一个程式，该程式为了在 GDI 函式中使用与图像实际尺寸无关的逻辑单位而采用 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式。当程式只想保持纵横比而可以忽略图形显示平面的大小时，采用 MM_ISOTROPIC 模式；反之，当不需要考虑纵横比时采用 MM_ANISOTROPIC 模式。您也许还记得，第五章中在程式将映射方式设定为 MM_ISOTROPIC 或 MM_ANISOTROPIC 後，通常会呼叫 SetWindowExtEx 和 SetViewportExtEx。SetWindowExtEx 呼叫使用逻辑单位来指定程式在绘制时使用的单位，而 SetViewportExtEx 呼叫使用的装置单位大小则取决於图形显示平面（例如，视窗显示区域的大小）。

如果程式为剪贴簿建立了 MM_ISOTROPIC 或 MM_ANISOTROPIC 方式的 metafile，则该 metafile 本身不应包含对 SetViewportExtEx 的呼叫，因为该呼叫中的装置单位应该依据建立 metafile 的程式的显示平面，而不是依据从剪贴簿读取并显示 metafile 的程式的显示平面。从剪贴簿取得 metafile 的程式可以利用 xExt 和 yExt 值来设定合适的视埠范围以便显示 metafile。但是当映射方式是 MM_ISOTROPIC 或 MM_ANISOTROPIC 时，metafile 本身包含设定视窗范围的呼叫。Metafile 内的 GDI 绘图函式的座标依据这些视窗的范围。

建立 metafile 和 metafile 图片遵循以下规则：

- 设定 METAFILEPICT 结构的 mm 栏位来指定映射方式。
- 對於 MM_ISOTROPIC 和 MM_ANISOTROPIC 以外的映射方式，xExt 与 yExt 栏位设定为图像的宽和高，单位与 mm 栏位相对应。對於在 MM_ISOTROPIC 或 MM_ANISOTROPIC 方式下显示的 metafile，工作要复杂一些。在 MM_ANISOTROPIC 模式下，当程式既不对图片大小跟纵横比给出任何建议资讯时，xExt 和 yExt 的值均为零。在这两种模式下，如果 xExt 和 yExt 的值为正数，它们就是以 0.01mm 单位（MM_HIMETRIC 单位）表示该图像的宽度和高度。在 MM_ISOTROPIC 方式下，如果 xExt 和 yExt 为负值，它们就指出了图像的纵横比而不是大小。

- 在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式下, metafile 本身含有对 SetWindowExtEx 的呼叫, 也可能有对 SetWindowOrgEx 的呼叫。亦即, 建立 metafile 的程式在 metafile 装置内容中呼叫这些函式。Metafile 一般不会包含对 SetMapMode、SetViewportExtEx 或 SetViewportOrgEx 的呼叫。
- metafile 应该是记忆体 metafile, 而不是 metafile 档案。

这里有一段范例程式码, 它建立 metafile 并将其复制到剪贴簿。如果 metafile 使用 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式, 则该 metafile 的第一个呼叫应该设定视窗范围 (在其他模式中, 视窗的大小是固定的)。无论在何种模式下, 视窗的位置应如下设定:

```
hdcMeta = CreateMetaFile (NULL) ;
SetWindowExtEx (hdcMeta, ...) ;
SetWindowOrgEx (hdcMeta, ...) ;
```

Metafile 绘图函式中的座标决定於这些视窗范围和视窗原点。当程式使用 GDI 呼叫在 metafile 装置内容中绘制完成後, 关闭 metafile 以得到 metafile 代号:

```
hmf = CloseMetaFile (hdcMeta) ;
```

该程式还需要定义指向 METAFILEPICT 型态结构的指标, 并为此结构配置一块整体记忆体:

```
GLOBALHANDLE          hGlobal ;
LPMETAFILEPICT        pMFP ;
其他行程式
hGlobal= GlobalAlloc (GHND | GMEM_SHARE, sizeof (METAFILEPICT)) ;
pMFP = (LPMETAFILEPICT) GlobalLock (hGlobal) ;
接著, 程式设定该结构的 4 个栏位:
pMFP->mm      = MM_... ;
pMFP->xExt     = ... ;
pMFP->yExt     = ... ;
pMFP->hMF      = hmf ;

GlobalUnlock (hGlobal) ;
```

然後, 程式将包含有 metafile 图片的整体记忆体块传送给剪贴簿:

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (CF_METAFILEPICT, hGlobal) ;
CloseClipboard () ;
```

完成这些呼叫後, hGlobal 代号 (包含 metafile 图片结构的记忆体块) 和 hmf 代号 (metafile 本身) 就对建立它们的程式失效了。

现在来看一看难的部分。当程式从剪贴簿取得 metafile 并显示它时, 必须完成下列步骤:

1. 程式利用 metafile 图片结构的 mm 栏位设定映射方式。
2. 对於 MM_ISOTROPIC 或 MM_ANISOTROPIC 以外的映射方式, 程式用 xExt 和 yExt 值设定剪贴矩形或简单地设定图像大小。而在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式, 程式使用 xExt 和 yExt 来设定视埠范围。
3. 然後, 程式显示 metafile。

下面程式码, 首先打开剪贴簿, 得到 metafile 图片结构代号并将其锁定:

```
OpenClipboard (hwnd) ;
hGlobal = GetClipboardData (CF_METAFILEPICT) ;
pMFP = (LPMETAFILEPICT) GlobalLock (hGlobal) ;
```

现在可以储存目前装置内容的属性, 并将映射方式设定为结构中的 mm 值:

```
SaveDC (hdc) ;
SetMappingMode (pMFP->mm) ;
```

如果映射方式不是 MM_ISOTROPIC 或 MM_ANISOTROPIC, 则可以用 xExt 和 yExt 的值设定剪贴矩形。由於这两个值是逻辑单位, 必须用 LPtoDP 将其转换为用於剪贴矩形的装置单位的座标。也可以简单地储存这些值以掌握图像的大小。

对於 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式, xExt 和 yExt 用来设定视埠范围。下面有一个用来完成此项任务的函式, 如果 xExt 和 yExt 没有建议的大小, 则该函式假定 cxClient 和 cyClient 分别表示 metafile 显示区域的图素高度和宽度。

```
void PrepareMetaFile ( HDC hdc, LPMETAFILEPICT pmfp,
                      int cxClient, int cyClient)
{
    int xScale, yScale, iScale ;
    SetMapMode (hdc, pmfp->mm) ;
    if (pmfp->mm == MM_ISOTROPIC || pmfp->mm == MM_ANISOTROPIC)
    {
        if (pmfp->xExt == 0)
            SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
        else if (pmfp->xExt > 0)
            SetViewportExtEx (hdc,
                              pmfp->xExt * GetDeviceCaps (hdc, HORZRES) /
                              GetDeviceCaps (hdc, HORZSIZE) / 100,
                              pmfp->yExt * GetDeviceCaps (hdc, VERTRES) /
                              GetDeviceCaps (hdc, VERTSIZE) / 100, NULL) ;
        else if (pmfp->xExt < 0)
        {
            xScale = 100 * cxClient * GetDeviceCaps (hdc, HORZSIZE) /
                      GetDeviceCaps (hdc, HORZRES) / -pmfp->xExt ;
            yScale = 100 * cyClient * GetDeviceCaps (hdc, VERTSIZE) /
                      GetDeviceCaps (hdc, VERTRES) / -pmfp->yExt ;
            iScale = min (xScale, yScale) ;
        }
    }
}
```

```

    SetViewportExtEx (hdc, -pmfp->xExt * iScale * GetDeviceCaps (hdc, HORZRES)
/
    GetDeviceCaps (hdc, HORZSIZE) / 100, -pmfp->yExt * iScale
* GetDeviceCaps (hdc, VERTRES) / GetDeviceCaps (hdc, VERTSIZE) / 100, NULL) ;
    }
}
}

```

上面的程式码假设 xExt 和 yExt 同时都为零、大於零或小於零，这三种状态之一。如果范围为零，表示没有建议大小或纵横比，视埠范围设定为显示 metafile 的区域。如果大於零，则 xExt 和 yExt 的值代表图像的建议大小，单位是 0.01mm。GetDeviceCaps 函式用来确定每 0.01mm 中包含的图素数，并且该值与 metafile 图片结构的范围值相乘。如果小於零，则 xExt 和 yExt 的值表示建议的纵横比而不是建议的大小。iScale 的值首先根据对应 cxClient 和 cyClient 的毫米表示的纵横比计算出来，该缩放因数用於设定图素单位的视埠范围。

完成了上述工作後，可以设定视埠原点，显示 metafile，并恢复装置内容：

```

PlayMetaFile (pMFP->hMF) ;
RestoreDC (hdc, -1) ;

```

然後，对记忆体块解锁并关闭剪贴簿：

```

GlobalUnlock (hGlobal) ;
CloseClipboard () ;

```

如果程式使用增强型 metafile 就可以省去这项工作。当某个应用程式将这些格式放入剪贴簿而另一个程式却要求从剪贴簿中获得其他格式时，Windows 剪贴簿会自动在老式 metafile 和增强型 metafile 之间进行格式转换。

增强型 metafile

「增强型 metafile」格式是在 32 位元 Windows 版本中发表的。它包含一组新的函式呼叫、一对新的资料结构、新的剪贴簿格式和新的档案副档名. EMF。

这种新的 metafile 格式最重要的改进是加入可通过函式呼叫取得的更丰富的表头资讯，这种表头资讯可用来帮助应用程式显示 metafile 图像。

有些增强型 metafile 函式使您能够在增强型 metafile(EMF)格式和老式 metafile 格式（也称作 Windows metafile(WMF)格式）之间来回转换。当然，这种转换很可能遇到麻烦，因为老式 metafile 格式并不支援某些，例如 GDI 绘图路径等，新的 32 位元图形功能。

基本程序

程式 18-2 所示的 EMF1 建立并显示增强型 metafile。

程式 18-2 EMF1

```

EMF1.C
/*-----
--
    EMF1.C --    Enhanced Metafile Demo #1
                    (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int
nCmdShow)
{
    static TCHAR        szAppName[] = TEXT ("EMF1") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #1"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))

```

```

    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    static HENHMETAFILE    hemf ;
    HDC                    hdc, hdcEMF ;
    PAINTSTRUCT            ps ;
    RECT                    rect ;

    switch (message)
    {
    case WM_CREATE:
        hdcEMF = CreateEnhMetaFile (NULL, NULL, NULL, NULL) ;

        Rectangle    (hdcEMF, 100, 100, 200, 200) ;

        MoveToEx      (hdcEMF, 100, 100, NULL) ;
        LineTo        (hdcEMF, 200, 200) ;

        MoveToEx      (hdcEMF, 200, 100, NULL) ;
        LineTo        (hdcEMF, 100, 200) ;

        hemf = CloseEnhMetaFile (hdcEMF) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;

        rect.left      = rect.right / 4 ;
        rect.right     = 3 * rect.right / 4 ;
        rect.top       = rect.bottom / 4 ;
        rect.bottom    = 3 * rect.bottom / 4 ;

        PlayEnhMetaFile (hdc, hemf, &rect) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        DeleteEnhMetaFile (hemf) ;

```

```
        PostQuitMessage (0) ;  
        return 0 ;  
    }  
    return DefWindowProc (hwnd, message, wParam, lParam) ;  
}
```

在 EMF1 的视窗讯息处理程式处理 WM_CREATE 讯息处理期间, 程式首先通过呼叫 CreateEnhMetaFile 来建立增强型 metafile。该函式有 4 个参数, 但可以把它们都设为 NULL。稍候我将说明这 4 个参数在非 NULL 情况下的使用方法。

和 CreateMetaFile 一样, CreateEnhMetaFile 函式传回特定的装置内容代号。该程式利用这个代号绘制一个矩形和该矩形的两条对角线。这些函式呼叫及其参数被转换为二进位元的形式并储存在 metafile 中。

最後通过对 CloseEnhMetaFile 函式的呼叫结束了增强型 metafile 的建立并传回指向它的代号。该档案代号储存在 HENHMETAFILE 型态的静态变数中。

在 WM_PAINT 讯息处理期间, EMF1 以 RECT 结构取得程式的显示区域视窗大小。通过调整结构中的 4 个栏位, 使该矩形的长和宽为显示区域视窗长和宽的一半并位於视窗的中央。然後 EMF1 呼叫 PlayEnhMetaFile, 该函式的第一个参数是视窗的装置内容代号, 第二个参数是该增强型 metafile 的代号, 第三个参数是指向 RECT 结构的指标。

在 metafile 的建立程序中, GDI 得出整个 metafile 图像的尺寸。在本例中, 图像的长和宽均为 100 个单位。在 metafile 的显示程序中, GDI 将图像拉伸以适应 PlayEnhMetaFile 函式指定的矩形大小。EMF1 在 Windows 下执行的三个执行实体如图 18-2 所示。

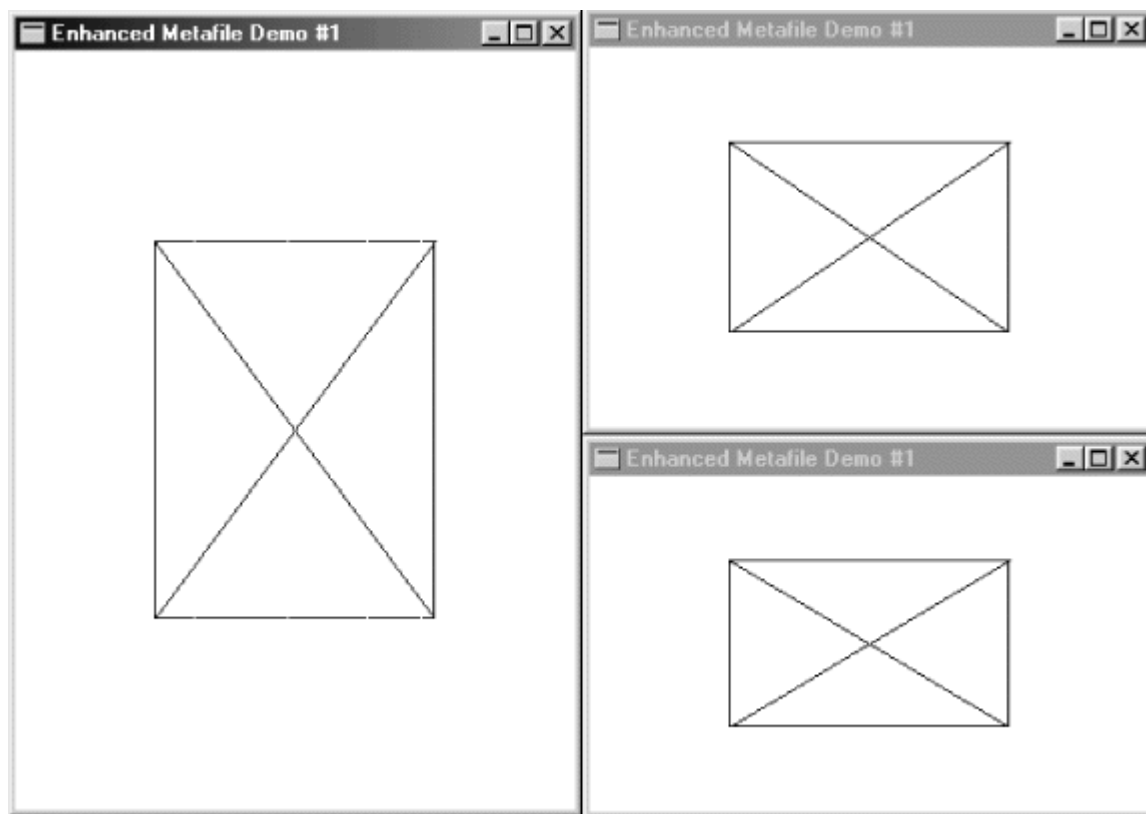


图 18-2 EMF1 得萤幕显示

最後，在 WM_DESTROY 讯息处理期间，EMF1 呼叫 DeleteEnhMetaFile 删除 metafile。

让我们总结一下从 EMF1 程式学到的一些东西。

首先，该程式在建立增强型 metafile 时，画矩形和直线的函式所使用的座标并不是实际意义上的座标。您可以将它们同时加倍或都减去某个常数，而其结果不会改变。这些座标只是在定义图像时说明彼此间的对应关系。

其次，为了适於在传递给 PlayEnhMetaFile 函式的矩形中显示，图像大小会被缩放。因此，如图 18-2 所示，图像可能会变形。尽管 metafile 座标指出该图像是正方形的，但一般情况下我们却得不到这样的图像。而在某些时候，这又正是我们想要得到的图像。例如，将图像嵌入一段文书处理格式的文字中时，可能会要求使用者为图像指定矩形，并且确保整个图像恰好位於矩形中而不浪费空间。这样，使用者可通过适当调整矩形的大小来得到正确的纵横比。

然而有时候，您也许希望保留图像最初的纵横比，因为这一点對於表现视觉资讯尤为重要。例如，警察的嫌疑犯草图既不能比原型胖也不能比原型瘦。或者您希望保留原来图像的度量尺寸，图像必须是两英寸高，否则就不能正常显示。在这种情况下，保留图像的原来尺寸就非常重要了。

同时也要注意 metafile 中画出的那些对角线似乎没有与矩形顶点相交。这是由於 Windows 在 metafile 中储存矩形座标的方式造成的。稍後，会说明解决这个问题方法。

揭开内幕

如果看一看 metafile 的内容会对 metafile 工作的方式有一个更好的理解。如果您有一个 metafile 档案, 这将很容易做到, 程式 18-3 中的 EMF2 程式建立了一个 metafile。

程式 18-3 EMF2

```
EMF2.C
/*-----
---
      EMF2.C --  Enhanced Metafile Demo #2
                        (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int
nCmdShow)
{
    static TCHAR          szAppName[] = TEXT ("EMF2") ;
    HWND                  hwnd ;
    MSG                   msg ;
    WNDCLASS               wndclass ;
    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #2"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
```

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

ShowWindow (hwnd, nCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    HDC                hdc, hdcEMF ;
    HENHMETAFILE       hemf ;
    PAINTSTRUCT        ps ;
    RECT               rect ;

    switch (message)
    {
    case WM_CREATE:
        hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf2.emf"), NULL,
        TEXT ("EMF2\0EMF Demo #2\0")) ;

        if (!hdcEMF)
            return 0 ;

        Rectangle (hdcEMF, 100, 100, 200, 200) ;

        MoveToEx (hdcEMF, 100, 100, NULL) ;
        LineTo (hdcEMF, 200, 200) ;

        MoveToEx (hdcEMF, 200, 100, NULL) ;
        LineTo (hdcEMF, 100, 200) ;

        hemf = CloseEnhMetaFile (hdcEMF) ;

        DeleteEnhMetaFile (hemf) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;
```

```

        rect.left      =      rect.right      / 4 ;
        rect.right     = 3 *      rect.right     / 4 ;
        rect.top      =      rect.bottom      / 4 ;
        rect.bottom    = 3 * rect.bottom      / 4 ;

        if (hemf = GetEnhMetaFile (TEXT ("emf2.emf")))
        {
                PlayEnhMetaFile (hdc, hemf, &rect) ;
                DeleteEnhMetaFile (hemf) ;
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

在 EMF1 程式中, CreateEnhMetaFile 函式的所有参数均被设定为 NULL。在 EMF2 中, 第一个参数仍旧设定为 NULL, 该参数还可以是装置内容代号。GDI 使用该参数在 metafile 表头中插入度量资讯, 很快我会讨论它。如果该参数为 NULL, 则 GDI 认为度量资讯是由视讯装置内容决定的。

CreateEnhMetaFile 函式的第二个参数是档案名称。如果该参数为 NULL (在 EMF1 中为 NULL, 但在 EMF2 中不为 NULL), 则该函式建立记忆体 metafile。EMF2 建立名为 EMF2.EMF 的 metafile 档案。

函式的第三个参数是 RECT 结构的位址, 它指出了以 0.01mm 为单位的 metafile 的总大小。这是 metafile 表头资料中极其重要的资讯 (这是早期的 Windows metafile 格式的缺陷之一)。如果该参数为 NULL, GDI 会计算出尺寸。我比较喜欢让作业系统替我做这些事, 所以将该参数设定为 NULL。当应用程式对性能要求比较严格时, 就需要使用该参数以避免让 GDI 处理太多东西。

最後的参数是描述该 metafile 的字串。该字串分为两部分: 第一部分是以 NULL 字元结尾的应用程式名称 (不一定是程式的档案名称), 第二部分是描述视觉图像内容的说明, 以两个 NULL 字元结尾。例如用 C 中的符号「\0」作为 NULL 字元, 则该描述字串可以是「LoonyCad V6.4\0Flying Frogs\0\0」。由於在 C 中通常会在使用的字串末尾放入一个 NULL 字元, 所以如 EMF2 所示, 在末尾仅需一个「\0」。

建立完 metafile 後, 与 EMF1 一样, EMF2 也透过利用由 CreateEnhMetaFile 函式传回的装置内容代号进行一些 GDI 函式呼叫。然後程式呼叫 CloseEnhMetaFile 删除装置内容代号并取得完成的 metafile 的代号。

然後，在 WM_CREATE 讯息还没处理完毕时，EMF2 做了一些 EMF1 没有做的事情：在获得 metafile 代号之後，程式呼叫 DeleteEnhMetaFile。该操作释放了用於储存 metafile 的所有记忆体资源。然而，metafile 档案仍然保留在磁碟机中（如果愿意，您可以使用如 DeleteFile 的档案删除函式来删除该档案）。注意 metafile 代号并不像 EMF1 中那样储存在静态变数中，这意味著在讯息之间不需要储存它。

现在，为了使用该 metafile，EMF2 需要存取磁片档案。这是在 WM_PAINT 讯息处理期间透过呼叫 GetEnhMetaFile 进行的。Metafile 的档案名称是该函式的唯一参数，该函式传回 metafile 代号。和 EMF1 一样，EMF2 将这个档案代号传递给 PlayEnhMetaFile 函式。该 metafile 图像在 PlayEnhMetaFile 函式的最後一个参数所指定的矩形中显示。与 EMF1 不同的是，EMF2 在 WM_PAINT 讯息结束之前就删除该 metafile。此後每次处理 WM_PAINT 讯息时，EMF2 都会再次读取 metafile，显示并删除它。

要记住，对 metafile 的删除操作仅是释放了用以储存 metafile 的记忆体资源而已，磁片 metafile 甚至在程式执行结束後还保留在磁片上。

由於 EMF2 留下了 metafile 档案，您可以看一看它的内容。图 18-3 显示了该程式建立的 EMF2.EMF 档案的一堆十六进位代码。

0000	01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00d...d...
0010	C8 00 00 00 C8 00 00 00 35 0C 00 00 35 0C 00 005...5...
0020	6A 18 00 00 6A 18 00 00 20 45 4D 46 00 00 01 00	j...j...EMF....
0030	F4 00 00 00 07 00 00 00 01 00 00 00 12 00 00 00
0040	64 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00	d.....
0050	40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00	@.....
0060	00 00 00 00 45 00 4D 00 46 00 32 00 00 00 45 00	...E.M.F.2...E.
0070	4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00	M.F..D.e.m.o..
0080	23 00 32 00 00 00 00 00 2B 00 00 00 18 00 00 00	#.2.....+.....
0090	63 00 00 00 63 00 00 00 C6 00 00 00 C6 00 00 00	c...c.....
00A0	1B 00 00 00 10 00 00 00 64 00 00 00 64 00 00 00d...d...
00B0	36 00 00 00 10 00 00 00 C8 00 00 00 C8 00 00 00	6.....
00C0	1B 00 00 00 10 00 00 00 C8 00 00 00 64 00 00 00d...
00D0	36 00 00 00 10 00 00 00 64 00 00 00 C8 00 00 00	6.....d.....
00E0	0E 00 00 00 14 00 00 00 00 00 00 00 10 00 00 00
00F0	14 00 00 00

图 18-3 EMF2.EMF 的十六进位代码

图 18-3 所示的 metafile 是 EMF2 在 Microsoft Windows NT 4 下，视讯显示器的解析度为 1024 768 时建立的。同一程式在 Windows 98 下建立的 metafile 会比前者少 12 个位元组，这一点将在稍後讨论。同样地，视讯显示器的解析度也影响 metafile 表头的某些资讯。

增强型 metafile 格式使我们对 metafile 的工作方式有更深刻的理解。增

强型 metafile 由可变长度的记录组成，这些记录的一般格式由 ENHMETARECORD 结构说明，它在 WINGDI.H 表头档案中定义如下：

```
typedef struct tagENHMETARECORD
{
    DWORD iType ;                // record type
    DWORD nSize ;                // record size
    DWORD dParm [1] ;           // parameters
}
ENHMETARECORD ;
```

当然，那个只有一个元素的阵列指出了阵列元素的变数。参数的数量取决于记录型态。iType 栏位可以是定义在 WINGDI.H 档案中以字首 EMR_ 开始的近百个常数之一。nSize 栏位是总记录的大小，包括 iType 和 nSize 栏位以及一个或多个 dParm 栏位。

有了这些知识后，让我们看一下图 18-3。第一个栏位型态为 0x00000001，大小为 0x00000088，所以它占据档案的前 136 个位元组。记录型态为 1 表示常数 EMR_HEADER。我们不妨把对表头纪录的讨论往后搁，先跳到位于第一个记录末尾的偏移量 0x0088 处。

后面的 5 个记录与 EMF2 建立 metafile 之后的 5 个 GDI 函式呼叫有关。该记录在偏移量 0x0088 处有一个值为 0x0000002B 的型态代码，这代表 EMR_RECTANGLE，很明显是用于 Rectangle 呼叫的 metafile 记录。它的长度为 0x00000018（十进位 24）位元组，用以容纳 4 个 32 位元参数。实际上 Rectangle 函式有 5 个参数，但是第一个参数，也就是装置内容代号并未储存在 metafile 中，因为它没有实际意义。尽管在 EMF2 的函式呼叫中指定了矩形的顶点座标分别是 (100, 100) 和 (200, 200)，但 4 个参数中的 2 个是 0x00000063 (99)，另外 2 个是 0x000000C6 (198)。EMF2 程式在 Windows 98 下建立的 metafile 显示出前两个参数是 0x00000064 (100)，后 2 个参数是 0x000000C7 (199)。显然，在 Rectangle 参数储存到 metafile 之前，Windows 对它们作了调整，但没有保持一致。这就是对角线端点与矩形顶点不能重合的原因。

其次，有 4 个 16 位元记录与 2 个 MoveToEx (0x0000001B 或 EMR_MOVETOEX) 和 LineTo (0x00000036 或 EMR_LINETO) 呼叫有关。位于 metafile 中的参数与传递给函式的参数相同。

Metafile 以 20 个位元组长的型态代码为 0x0000000E 或 EMR_EOF（「end of file」）的记录结尾。

增强型 metafile 总是以表头纪录开始。它对应于 ENHMETAHEADER 型态的结构，定义如下：

```
typedef struct tagENHMETAHEADER
{
```

```

    DWORD iType ;          // EMR_HEADER = 1
    DWORD nSize ;          // structure size
    RECTL rclBounds ;      // bounding rectangle in pixels
    RECTL rclFrame ;       // size of image in 0.01 millimeters
    DWORD dSignature ;     // ENHMETA_SIGNATURE = " EMF"
    DWORD nVersion ;       // 0x00010000
    DWORD nBytes ;         // file size in bytes
    DWORD nRecords ;       // total number of records
    WORD nHandles ;        // number of handles in handle table
    WORD sReserved ;
    DWORD nDescription ;    // character length of description string
    DWORD offDescription ;  // offset of description string in file
    DWORD nPalEntries ;     // number of entries in palette
    SIZEL szlDevice ;       // device resolution in pixels
    SIZEL szlMillimeters ;  // device resolution in millimeters
    DWORD cbPixelFormat ;   // size of pixel format
    DWORD offPixelFormat ;  // offset of pixel format
    DWORD bOpenGL ;        // FALSE if no OpenGL records
}
ENHMETAHEADER ;

```

这种表头纪录的存在可能是增强型 metafile 格式对早期 Windows metafile 所做的最为重要的改进。不需要对 metafile 档案使用档案 I/O 函式来取得这些表头资讯。如果具有 metafile 代号, 就可以使用 GetEnhMetaFileHeader 函式:

```
GetEnhMetaFileHeader (hemf, cbSize, &emh) ;
```

第一个参数是 metafile 代号。最後一个参数是指向 ENHMETAHEADER 结构的指标。第二个参数是该结构的大小。可以使用类似的 GetEnh-MetaFileDescription 函式取得描述字串。

如上面所定义的, ENHMETAHEADER 结构有 100 位元组长, 但在 MF2.EMFmetafile 中, 记录的大小包括描述字串, 所以大小为 0x88, 即 136 位元组。而 Windows 98metafile 的表头纪录不包含 ENHMETAHEADER 结构的最後 3 个栏位, 这一点解释了 12 个位元组的差别。

rclBounds 栏位是指出图像大小的 RECT 结构, 单位是图素。将其从十六进位转换过来, 我们看到该图像正如我们希望的那样, 其左上角位於(100, 100), 右下角位於(200, 200)。

rclFrame 栏位是提供相同资讯的另一个矩形结构, 但它是以 0.01 毫米为单位。在这种情况下, 该档案显示两对角顶点分别位於(0x0C35, 0x0C35)和(0x186A, 0x186A), 用十进位表示为(3125, 3125)和(6250, 6250)的矩形。这些数字是怎么来的? 我们很快就会明白。

dSignature 栏位始终为值 ENHMETA_SIGNATURE 或 0x464D4520。这看上去是一个奇怪的数字, 但如果将位元组的排列顺序倒过来 (就像 Intel 处理器在记

忆体中储存多位元组数那样)并转换成 ASCII 码,就变成字串"EMF"。dVersion 栏位的值始终是 0x00010000。

其後是 nBytes 栏位,该栏位在本例中是 0x000000F4,这是该 metafile 的总位元组数。nRecords 栏位(在本例中是 0x00000007)指出了记录数——包括表头纪录、5 个 GDI 函式呼叫和档案结束记录。

下面是两个十六位元的栏位。nHandles 栏位为 0x0001。该栏位一般指出 metafile 所使用的图形物件(如画笔、画刷和字体)的非内定代号的数量。由於没有使用这些图形物件,您可能会认为该栏位为零,但实际上 GDI 自己保留了第一个栏位。我们将很快见到代号储存在 metafile 中的方式。

下两个栏位指出描述字串的字元个数,以及描述字串在档案中的偏移量,这里它们分别为 0x00000012(十进位数字 18)和 0x00000064。如果 metafile 没有描述字串,则这两个栏位均为零。

nPalEntries 栏位指出在 metafile 的调色盘表中条目的个数,本例中没有这种情况。

接著表头纪录包括两个 SIZEL 结构,它们包含两个 32 位栏位, cx 和 cy。szlDevice 栏位(在 metafile 中的偏移量为 0x0040)指出了以图素为单位的输出设备大小, szlMillimeters 栏位(偏移量为 0x0050)指出了以毫米为单位的输出设备大小。在增强型 metafile 文件中,这个输出设备被称作「参考设备(reference device)」。它是依据作为第一个参数传递给 CreateEnhMetaFile 呼叫的代号所指出的装置内容。如果该参数设为 NULL,则 GDI 使用视讯显示器。当 EMF2 建立上面所示的 metafile 时,正巧是在 Windows NT 上以 1024 768 显示模式工作,因此这就是 GDI 使用的参考设备。

GDI 通过呼叫 GetDeviceCaps 取得此资讯。EMF2.EMF 中的 szlDevice 栏位是 0x0400 0x0300(即 1024 768),它是以 HORZRES 和 VERTRES 作为参数呼叫 GetDeviceCaps 得到的。szlMillimeters 栏位是 0x140 0xF0,或 320 240,是以 HORZSIZE 和 VERTSIZE 作为参数呼叫 GetDeviceCaps 得到的。

通过简单的除法就可以得出图素为 0.3125mm 高和 0.3125mm 宽,这就是前面描述的 GDI 计算 rc1Frame 矩形尺寸的方法。

在 metafile 中, ENHMETAHEADER 结构後跟一个描述字串,该字串是 CreateEnhMetaFile 函式的最後一个参数。在本例中,该字串由後跟一个 NULL 字元的「EMF2」字串和後跟两个 NULL 字元的「EMF Demo #2」字串组成。总共 18 个字元,要是以 Unicode 方式储存则为 36 个字元。无论建立 metafile 的程式执行在 Windows NT 还是 Windows 98 下,该字串始终以 Unicode 方式储存。

metafile 与 GDI 物件

我们已经知道了 GDI 绘图命令储存在 metafile 中方式，现在看一下 GDI 物件的储存方式。程式 18-4 EMF3 除了建立用於绘制矩形和直线的非内定画笔和画刷以外，与前面介绍的 EMF2 程式很相似。该程式也对 Rectangle 座标的问题提出了一点修改。EMF3 程式使用 GetVersion 来确定执行环境是 Windows 98 还是 Windows NT，并适当地调整参数。

程式 18-4 EMF3

```
EMF3.C
/*-----
--
    EMF3.C --    Enhanced Metafile Demo #3
                        (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("EMF3") ;
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon      (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }
}
```

```

    hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #3"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    LOGBRUSH          lb ;
    HDC                hdc, hdcEMF ;
    HENHMETAFILE       hemf ;
    PAINTSTRUCT        ps ;
    RECT               rect ;
    switch (message)
    {
    case  WM_CREATE:
        hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf3.emf"), NULL,
        TEXT ("EMF3\0EMF Demo #3\0")) ;

        SelectObject (hdcEMF, CreateSolidBrush (RGB (0, 0, 255))) ;

        lb.lbStyle = BS_SOLID ;
        lb.lbColor = RGB (255, 0, 0) ;
        lb.lbHatch = 0 ;

        SelectObject (hdcEMF,
        ExtCreatePen (PS_SOLID | PS_GEOMETRIC, 5, &lb, 0, NULL)) ;

        if (GetVersion () & 0x80000000) // Windows 98
            Rectangle (hdcEMF, 100, 100, 201, 201) ;
        else
            // Windows NT
            Rectangle (hdcEMF, 101, 101, 202, 202) ;

        MoveToEx          (hdcEMF, 100, 100, NULL) ;

```

```

        LineTo          (hdcEMF, 200, 200) ;

        MoveToEx        (hdcEMF, 200, 100, NULL) ;
        LineTo          (hdcEMF, 100, 200) ;

        DeleteObject    (SelectObject (hdcEMF, GetStockObject
(BLACK_PEN))) ;
        DeleteObject    (SelectObject (hdcEMF, GetStockObject
(WHITE_BRUSH))) ;

        hemf = CloseEnhMetaFile (hdcEMF) ;

        DeleteEnhMetaFile (hemf) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;

        rect.left  =  rect.right          / 4 ;
        rect.right = 3 * rect.right        / 4 ;
        rect.top   =  rect.bottom         / 4 ;
        rect.bottom = 3 * rect.bottom      / 4 ;

        hemf = GetEnhMetaFile (TEXT ("emf3.emf")) ;

        PlayEnhMetaFile (hdc, hemf, &rect) ;
        DeleteEnhMetaFile (hemf) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

如我们所看到的，当利用 CreateEnhMetaFile 传回的装置内容代号来呼叫 GDI 函式时，这些函式呼叫被储存在 metafile 中而不是直接输出到萤幕或印表机上。然而，一些 GDI 函式根本不涉及特定的装置内容。其中有关建立画笔和画刷等图形物件的 GDI 函式十分重要。虽然逻辑画笔和画刷的定义储存在由 GDI 保留的记忆体中，但是在建立这些物件时，这些抽象的定义并未与任何特定的装置内容相关。

EMF3 呼叫 CreateSolidBrush 和 ExtCreatePen 函式。因为这些函式不需要

装置内容代号, 所以 GDI 不会把这些呼叫储存在 metafile 里。当呼叫它们时, GDI 函式只是简单地建立图形绘制物件而不会影响 metafile。

然而, 当程式呼叫 SelectObject 函式将 GDI 物件选入 metafile 装置内容时, GDI 既为物件建立函式编码 (源自用於储存物件的内部 GDI 资料) 也为 metafile 中的 SelectObject 呼叫进行编码。为了解其工作方式, 我们来看一下 EMF3.EMF 档案的十六进位代码, 如图 18-4 所示:

```

0000  01 00 00 00 88 00 00 00 60 00 00 00 60 00 00 00  .....`...`...
0010  CC 00 00 00 CC 00 00 00 B8 0B 00 00 B8 0B 00 00  .....
0020  E7 18 00 00 E7 18 00 00 20 45 4D 46 00 00 01 00  .....EMF.....
0030  88 01 00 00 0F 00 00 00 03 00 00 00 12 00 00 00  .....
0040  64 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00  d.....
0050  40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00  @.....
0060  00 00 00 00 45 00 4D 00 46 00 33 00 00 00 45 00  ....E.M.F.3...E.
0070  4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00  M.F....D.e.m.o..
0080  23 00 33 00 00 00 00 00 27 00 00 00 18 00 00 00  #.3.....'.....
0090  01 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00  .....
00A0  25 00 00 00 0C 00 00 00 01 00 00 00 5F 00 00 00  %....._...
00B0  34 00 00 00 02 00 00 00 34 00 00 00 00 00 00 00  4.....4.....
00C0  34 00 00 00 00 00 00 00 00 00 01 00 05 00 00 00  4.....
00D0  00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00  .....
00E0  25 00 00 00 0C 00 00 00 02 00 00 00 2B 00 00 00  %.....+...
00F0  18 00 00 00 63 00 00 00 63 00 00 00 C6 00 00 00  ....c...c.....
0100  C6 00 00 00 1B 00 00 00 10 00 00 00 64 00 00 00  .....d...
0110  64 00 00 00 36 00 00 00 10 00 00 00 C8 00 00 00  d...6.....
0120  C8 00 00 00 1B 00 00 00 10 00 00 00 C8 00 00 00  .....
0130  64 00 00 00 36 00 00 00 10 00 00 00 64 00 00 00  d...6.....d...
0140  C8 00 00 00 25 00 00 00 0C 00 00 00 07 00 00 80  ....%.....
0150  28 00 00 00 0C 00 00 00 02 00 00 00 25 00 00 00  (.....%...
0160  0C 00 00 00 00 00 00 80 28 00 00 00 0C 00 00 00  .....(.....
0170  01 00 00 00 0E 00 00 00 14 00 00 00 00 00 00 00  .....
0180  10 00 00 00 14 00 00 00
.....

```

图 18-4 EMF3.EMF 的十六进位代码

如果把这个 metafile 跟前面的 EMF2.EMF 档案进行比较, 第一个不同点就是 EMF3.EMF 表头部分中的 rclBounds 栏位。在 EMF2.EMF 中, 它指出图像限定在座标 (0x64, 0x64) 和 (0xC8, 0xC8) 区域内。而在 EMF3.EMF 中, 座标是 (0x60, 0x60) 和 (0xCC, 0xCC)。这表示使用了较粗的笔。rclFrame 栏位 (以 0.01mm 为单位指出图像大小) 也受到影响。

EMF2.EMF 中的 nBytes 栏位 (偏移量为 0x0030) 显示该 metafile 长度为 0xFA 位元组, EMF3.EMF 中长度为 0x0188 位元组。EMF2.EMFmetafile 包含 7 个记录 (一个表头纪录, 5 个 GDI 函式呼叫和一个档案结束记录), 但是 EMF3.EMF 档案包含 15 个记录。多出的 8 个记录是两个物件建立函式、4 个对 SelectObject

函式的呼叫和两个对 DeleteObject 函式的呼叫。

nHandles 栏位（在档案中偏移量为 0x0038）指出 GDI 物件的代号个数。该栏位的值总是比 metafile 使用的非内定物件数多一。（Platform SDK 文件解释这个多出来的一是「此表中保留的零索引」）。该栏位在 EMF2.EMF 的值为 1，而在 EMF3.EMF 中的值为 3，多出的数指出了画笔和画刷。

让我们跳到档案中偏移量为 0x0088 的地方，即第二个记录（表头纪录之後的第一个记录）。记录型态为 0x27，对应常数为 EMR_CREATEBRUSHINDIRECT。该 metafile 记录用於 CreateBrushIndirect 函式，此函式需要指向 LOGBRUSH 结构的指标作为参数。该记录的长度为 0x18（或 24）位元组。

每个被选入 metafile 装置内容的非备用 GDI 物件得到一个号码，该号码从 1 开始编号。这在此记录的下 4 个位元组中指出，在 metafile 中的偏移量是 0x0090。此记录下面的 3 个 4 位元组栏位分别对应 LOGBRUSH 结构的 3 个栏位：0x00000000（BS_SOLID 的 lbStyle 栏位）、0x00FF0000（lbColor 栏位）和 0x00000000（lbHatch 栏位）。

下一个记录在 EMF3.EMF 中的偏移量为 0x00A0，记录型态为 0x25，或 EMR_SELECTOBJECT，是用于 SelectObject 呼叫的 metafile 记录。该记录的长度为 0x0C（或 12）位元组，下一个栏位是数值 0x01，指出它是选中的第一个 GDI 物件，这就是逻辑画刷。

EMF3.EMF 中的偏移量 0x00AC 是下一个记录，它的记录型态为 0x5F 或 EMR_EXTCREATEPEN。该记录有 0x34（或 52）个位元组。下一个 4 位元组栏位是 0x02，它表示这是在 metafile 内使用的第二个非备用 GDI 物件。

EMR_EXTCREATEPEN 记录的下 4 个栏位重复记录大小两次，之间用 0 栏位隔开：0x34、0x00、0x34 和 0x00。下一个栏位是 0x00010000，它是 PS_SOLID (0x00000000) 与 PS_GEOMETRIC (0x00010000) 组合的画笔样式。接下来是 5 个单元的宽度，紧接著是 ExtCreatePen 中使用的逻辑画刷结构的 3 个栏位，後接 0 栏位。

如果建立了自订的扩展画笔样式，EMR_EXTCREATEPEN 记录会超过 52 个位元组，这样会影响记录的第二栏位及两个重复的大小栏位。在描述 LOGBRUSH 结构的 3 个栏位後面不会是 0（像在 EMF3.EMF 中那样），而是指出了虚线和空格的数量。这後面接著用于虚线和空格长度的许多栏位。

EMF3.EMF 的下一个 12 位元组的栏位是指出第二个物件（画笔）的另一个 SelectObject 呼叫。接下来的 5 个记录与 EMF2.EMF 中的一样——一个 0x2B (EMR_RECTANGLE) 的记录型态和两组 0x1B (EMR_MOVETOEX) 和 0x36 (EMR_LINETO) 记录。

这些绘图函式後面跟著两组 0x25 (EMR_SELECTOBJECT) 和 0x28 (EMR_DELETEOBJECT) 的 12 位元组记录。选择物件记录具有 0x80000007 和 0x80000000 的参数。在设定高位元时，它指出一个备用物件，在此例中是 0x07 (对应 BLACK_PEN) 和 0x00 (WHITE_BRUSH)。

DeleteObject 呼叫有 2 和 1 两个参数，用於在 metafile 中使用的两个非内定物件。虽然 DeleteObject 函式并不需要装置内容代号作为它的第一个参数，但 GDI 显然保留了 metafile 中使用的被程式删除的物件。

最後，metafile 以 0x0E (EMF_EOF) 记录结束。

总结一下，每当非内定的 GDI 物件首次被选入 metafile 装置内容时，GDI 都会为该物件建立函式的记录编码（此例中，为 EMR_CREATEBRUSHINDIRECT 和 EMR_EXTCREATEPEN）。每个物件有一个依序从 1 开始的唯一数值，此数值由记录的第三个栏位表示。跟在此记录後的是引用该数值的 EMR_SELECTOBJECT 记录。以後，将物件选入 metafile 装置内容时（在中间时期没有被删除），就只需要 EMR_SELECTOBJECT 记录了。

metafile 和点阵图

现在，让我们做点稍微复杂的事，在 metafile 装置内容中绘制一幅点阵图，如程式 18-5 EMF4 所示。

程式 18-5 EMF4

```
EMF4.C
/*-----
--
--      EMF4.C --   Enhanced Metafile Demo #4
--                      (c) Charles Petzold, 1998
--
*/

#define OEMRESOURCE
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("EMF4") ;
    HWND              hwnd ;
    MSG                msg ;
    WNDCLASS           wndclass ;
```

```

        wndclass.style = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc = WndProc ;
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,
IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        if (!RegisterClass (&wndclass))
        {
            MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                            szAppName, MB_ICONERROR) ;

            return 0 ;
        }

        hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #4"),
                               WS_OVERLAPPEDWINDOW,
                               CW_USEDEFAULT, CW_USEDEFAULT,
                               CW_USEDEFAULT, CW_USEDEFAULT,
                               NULL, NULL, hInstance, NULL) ;

        ShowWindow (hwnd, iCmdShow) ;
        UpdateWindow (hwnd) ;
        while (GetMessage (&msg, NULL, 0, 0))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
        return msg.wParam ;
    }

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    BITMAP bm ;
    HBITMAP hbm ;
    HDC hdc, hdcEMF, hdcMem ;
    HENHMETAFILE hemf ;
    PAINTSTRUCT ps ;
    RECT rect ;

    switch (message)
    {
        case WM_CREATE:

```

```

    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf4.emf"), NULL,
    TEXT ("EMF4\0EMF Demo #4\0")) ;

    hbm = LoadBitmap (NULL, MAKEINTRESOURCE (OBM_CLOSE)) ;

    GetObject (hbm, sizeof (BITMAP), &bm) ;

    hdcMem = CreateCompatibleDC (hdcEMF) ;

    SelectObject (hdcMem, hbm) ;

    StretchBlt (hdcEMF, 100, 100, 100, 100,
    hdcMem, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY) ;

    DeleteDC (hdcMem) ;
    DeleteObject (hbm) ;

    hemf = CloseEnhMetaFile (hdcEMF) ;

    DeleteEnhMetaFile (hemf) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;
    rect.left      = rect.right      / 4 ;
    rect.right     = 3 * rect.right   / 4 ;
    rect.top       = rect.bottom / 4 ;
    rect.bottom    = 3 * rect.bottom  / 4 ;

    hemf = GetEnhMetaFile (TEXT ("emf4.emf")) ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

为了方便, EMF4 载入由常数 OEM_CLOSE 指出的系统点阵图。在装置内容中显示点阵图的惯用方法是通过使用 CreateCompatibleDC 建立与目的装置内容 (此例为 metafile 装置内容) 相容的记忆体装置内容。然後, 通过使用

SelectObject 将点阵图选入该记忆体装置内容并且从该记忆体装置内容呼叫 BitBlt 或 StretchBlt 把点阵图画到目的装置内容。结束后，删除记忆体装置内容和点阵图。

您会注意到 EMF4 也呼叫 GetObject 来确定点阵图的大小。这对 SelectObject 呼叫是很必要的。

首先，这份程式码储存 metafile 的空间对 GDI 来说就是个挑战。在 StretchBlt 呼叫前根本没有别的 GDI 函式去处理 metafile 的装置内容。因此，让我们来看一看 EMF4. EMF 里头是如何做的，图 18-5 只显示了一部分。

```

0000  01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00      .....d...d...
0010  C7 00 00 00 C7 00 00 00 35 0C 00 00 35 0C 00 00      .....5...5...
0020  4B 18 00 00 4B 18 00 00 20 45 4D 46 00 00 01 00      K...K...EMF...
0030  F0 0E 00 00 03 00 00 00 01 00 00 00 12 00 00 00      .....
0040  64 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00      d.....
0050  40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00      @.....
0060  00 00 00 00 45 00 4D 00 46 00 34 00 00 00 45 00      ....E.M.F.4...E.
0070  4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00      M.F...D.e.m.o...
0080  23 00 34 00 00 00 00 00 4D 00 00 00 54 0E 00 00      #.4....M...T...
0090  64 00 00 00 64 00 00 00 C7 00 00 00 C7 00 00 00      d...d.....
00A0  64 00 00 00 64 00 00 00 64 00 00 00 64 00 00 00      d...d...d...d...
00B0  20 00 CC 00 00 00 00 00 00 00 00 00 00 00 80 3F      .....?
00C0  00 00 00 00 00 00 00 00 00 00 80 3F 00 00 00 00      .....?....
00D0  00 00 00 00 FF FF FF 00 00 00 00 00 6C 00 00 00      .....l...
00E0  28 00 00 00 94 00 00 00 C0 0D 00 00 28 00 00 00      (.....(...
00F0  16 00 00 00 28 00 00 00 28 00 00 00 16 00 00 00      ....(...(.....
0100  01 00 20 00 00 00 00 00 C0 0D 00 00 00 00 00 00      .. .....
0110  00 00 00 00 00 00 00 00 00 00 00 00 00 C0 C0 C0 00      .....
0120  C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00      .....
. . . .
0ED0  C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 0E 00 00 00      .....
0EE0  14 00 00 00 00 00 00 00 10 00 00 00 14 00 00 00      .....

```

图 18-5 EMF4. EMF 的部分十六进位代码

此 metafile 只包含 3 个记录——表头纪录、0x0E54 位元组长度的 0x4D (或 EMR_STRETCHBLT) 和档案结束记录。

我不解释该记录每个栏位的含义，但我会指出关键部分，以便理解 GDI 把 EMF4. C 中的一系列函式呼叫转化为单个 metafile 记录的方法。

GDI 已经把原始的与设备相关的点阵图转化为与装置无关的点阵图 (DIB)。整个 DIB 储存在记录著自身大小的记录中。我想，在显示 metafile 和点阵图时，GDI 实际上使用 StretchDIBits 函式而不是 StretchBlt。或者，GDI 使用 CreateDIBitmap 把 DIB 转变回与设备相关的点阵图，然後使用记忆体装置内容及 StretchBlt 来显示点阵图。

EMR_STRETCHBLT 记录开始於 metafile 的偏移量 0x0088 处。DIB 储存在

metafile 中，以偏移量 0x00F4 开始，到 0x0EDC 处的记录结尾结束。DIB 以 BITMAPINFOHEADER 型态的 40 位元组的结构开始。在偏移量 0x011C 处接有 22 个图素行，每行 40 个图素。这是每图素 32 位元的 DIB，所以每个图素需要 4 个位元组。

列举 metafile 内容

当您希望存取 metafile 内的个别记录时，可以使用称作 metafile 列举的程序。如程式 18-6 EMF5 所示。此程式使用 metafile 来显示与 EMF3 相同的图像，但它是通过 metafile 列举来进行的。

程式 18-6 EMF5

```
EMF5.C
/*-----
--
--      EMF5.C --      Enhanced Metafile Demo #5
--                      (c) Charles Petzold, 1998
--
--*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("EMF5") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor    = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
    }
}
```

```

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #5"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

int CALLBACK EnhMetaFileProc (    HDC hdc, HANDLETABLE * pHandleTable,
                                CONST ENHMETARECORD * pEmfRecord,
                                int iHandles, LPARAM pData)
{
    PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfRecord, iHandles) ;
    return TRUE ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    HDC                hdc ;
    HENHMETAFILE       hemf ;
    PAINTSTRUCT        ps ;
    RECT               rect ;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps) ;

            GetClientRect (hwnd, &rect) ;

            rect.left      =    rect.right      / 4 ;
            rect.right     = 3 * rect.right     / 4 ;
            rect.top       =    rect.bottom     / 4 ;
            rect.bottom    = 3 * rect.bottom    / 4 ;
    }
}

```

```

        hemf = GetEnhMetaFile (TEXT ("..\emf3\emf3.emf")) ;

        EnumEnhMetaFile (hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
        DeleteEnhMetaFile (hemf) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

此程式使用 EMF3 程式建立的 EMF3.EMF 档案，所以确定在执行此程式前先执行 EMF3 程式。同时，需要在 Visual C++ 环境中执行两个程式，以确保路径的正确。在处理 WM_PAINT 时，两个程式的主要区别是 EMF3 呼叫 PlayEnhMetaFile，而 EMF5 呼叫 EnumEnhMetaFile。PlayEnhMetaFile 函式有下面的语法：

```
PlayEnhMetaFile (hdc, hemf, &rect) ;
```

第一个参数是要显示的 metafile 的装置内容代号。第二个参数是增强型 metafile 代号。第三个参数是指向描述装置内容平面上矩形的 RECT 结构的指标。Metafile 图像大小被缩放过，以便刚好能够显示在不超过该矩形的区域内。

EnumEnhMetaFile 有 5 个参数，其中 3 个与 PlayEnhMetaFile 一样（虽然 RECT 结构的指标已经移到参数表的末尾）。

EnumEnhMetaFile 的第三个参数是列举函式的名称，它用於呼叫 EnhMetaFileProc。第四个参数是希望传递给列举函式的任意资料的指标，这里将该参数简单地设定为 NULL。

现在看一看列举函式。当呼叫 EnumEnhMetaFile 时，对于 metafile 中的每一个记录，GDI 都将呼叫 EnhMetaFileProc 一次，包括表头纪录和档案结束记录。通常列举函式传回 TRUE，但它可能传回 FALSE 以略过剩下的列举程序。

该列举函式有 5 个参数，稍后会描述它们。在这个程式中，我仅把前 4 个参数传递给 PlayEnhMetaFileRecord，它使 GDI 执行由该记录代表的函式呼叫，好像您明确地呼叫它一样。

EMF5 使用 EnumEnhMetaFile 和 PlayEnhMetaFileRecord 得到的结果与 EMF3 呼叫 PlayEnhMetaFile 得到的结果一样。区别在于 EMF5 现在直接介入了 metafile 的显示程序，并能够存取各个 metafile 记录。这是很有用的。

列举函式的第一个参数是装置内容代号。GDI 从 EnumEnhMetaFile 的第一个参数中简单地取得此代号。列举函式把该代号传递给 PlayEnhMetaFileRecord 来标识图像显示的目的装置内容。

我们先跳到列举函式的第三个参数，它是指向 ENHMETARECORD 型态结构的指标，前面已经提到过。这个结构描述实际的 metafile 记录，就像它亲自在 metafile 中编码一样。

您可以写一些程式码来检查这些记录。您也许不想把某些记录传送到 PlayEnhMetaFileRecord 函式。例如，在 EMF5.C 中，把下行插入到 PlayEnhMetaFileRecord 呼叫的前面：

```
if (pEmfRecord->iType != EMR_LINETO)
```

重新编译程序，执行它，将只看到矩形，而没有两条线。或使用下面的叙述：

```
if (pEmfRecord->iType != EMR_SELECTOBJECT)
```

这个小改变会让 GDI 用内定物件显示图像，而不是用 metafile 所建立的画笔和画刷。

程式中不应该修改 metafile 记录，不过先不要担心这一点。先来看看程式 18-7 EMF6。

程式 18-7 EMF6

```
EMF6.C
/*-----
--
    EMF6.C --    Enhanced Metafile Demo #6
                        (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR lpszCmdLine, int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("EMF6") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
    wndclass.lpfnWndProc
    wndclass.cbClsExtra
    wndclass.cbWndExtra
    wndclass.hInstance
    wndclass.hIcon
    wndclass.hCursor
    wndclass.hbrBackground
    wndclass.lpszMenuName
        = CS_HREDRAW | CS_VREDRAW ;
        = WndProc ;
        = 0 ;
        = 0 ;
        = hInstance ;
        = LoadIcon (NULL, IDI_APPLICATION) ;
        = LoadCursor (NULL, IDC_ARROW) ;
        = GetStockObject (WHITE_BRUSH) ;
        = NULL ;
```

```
wndclass.lpszClassName      = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #6"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

int CALLBACK EnhMetaFileProc (      HDC hdc, HANDLETABLE * pHandleTable,
                                CONST ENHMETARECORD * pEmfRecord,
                                int iHandles, LPARAM pData)
{
    ENHMETARECORD * pEmfr ;
    pEmfr = (ENHMETARECORD *) malloc (pEmfRecord->nSize) ;
    CopyMemory (pEmfr, pEmfRecord, pEmfRecord->nSize) ;
    if (pEmfr->iType == EMR_RECTANGLE)
        pEmfr->iType = EMR_ELLIPSE ;
    PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, iHandles) ;
    free (pEmfr) ;
    return TRUE ;
}

LRESULT CALLBACK WndProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    HDC          hdc ;
    HENHMETAFILE hemf ;
    PAINTSTRUCT  ps ;
    RECT         rect ;

    switch (message)
```

```

{
case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    rect.left          =    rect.right  / 4 ;
    rect.right         = 3 * rect.right / 4 ;
    rect.top           =    rect.bottom / 4 ;
    rect.bottom        = 3 * rect.bottom / 4 ;

    hemf = GetEnhMetaFile (TEXT ("..\emf3\emf3.emf")) ;
    EnumEnhMetaFile (hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}

return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

与 EMF5 一样, EMF6 使用 EMF3 程式建立的 EMF3.EMFmetafile, 因此要在 Visual C++ 中执行这个程式之前先执行过 EMF3 程式。

EMF6 展示了如果在显示 metafile 之前要修改它们, 解决方法是非常简单的: 做个被修改过的副本出来就好了。您可以看到, 列举程序一开始使用 malloc 配置一块 metafile 记录大小的记忆体, 它是由传递给该函式的 pEmfRecord 结构的 nSize 栏位表示的。这个记忆体块的指标储存在变数 pEmfr 中, pEmfr 本身是指向 ENHMETARECORD 结构的指标。

程式使用 CopyMemory 把 pEmfRecord 指向的结构内容复制到 pEmfr 指向的结构中。现在我们就可以做些修改了。程式检查记录是否为 EMR_RECTANGLE 型态, 如果是, 则用 EMR_ELLIPSE 取代 iType 栏位。PEmfr 指标被传递到 PlayEnhMetaFileRecord 然後被释放。结果是程式画出一个椭圆而不是矩形。其他的内容的修改方式都是相同的。

当然, 我们的小改变很容易起作用, 因为 Rectangle 和 Ellipse 函式有同样的参数, 这些参数都定义同一件事——图画的边界框。要进行范围更广的修改需要一些不同 metafile 记录格式的相关知识。

另一个可能性是插入一、两个额外的记录。例如, 用下面的叙述代替 EMF6.C 中的 if 叙述:

```
if (pEmfr->iType == EMR_RECTANGLE)
```

```
{
    PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, nObjects) ;
    pEmfr->iType = EMR_ELLIPSE ;
}
```

无论何时出现 Rectangle 记录,程式都会处理此记录并把它更改为 Ellipse,然後再显示。现在程式将画出矩形和椭圆。

现在讨论一下在列举 metafile 时 GDI 物件处理的方式。

在 metafile 表头中,ENHMETAHEADER 结构的 nHandles 栏位是比在 metafile 中建立的 GDI 物件数还要大的值。因此,对於 EMF5 和 EMF6 中的 metafile,此栏位是 3,表示画笔、画刷和其他东西。「其他东西」的具体内容,稍後我会说明。

您会注意到 EMF5 和 EMF6 中列举函式的倒数第二个参数,也称作 nHandles,它是同一个数,3。

列举函式的第二个参数是指向 HANDLETABLE 结构的指标,在 WINGDI.H 中定义如下:

```
typedef struct tagHANDLETABLE
{
    HGDIOBJ objectHandle [1] ;
}
HANDLETABLE ;
```

HGDIOBJ 资料型态是 GDI 物件的代号,被定义为 32 位元的指标,类似於所有其他 GDI 物件。这是那些带有一个元素的阵列栏位的结构之一。这意味著此栏位具有可变的长度。objectHandle 阵列中的元素数等於 nHandles,在此程式中是 3。

在列举函式中,可以使用以下运算式取得这些 GDI 物件代号:

```
pHandleTable->objectHandle[i]
```

对於 3 个代号,i 是 0、1 和 2。

每次呼叫列举函式时,阵列的第一个元素都将包含所列举的 metafile 代号。这就是前面提到的「其他东西」。

在第一次呼叫列举函式时,表的第二、第三个元素将是 0。它们是画笔和画刷代号的保留位置。

以下是列举函式运作的方式: metafile 中的第一个物件建立函式具有 EMR_CREATEBRUSHINDIRECT 的记录型态,此记录指出了物件编号 1。当把该记录传递给 PlayEnhMetaFileRecord 时,GDI 建立画刷并取得它的代号。此代号储存在 objectHandle 阵列的元素 1(第二个元素)中。当把第一个 EMR_SELECTOBJECT 记录传递给 PlayEnhMetaFileRecord 时,GDI 发现此物件编号为 1,并能够从表中找到该物件实际的代号,而把它用来呼叫 SelectObject。当 metafile 最後删

除画刷时，GDI 将 objectHandle 阵列的元素 1 设定回 0。

通过存取 objectHandle 阵列，可以使用例如 GetObjectType 和 GetObject 等呼叫取得在 metafile 中使用的物件资讯。

嵌入图像

列举 metafile 的最重要应用也许是在现有的 metafile 中嵌入其他图像(甚至是整个 metafile)。事实上，现有的 metafile 保持不变；真正进行的是建立包含现有 metafile 和新嵌入图像的新 metafile。基本的技巧是把 metafile 装置内容代号传递给 EnumEnhMetaFile，作为它的第一个参数。这使您能够在 metafile 装置内容上显示 metafile 记录和 GDI 函数呼叫。

在 metafile 命令序列的开头或结尾嵌入新图像是极简单的——就在 EMR_HEADER 记录之後或在 EMF_EOF 记录之前。然而，如果您熟悉现有的 metafile 结构，就可以把新的绘图命令嵌入所需的任何地方。如程式 18-8 EMF7 所示。

程式 18-8 EMF7

```
EMF7.C
/*-----
-
    EMF7.C -- Enhanced Metafile Demo #7
                (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        PSTR      lpszCmdLine,
int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("EMF7") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
```

```
wndclass.lpszClassName      = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #7"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

int CALLBACK EnhMetaFileProc (      HDC hdc, HANDLETABLE * pHandleTable,
                                  CONST ENHMETARECORD * pEmfRecord,
                                  int iHandles, LPARAM pData)
{
    HBRUSH          hBrush ;
    HPEN            hPen ;
    LOGBRUSH        lb ;

    if (pEmfRecord->iType != EMR_HEADER && pEmfRecord->iType != EMR_EOF)
        PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfRecord,
iHandles) ;
    if (pEmfRecord->iType == EMR_RECTANGLE)
    {
        hBrush = SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
        lb.lbStyle = BS_SOLID ;
        lb.lbColor = RGB (0, 255, 0) ;
        lb.lbHatch = 0 ;

        hPen = SelectObject (hdc,
                                ExtCreatePen (PS_SOLID | PS_GEOMETRIC,
5, &lb, 0, NULL)) ;
        Ellipse (hdc, 100, 100, 200, 200) ;
    }
}
```

```

        DeleteObject (SelectObject (hdc, hPen)) ;
        SelectObject (hdc, hBrush) ;
    }
    return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND  hwnd,   UINT  message,   WPARAM  wParam,LPARAM
lParam)
{
    ENHMETAHEADER          emh ;
    HDC                     hdc,   hdcEMF ;
    HENHMETAFILE            hemfOld, hemf ;
    PAINTSTRUCT             ps ;
    RECT                    rect ;

    switch (message)
    {
    case  WM_CREATE:

        // Retrieve existing metafile and header

        hemfOld = GetEnhMetaFile (TEXT ("..\emf3\emf3.emf")) ;

        GetEnhMetaFileHeader (hemfOld, sizeof (ENHMETAHEADER), &emh) ;

        // Create a new metafile DC

        hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf7.emf"), NULL,
            TEXT ("EMF7\0EMF Demo #7\0")) ;

        // Enumerate the existing metafile

        EnumEnhMetaFile (hdcEMF, hemfOld, EnhMetaFileProc, NULL,
            (RECT *) & emh.rclBounds) ;

        // Clean up

        hemf = CloseEnhMetaFile (hdcEMF) ;

        DeleteEnhMetaFile (hemfOld) ;
        DeleteEnhMetaFile (hemf) ;
        return 0 ;

    case  WM_PAINT:

        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;
        rect.left          =      rect.right   / 4 ;

```

```

        rect.right          = 3 * rect.right / 4 ;
        rect.top            = rect.bottom / 4 ;
        rect.bottom         = 3 * rect.bottom / 4 ;

        hemf = GetEnhMetaFile (TEXT ("emf7.emf")) ;

        PlayEnhMetaFile (hdc, hemf, &rect) ;
        DeleteEnhMetaFile (hemf) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

EMF7 使用 EMF3 程式建立的 EMF3.EMF，所以在执行 EMF7 之前要执行 EMF3 程式建立 metafile。

EMF7 中的 WM_PAINT 处理使用 PlayEnhMetaFile 而不是 EnumEnhMetaFile，而且 WM_CREATE 处理有很大的差别。

首先，程式通过呼叫 GetEnhMetaFile 取得 EMF3.EMF 档案的 metafile 代号，还呼叫 GetEnhMetaFileHeader 得到增强型 metafile 表头记录，目的是在后面的 EnumEnhMetaFile 呼叫中使用 rclBounds 栏位。

接下来，程式建立新的 metafile 档案，名为 EMF7.EMF。CreateEnhMetaFile 函式为 metafile 传回装置内容代号。然後，使用 EMF7.EMF 的 metafile 装置内容代号和 EMF3.EMF 的 metafile 代号呼叫 EnumEnhMetaFile。

现在来看一看 EnhMetaFileProc。如果被列举的记录不是表头纪录或档案结束记录，函式就呼叫 PlayEnhMetaFileRecord 把记录转换为新的 metafile 装置内容（并不一定排除表头纪录或档案结束记录，但它们会使 metafile 变大）。

如果刚转换的记录是 Rectangle 呼叫，则函式建立画笔用绿色的轮廓线和透明的内部来绘制椭圆。注意程式中经由储存先前的画笔和画刷代号来恢复装置内容状态的方法。在此期间，所有这些函式都被插入到 metafile 中（记住，也可以使用 PlayEnhMetaFile 在现有的 metafile 中插入整个 metafile）。

回到 WM_CREATE 处理，程式呼叫 CloseEnhMetaFile 取得新 metafile 的代号。然後，它删除两个 metafile 代号，将 EMF3.EMF 和 EMF7.EMF 档案留在磁片上。

从程式显示输出中可以很明显地看到，椭圆是在矩形之後两条交叉线之前绘制的。

增强型 metafile 浏览器和印表机

使用剪贴簿转换增强型 metafile 非常简单,剪贴簿型态是 CF_ENHMETAFILE。GetClipboardData 函式传回增强型 metafile 代号,SetClipboardData 也使用该 metafile 代号。复制 metafile 时可以使用 CopyEnhMetaFile 函式。如果把增强型 metafile 放在剪贴簿中,Windows 会让需要旧格式的那些程式也可以使用它。如果在剪贴簿中放置旧格式的 metafile,Windows 将也会自动视需要把内容转换为增强型 metafile 的格式。

程式 18-9 EMFVIEW 所示为在剪贴簿中传送 metafile 的程式码,它也允许载入、储存和列印 metafile。

程式 18-9 EMFVIEW

```
EMFVIEW.C
/*-----
-
    EMFVIEW.C --      View Enhanced Metafiles
                                (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("EmfView") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwnd ;
    MSG             msg ;
    WNDCLASS wndclass ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName  = szAppName ;
```

```
if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Viewer"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

hAccel = LoadAccelerators (hInstance, szAppName) ;
while (GetMessage (&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}
return msg.wParam ;
}

HPALETTE CreatePaletteFromMetaFile (HENHMETAFILE hemf)
{
    HPALETTE          hPalette ;
    int               iNum ;
    LOGPALETTE *      plp ;

    if (!hemf)
        return NULL ;
    if (0 == (iNum = GetEnhMetaFilePaletteEntries (hemf, 0, NULL)))
        return NULL ;
    plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
    plp->palVersion    = 0x0300 ;
    plp->palNumEntries = iNum ;

    GetEnhMetaFilePaletteEntries (hemf, iNum, plp->palPalEntry) ;
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return hPalette ;
}
```

```

LRESULT CALLBACK WndProc (   HWND  hwnd,   UINT  message,   WPARAM  wParam,LPARAM
lParam)
{
    static DOCINFO    di = { sizeof (DOCINFO), TEXT ("EmfView: Printing") } ;
    static HENHMETAFILE hemf ;
    static OPENFILENAME ofn ;
    static PRINTDLG          printdlg = { sizeof (PRINTDLG) } ;
    static TCHAR              szFileName  [MAX_PATH],  szTitleName
[MAX_PATH] ;
    static TCHAR              szFilter[] =
                                                                    TEXT
("Enhanced Metafiles (*.EMF)\0*.emf\0")
                                                                    TEXT
("All Files (*.*)\0*.*\0\0") ;

    BOOL                      bSuccess ;
    ENHMETAHEADER             header ;
    HDC                        hdc,   hdcPrn ;
    HENHMETAFILE               hemfCopy ;
    HMENU                      hMenu ;
    HPALETTE                   hPalette ;
    int                        i, iLength, iEnable ;
    PAINTSTRUCT                ps ;
    RECT                       rect ;
    PTSTR                      pBuffer ;

    switch (message)
    {
        case WM_CREATE:
            // Initialize OPENFILENAME structure
            ofn.lStructSize          =          sizeof
(OPENFILENAME) ;

            ofn.hwndOwner            = hwnd ;
            ofn.hInstance            = NULL ;
            ofn.lpstrFilter          = szFilter ;
            ofn.lpstrCustomFilter    = NULL ;
            ofn.nMaxCustFilter       = 0 ;
            ofn.nFilterIndex        = 0 ;
            ofn.lpstrFile            = szFileName ;
            ofn.nMaxFile            = MAX_PATH ;
            ofn.lpstrFileTitle       = szTitleName ;
            ofn.nMaxFileTitle       = MAX_PATH ;
            ofn.lpstrInitialDir      = NULL ;
            ofn.lpstrTitle           = NULL ;
            ofn.Flags                = 0 ;
            ofn.nFileOffset          = 0 ;
            ofn.nFileExtension       = 0 ;
            ofn.lpstrDefExt          = TEXT ("emf") ;
            ofn.lCustData            = 0 ;
    }
}

```

```

        ofn.lpfHook                = NULL ;
        ofn.lpTemplateName         = NULL ;
        return 0 ;

case WM_INITMENUPOPUP:
        hMenu = GetMenu (hwnd) ;

        iEnable = hemf ? MF_ENABLED : MF_GRAYED ;

        EnableMenuItem (hMenu, IDM_FILE_SAVE_AS,          iEnable) ;
        EnableMenuItem (hMenu, IDM_FILE_PRINT,            iEnable) ;
        EnableMenuItem (hMenu, IDM_FILE_PROPERTIES,       iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_CUT,              iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_COPY,
iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_DELETE,
iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_PASTE,
        IsClipboardFormatAvailable (CF_ENHMETAFILE) ?
        MF_ENABLED : MF_GRAYED) ;
        return 0 ;

case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDM_FILE_OPEN:
            // Show the File Open dialog box

            ofn.Flags = 0 ;

            if (!GetOpenFileName (&ofn))
                return 0 ;

            // If there's an existing EMF, get rid of it.

            if (hemf)
            {
                DeleteEnhMetaFile (hemf) ;
                hemf = NULL ;
            }

            // Load the EMF into memory

            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            hemf = GetEnhMetaFile (szFileName) ;

            ShowCursor (FALSE) ;

```



```
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

// Invalidate the client area for later update

        InvalidateRect (hwnd, NULL, TRUE) ;

        if (hemf == NULL)
        {
            MessageBox (    hwnd, TEXT ("Cannot load metafile"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        }
        return 0 ;

case  IDM_FILE_SAVE_AS:
    if (!hemf)
        return 0 ;

        // Show the File Save dialog box

        ofn.Flags = OFN_OVERWRITEPROMPT ;

        if (!GetSaveFileName (&ofn))
            return 0 ;

        // Save the EMF to disk file

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        hemfCopy = CopyEnhMetaFile (hemf, szFileName) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
        if (hemfCopy)
        {
            DeleteEnhMetaFile (hemf) ;
            hemf = hemfCopy ;
        }
        else
            MessageBox (    hwnd, TEXT ("Cannot save metafile"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;

case  IDM_FILE_PRINT:
    // Show the Print dialog box and get printer DC

    printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;
```

```
        if (!PrintDlg (&printdlg))
            return 0 ;

        if (NULL == (hdcPrn = printdlg.hDC))
        {
            MessageBox ( hwnd, TEXT ("Cannot obtain printer DC"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;
        }
        // Get size of printable area of page

        rect.left  = 0 ;
        rect.right = GetDeviceCaps (hdcPrn, HORZRES) ;
        rect.top   = 0 ;
        rect.bottom = GetDeviceCaps (hdcPrn, VERTRES) ;

        bSuccess = FALSE ;

        // Play the EMF to the printer

        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
        {
            PlayEnhMetaFile (hdcPrn, hemf, &rect) ;

            if (EndPage (hdcPrn) > 0)
            {
                bSuccess = TRUE ;
                EndDoc (hdcPrn) ;
            }
        }
        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        DeleteDC (hdcPrn) ;

        if (!bSuccess)
            MessageBox (  hwnd, TEXT ("Could not print metafile"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;

    case  IDM_FILE_PROPERTIES:
        if (!hemf)
            return 0 ;

        iLength = GetEnhMetaFileDescription (hemf, 0, NULL) ;
```

```

        pBuffer = malloc ((iLength + 256) * sizeof (TCHAR)) ;

        GetEnhMetaFileHeader      (hemf,      sizeof
(ENHMETAHEADER), &header) ;

        // Format header file information

        i = wsprintf (pBuffer,
            TEXT ("Bounds = (%i, %i) to (%i, %i)
pixels\n"),

            header.rc1Bounds.left, header.rc1Bounds.top,

            header.rc1Bounds.right, header.rc1Bounds.bottom) ;

        i += wsprintf (pBuffer + i,
            TEXT ("Frame =
(%i, %i) to (%i, %i) mms\n"),

            header.rc1Frame.left, header.rc1Frame.top,

            header.rc1Frame.right, header.rc1Frame.bottom) ;

        i += wsprintf (pBuffer + i,
            TEXT
("Resolution = (%i, %i) pixels")

            TEXT (" = (%i,
%i) mms\n"),

            header.sz1Device.cx, header.sz1Device.cy,

            header.sz1Millimeters.cx,

            header.sz1Millimeters.cy) ;

        i += wsprintf (pBuffer + i,
            TEXT ("Size = %i, Records = %i, ")
            TEXT ("Handles = %i, Palette entries = %i\n"),
            header.nBytes, header.nRecords,
            header.nHandles, header.nPalEntries) ;
        // Include the metafile description, if present

        if (iLength)
        {
            i += wsprintf (pBuffer + i, TEXT ("Description = ")) ;
            GetEnhMetaFileDescription (hemf, iLength, pBuffer + i) ;
            pBuffer [lstrlen (pBuffer)] = '\t' ;
        }

```

```
    MessageBox (hwnd, pBuffer, TEXT ("Metafile Properties"), MB_OK) ;
        free (pBuffer) ;
        return 0 ;

case  IDM_EDIT_COPY:
case  IDM_EDIT_CUT:
        if (!hemf)
            return 0 ;

        // Transfer metafile copy to the clipboard

        hemfCopy = CopyEnhMetaFile (hemf, NULL) ;

        OpenClipboard (hwnd) ;
        EmptyClipboard () ;
        SetClipboardData (CF_ENHMETAFILE, hemfCopy) ;
        CloseClipboard () ;

        if (LOWORD (wParam) == IDM_EDIT_COPY)
            return 0 ;
        // fall through if IDM_EDIT_CUT
case  IDM_EDIT_DELETE:
        if (hemf)
        {
            DeleteEnhMetaFile (hemf) ;
            hemf = NULL ;
            InvalidateRect (hwnd, NULL, TRUE) ;
        }
        return 0 ;

case  IDM_EDIT_PASTE:
        OpenClipboard (hwnd) ;
        hemfCopy = GetClipboardData (CF_ENHMETAFILE) ;

        CloseClipboard () ;
        if (hemfCopy && hemf)
        {
            DeleteEnhMetaFile (hemf) ;
            hemf = NULL ;
        }

        hemf = CopyEnhMetaFile (hemfCopy, NULL) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

case  IDM_APP_ABOUT:
        MessageBox (        hwnd,        TEXT
```

```
("Enhanced Metafile Viewer\n")
        TEXT ("(c) Charles Petzold, 1998"),

szAppName, MB_OK) ;

        return 0 ;

case  IDM_APP_EXIT:

        SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
        return 0 ;

        }
        break ;

case  WM_PAINT:

        hdc = BeginPaint (hwnd, &ps) ;

        if (hemf)
        {
                if ( hPalette = CreatePaletteFromMetaFile (hemf))
                {
                        SelectPalette (hdc, hPalette, FALSE) ;
                        RealizePalette (hdc) ;
                }
                GetClientRect (hwnd, &rect) ;
                PlayEnhMetaFile (hdc, hemf, &rect) ;

                if (hPalette)
                        DeleteObject (hPalette) ;
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;

case  WM_QUERYNEWPALETTE:

        if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
                return FALSE ;

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        InvalidateRect (hwnd, NULL, FALSE) ;

        DeleteObject (hPalette) ;
        ReleaseDC (hwnd, hdc) ;
        return TRUE ;

case  WM_PALETTECHANGED:

        if ((HWND) wParam == hwnd)
                break ;
```

```

        if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
            break ;

        hdc = GetDC (hwnd) ;
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
        UpdateColors (hdc) ;

        DeleteObject (hPalette) ;
        ReleaseDC (hwnd, hdc) ;
        break ;

case WM_DESTROY:
    if (hemf)
        DeleteEnhMetaFile (hemf) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

EMFVIEW.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

EMFVIEW MENU DISCARDABLE

BEGIN

POPUP "&File"

BEGIN

MENUITEM "&Open\tCtrl+O", IDM_FILE_OPEN

MENUITEM "Save &As...", IDM_FILE_SAVE_AS

MENUITEM SEPARATOR

MENUITEM "&Print...\tCtrl+P",IDM_FILE_PRINT

MENUITEM SEPARATOR

MENUITEM "&Properties", IDM_FILE_PROPERTIES

MENUITEM SEPARATOR

MENUITEM "E&xit", IDM_APP_EXIT

END

POPUP "&Edit"

BEGIN

MENUITEM "Cu&t\tCtrl+X", IDM_EDIT_CUT

MENUITEM "&Copy\tCtrl+C", IDM_EDIT_COPY

MENUITEM "&Paste\tCtrl+V", IDM_EDIT_PASTE

MENUITEM "&Delete\tDel", IDM_EDIT_DELETE

```

        END
        POPUP "Help"
        BEGIN
            MENUITEM "&About EmfView...",          IDM_APP_ABOUT
        END
    END
END

```

```

////////////////////////////////////
/

```

```
// Accelerator
```

```
EMFVIEW ACCELERATORS DISCARDABLE
```

```
BEGIN
```

```
"C",IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
```

```
"O",IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
```

```
"P",IDM_FILE_PRINT,VIRTKEY, CONTROL, NOINVERT
```

```
"V",IDM_EDIT_PASTE,VIRTKEY, CONTROL, NOINVERT
```

```
VK_DELETE,IDM_EDIT_DELETE,VIRTKEY, NOINVERT
```

```
"X",IDM_EDIT_CUT,VIRTKEY, CONTROL, NOINVERT
```

```
END
```

[RESOURCE.H \(摘录\)](#)

```
// Microsoft Developer Studio generated include file.
```

```
// Used by EmfView.rc
```

```

#define          IDM_FILE_OPEN          40001
#define          IDM_FILE_SAVE_AS      40002
#define          IDM_FILE_PRINT        40003
#define          IDM_FILE_PROPERTIES   40004
#define          IDM_APP_EXIT          40005
#define          IDM_EDIT_CUT          40006
#define          IDM_EDIT_COPY         40007
#define          IDM_EDIT_PASTE        40008
#define          IDM_EDIT_DELETE       40009
#define          IDM_APP_ABOUT         40010

```

EMFVIEW 也支援完整的调色盘处理，以便支援有调色盘编码资讯的 metafile。（透过呼叫 Selectpalette 来进行）。该程式在 CreatePaletteFromMetaFile 函式中处理调色盘，在处理 WM_PAINT 显示 metafile 以及处理 WM_QUERYNEWPALETTE 和 WM_PALETTECHANGED 讯息时，呼叫这个函式。

在回应功能表中的「Print」命令时，EMFVIEW 显示普通的印表机对话方块，然後取得页面中可列印区域的大小。Metafile 被缩放成适当尺寸以填入整个区域。EMFVIEW 在视窗中以类似方式显示 metafile。

「File」功能表中的「Properties」项使 EMFVIEW 显示包含 metafile 表头资讯的讯息方块。

如果列印本章前面建立的 EMF2.EMFmetafile 图像，您将会发现用高解析度

的印表机列印出的线条非常细，几乎看不清楚线条的锯齿。列印向量图像时应该使用较宽的画笔（例如，一点宽）。本章後面所示的直尺图像就是这样做的。

显示精确的 metafile 图像

Metafile 图像的好处在於它能够以任意大小缩放并且仍能保持一定的逼真度。这是因为 metafile 通常由一系列向量图形的基本图形组成，基本图形是指线条、填入的区域以及轮廓字体等等。扩大或缩小图像只是简单地缩放定义这些基本图形的所有座标点。另一方面，对点阵图来说，压缩图像会遗漏整行列的图素，因而失去重要的显示资讯。

当然，metafile 的压缩并不是完美无缺的。我们所使用的图形输出设备的图素大小是有限的。当 metafile 图像压缩到一定大小时，组成 metafile 的大量线条会变成模糊的斑点，同时区域填入图案和混色看起来也很奇怪。如果 metafile 中包含嵌入的点阵图或旧的点阵字体，同样会引起类似的问题。

尽管如此，大多数情况下 metafile 可以任意地缩放。这在把 metafile 放入文书处理或桌上印刷文件内时非常有用。一般来说，在上述的应用程式中选择 metafile 图像时，会出现围绕图像的矩形，您可以用滑鼠拖动该矩形，将它缩放为任意大小。图像送到印表机时，它也具有同样对应的大小。

然而，有时任意缩放 metafile 并不是个好主意。例如：假设您有一个储存著存款客户签名样本的银行系统，这些签名以一系列折线的方式储存在 metafile 中。将 metafile 变宽或变高会使签名变形，因此应该保持图像的纵横比一致。

在前面的范例程式中，是以显示区域的大小来确定 PlayEnhMetaFile 呼叫使用的围绕矩形范围。所以，如果改变程式表单的大小，也就改变了图像的大小。这与在文书处理文件中改变 metafile 图像大小的概念相似。

正确地显示 metafile 图像（以特定的度量单位或用适当的纵横比），需要使用 metafile 表头中的大小资讯并根据此资讯设定矩形结构。

在本章剩下的范例程式中将使用名为 EMF.C 的程式架构，它包括列印处理的程式码、资源描述档 EMF.RC 和表头档案 RESOURCE.H。程式 18-10 显示了这些档案以及 EMF8.C 程式，该程式使用这些档案显示一把 6 英寸的直尺。

程式 18-10 EMF8

```
EMF8.C
/*-----
-
EMF8.C -- Enhanced Metafile Demo #8
(c) Charles Petzold, 1998
-----
```



```

-*/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF8") ;
TCHAR szTitle          [] = TEXT ("EMF8: Enhanced Metafile Demo #8") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
    int                iAdj, i, iHeight ;
    LOGFONT            lf ;
    TCHAR              ch ;

    iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
                // Black pen with 1-point width
    SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
                // Rectangle surrounding entire pen (with
adjustment)
    Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
                // Tick marks
    for (i = 1 ; i < 96 ; i++)
    {
        if (i % 16 == 0) iHeight = cy / 2 ;           // inches
        else if (i % 8 == 0) iHeight = cy / 3 ;       // half inches
        else if (i % 4 == 0) iHeight = cy / 5 ;       // quarter inches
        else if (i % 2 == 0) iHeight = cy / 8 ;       // eighths
        else              iHeight = cy / 12 ;         // sixteenths

        MoveToEx (hdc, i * cx / 96, cy, NULL) ;
        LineTo   (hdc, i * cx / 96, cy - iHeight) ;
    }

                // Create logical font
    FillMemory (&lf, sizeof (lf), 0) ;
    lf.lfHeight = cy / 2 ;
    lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

    SelectObject      (hdc, CreateFontIndirect (&lf)) ;
    SetTextAlign      (hdc, TA_BOTTOM | TA_CENTER) ;
    SetBkMode         (hdc, TRANSPARENT) ;

                // Display numbers

    for (i = 1 ; i <= 5 ; i++)
    {
        ch = (TCHAR) (i + '0') ;
        TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
    }

                // Clean up
    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
}

```

```

        DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
    }

void CreateRoutine (HWND hwnd)
{
    HDC                hdcEMF ;
    HENHMETAFILE hemf ;
    int                cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf8.emf"), NULL,
                                TEXT ("EMF8\0EMF Demo #8\0")) ;
    if (hdcEMF == NULL)
        return ;
    cxMms        = GetDeviceCaps (hdcEMF, HORZSIZE) ;
    cyMms        = GetDeviceCaps (hdcEMF, VERTSIZE) ;
    cxPix        = GetDeviceCaps (hdcEMF, HORZRES) ;
    cyPix        = GetDeviceCaps (hdcEMF, VERTRES) ;

    xDpi         = cxPix * 254 / cxMms / 10 ;
    yDpi         = cyPix * 254 / cyMms / 10 ;

    DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
    hemf = CloseEnhMetaFile (hdcEMF) ;
    DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER      emh ;
    HENHMETAFILE        hemf ;
    int                cxImage, cyImage ;
    RECT                rect ;

    hemf = GetEnhMetaFile (TEXT ("emf8.emf")) ;
    GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
    cxImage = emh.rclBounds.right - emh.rclBounds.left ;
    cyImage = emh.rclBounds.bottom - emh.rclBounds.top ;

    rect.left          = (cxArea - cxImage) / 2 ;
    rect.right         = (cxArea + cxImage) / 2 ;
    rect.top           = (cyArea - cyImage) / 2 ;
    rect.bottom        = (cyArea + cyImage) / 2 ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
}

```

EMF.C

/*-----

```

--
    EMF.C --      Enhanced Metafile Demonstration Shell Program
                                (c) Charles Petzold, 1998
-----

-*/

#include <windows.h>
#include <commdlg.h>
#include "..\\emf8\\resource.h"

extern void CreateRoutine (HWND) ;
extern void PaintRoutine (HWND, HDC, int, int) ;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HANDLE hInst ;
extern TCHAR szClass [] ;
extern TCHAR szTitle [] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    TCHAR                szResource [] = TEXT ("EMF") ;
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS            wndclass ;

    hInst = hInstance ;
    wndclass.style        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc  = WndProc ;
    wndclass.cbClsExtra   = 0 ;
    wndclass.cbWndExtra   = 0 ;
    wndclass.hInstance    = hInstance ;
    wndclass.hIcon        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName  = szResource ;
    wndclass.lpszClassName = szClass ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szClass, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szClass, szTitle,
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,

```

```

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

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

BOOL PrintRoutine (HWND hwnd)
{
    static DOCINFO          di ;
    static PRINTDLG          printdlg = { sizeof (PRINTDLG) } ;
    static TCHAR             szMessage [32] ;
    BOOL                     bSuccess = FALSE ;
    HDC                      hdcPrn ;
    int                      cxPage, cyPage ;

    printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;
    if (!PrintDlg (&printdlg))
        return TRUE ;
    if (NULL == (hdcPrn = printdlg.hDC))
        return FALSE ;
    cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
    cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

    lstrcpy (szMessage, szClass) ;
    lstrcat (szMessage, TEXT (": Printing")) ;

    di.cbSize          = sizeof (DOCINFO) ;
    di.lpszDocName      = szMessage ;

    if (StartDoc (hdcPrn, &di) > 0)
    {
        if (StartPage (hdcPrn) > 0)
        {
            PaintRoutine (hwnd, hdcPrn, cxPage, cyPage) ;
            if (EndPage (hdcPrn) > 0)
            {
                EndDoc (hdcPrn) ;
                bSuccess = TRUE ;
            }
        }
    }
}

```

```

    }
    DeleteDC (hdcPrn) ;
    return bSuccess ;
}

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,LPARAM
lParam)
{
    BOOL                                bSuccess ;
    static int                          cxClient, cyClient ;
    HDC                                hdc ;
    PAINTSTRUCT                          ps ;

    switch (message)
    {
    case WM_CREATE:
        CreateRoutine (hwnd) ;
        return 0 ;

    case WM_COMMAND:
        switch (wParam)
        {
        case IDM_PRINT:
            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            bSuccess = PrintRoutine (hwnd) ;

            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            if (!bSuccess)
                MessageBox (hwnd, TEXT ("Error encountered during printing"),
                    szClass, MB_ICONASTERISK | MB_OK) ;
            return 0 ;

        case IDM_EXIT:
            SendMessage (hwnd, WM_CLOSE, 0, 0) ;
            return 0 ;

        case IDM_ABOUT:
            MessageBox (hwnd, TEXT ("Enhanced Metafile Demo Program\n")
                TEXT ("Copyright (c) Charles Petzold, 1998"),
                szClass, MB_ICONINFORMATION | MB_OK) ;
            return 0 ;
        }
        break ;
    }
}

```

```

    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        PaintRoutine (hwnd, hdc, cxClient, cyClient) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

EMF.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Menu

EMF MENU DISCARDABLE

BEGIN

POPUP "&File"

BEGIN

MENUITEM "&Print...",

IDM_PRINT

MENUITEM SEPARATOR

MENUITEM "E&xit",

IDM_EXIT

END

POPUP "&Help"

BEGIN

MENUITEM "&About...",

IDM_ABOUT

END

END

RESOURCE.H (摘录)

// Microsoft Developer Studio generated include file.

// Used by Emf.rc

//

#define IDM_PRINT 40001

```
#define IDM_EXIT          40002
#define IDM_ABOUT         40003
```

在处理 WM_CREATE 讯息处理期间, EMF.C 呼叫名为 CreateRoutine 的外部函式, 该函式建立 metafile。EMF.C 在两个地方呼叫 PaintRoutine 函式: 一处在 WM_PAINT 讯息处理期间, 另一处在函式 PrintRoutine 中以回应功能表命令列印图像。

因为现代的印表机通常比视讯显示器有更高的解析度, 列印 metafile 的能力是测试以特定大小处理图像能力的重要工具。当 EMF8 建立的 metafile 图像以特定大小显示时, 最有意义。该图像是一把 6 英寸长 1 英寸宽的直尺, 每英寸分为十六格, 数字从 1 到 5 为 TrueType 字体。

要绘制一把 6 英寸的直尺, 需要知道一些设备解析度的知识。EMF8.C 中的 CreateRoutine 函式首先建立 metafile, 然後使用从 CreateEnhMetaFile 传回的装置内容代号呼叫 GetDeviceCaps 四次。这些呼叫取得单位分别为毫米和图素的显示平面的高度与宽度。

这听起来有点怪。Metafile 装置内容通常是作为 GDI 绘制命令的储存媒介, 它不是像视讯显示器或印表机的真正设备, 那么它的宽度和高度从何而来?

您可能已经想起来了, CreateEnhMetaFile 的第一个参数被称作「参考装置内容」。GDI 用这为 metafile 建立设备特徵。如果参数设定为 NULL (如 EMF8 中), GDI 就把显示器作为参考装置内容。因而, 当 EMF8 使用装置内容呼叫 GetDeviceCaps 时, 它实际上取得有关显示器的资讯。

EMF8.C 以图素大小除以毫米大小并乘以 25.4 (1 英寸为 25.4 毫米) 计算以每英寸的点数为单位的解析度。

即使我们非常认真地以 metafile 直尺的正确大小绘制它, 但是这样子作还是不够的。PlayEnhMetaFile 函式在显示图像时, 使用作为最後一个参数传递给它的矩形来缩放图像大小, 因此该矩形必须设定为直尺的大小。

由於此原因, EMF8 中的 PaintRoutine 函式呼叫 GetEnhMetaFileHeader 函式来取得 metafile 的表头资讯。ENHMETAHEADER 结构的 rc1Bounds 栏位指出以图素为单位的 metafile 图像的围绕矩形。程式使用此资讯使直尺位於显示区域中央, 如图 18-6 所示。

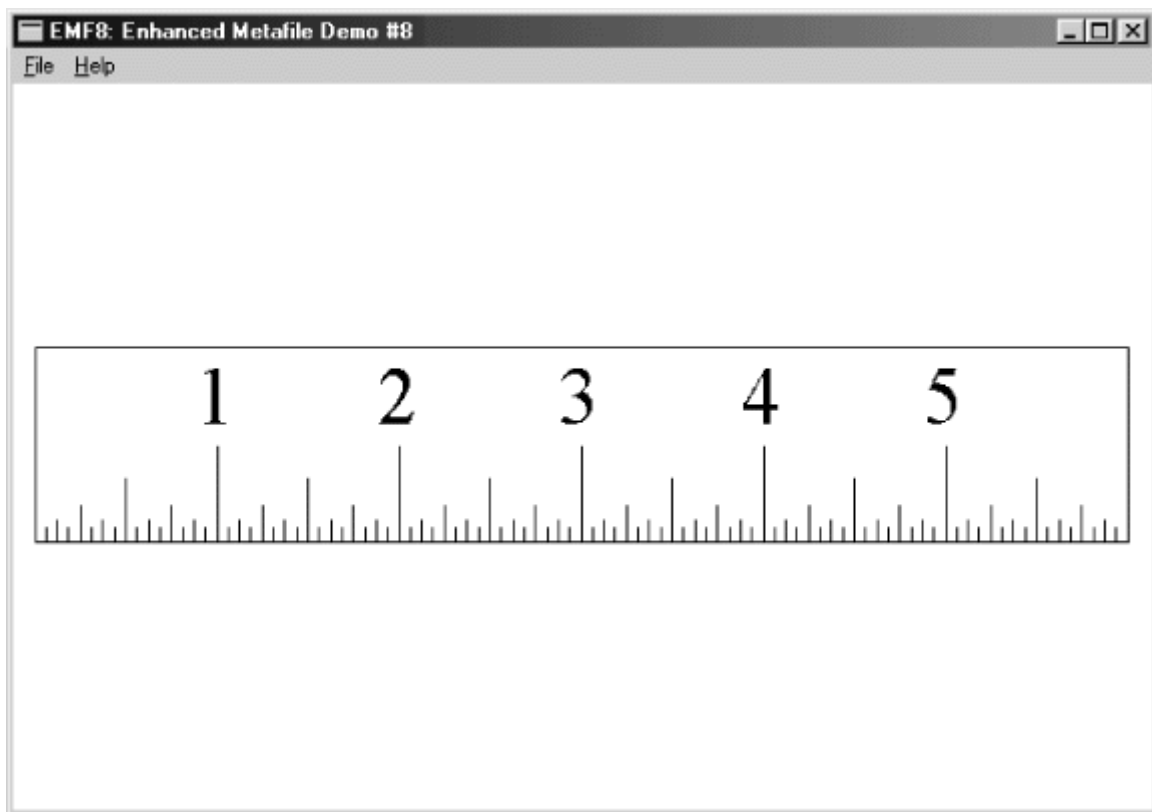


图 18-6 EMF8 得萤幕显示

记住，如果您拿直尺与萤幕中的直尺比较时，两者并不一定非常吻合。如同第五章中所论述的，显示器只能近似地实际显示尺寸。

既然这样做好像有用了，现在就来试著列印图像。哇！如果您有一台 300dpi 的雷射印表机，那么列印出的直尺的宽将会是 11/3 英寸。这是由於我们依据视讯显示器的图素尺寸来列印。虽然您可能认为这把小尺很可爱，但它不是我们所需要的。让我们再试一试。

ENHMETAHEADER 结构包括两个描述图像大小的矩形结构。第一个是 `rc1Bounds`，EMF8 使用这个，它以图素为单位给出图像的大小。第二为 `rc1Frame`，它以 0.01 毫米为单位给出图像的大小。这两个栏位之间的关系是由最初建立 metafile 时使用的参考装置内容决定的，在此情况下为显示器（metafile 表头也包括两个名为 `sz1Device` 和 `sz1Millimeters` 的栏位，它们是 `SIZEL` 结构，分别以图素单位和毫米单位指出了参考设备的大小，这与从 `GetDeviceCaps` 得到的资讯一样）。

EMF9 使用图像的毫米大小资讯，如程式 18-11 所示。

程式 18-11 EMF9

```
EMF9.C
/*-----
-
EMF9.C -- Enhanced Metafile Demo #9
(c) Charles Petzold, 1998
-----
```



```

*/

#include <windows.h>
#include <string.h>

TCHAR szClass          [] = TEXT ("EMF9") ;
TCHAR szTitle          [] = TEXT ("EMF9: Enhanced Metafile Demo #9") ;

void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER          emh ;
    HENHMETAFILE           hemf ;
    int                    cxMms,   cyMms,   cxPix,   cyPix,
cxImage, cyImage ;
    RECT                   rect ;

    cxMms                  = GetDeviceCaps (hdc, HORZSIZE) ;
    cyMms                  = GetDeviceCaps (hdc, VERTSIZE) ;
    cxPix                  = GetDeviceCaps (hdc, HORZRES) ;
    cyPix                  = GetDeviceCaps (hdc, VERTRES) ;

    hemf = GetEnhMetaFile (TEXT ("..\emf8\emf8.emf")) ;
    GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
    cxImage                = emh.rclFrame.right - emh.rclFrame.left ;
    cyImage                = emh.rclFrame.bottom - emh.rclFrame.top ;

    cxImage                = cxImage * cxPix / cxMms / 100 ;
    cyImage                = cyImage * cyPix / cyMms / 100 ;

    rect.left              = (cxArea - cxImage) / 2 ;
    rect.right              = (cxArea + cxImage) / 2 ;
    rect.top                = (cyArea - cyImage) / 2 ;
    rect.bottom             = (cyArea + cyImage) / 2 ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
}

```

EMF9 使用 EMF8 建立的 metafile, 因此确定执行 EMF8。

EMF9 中的 PaintRoutine 函式首先使用目的装置内容呼叫 GetDeviceCaps 四次。像在 EMF8 中的 CreateRoutine 函式一样, 这些呼叫提供有关设备解析度的资讯。在得到 metafile 代号之後, 它取得表头结构并使用 rclFrame 栏位来计算以 0.01 毫米为单位的 metafile 图像大小。这是第一步。

然後，函式通过乘以输出设备的图素大小、除以毫米大小再除以 100（因为度量尺寸以 0.01 毫米为单位）将此大小转换为图素大小。现在，PaintRoutine 函式具有以图素为单位的直尺大小——与显示器无关。这是适合目的设备的图素大小，而且很容易使图像居中对齐。

就显示器而言，EMF9 的显示与 EMF8 显示的一样。但是如果从 EMF9 列印直尺，您会看到更正常的直尺——6 英寸长、1 英寸宽。

缩放比例和纵横比

您也可能想要使用 EMF8 建立的直尺 metafile，而不必显示 6 英寸的图像。保持图像正确的 6 比 1 的纵横比是重要的。如前所述，在文书处理程式或别的应用程式中使用围绕方框来改变 metafile 的大小是很方便的，但是这样会导致某种程度的失真。在这种应用程式中，应该给使用者一个选项来保持原先的纵横比，而不用管围绕方框的大小如何变化。这就是说，传递给 PlayEnhMetaFile 的矩形结构不能直接由使用者选择的围绕方框定义。传递给该函式的矩形结构只是围绕方框的一部分。

让我们看一看程式 18-12 EMF10 是如何做的。

程式 18-12 EMF10

```
EMF10.C
/*-----
    EMF10.C -- Enhanced Metafile Demo #10
                                (c) Charles Petzold, 1998
-----*/

#include <windows.h>
TCHAR szClass      [] = TEXT ("EMF10") ;
TCHAR szTitle      [] = TEXT ("EMF10: Enhanced Metafile Demo #10") ;
void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER      emh ;
    float               fScale ;
    HENHMETAFILE        hemf ;
    int                 cxMms,   cyMms,   cxPix,   cyPix,
cxImage, cyImage ;
    RECT                rect ;

    cxMms               = GetDeviceCaps (hdc, HORZSIZE) ;
```

```

    cyMms          = GetDeviceCaps (hdc, VERTSIZE) ;
    cxPix          = GetDeviceCaps (hdc, HORZRES) ;
    cyPix          = GetDeviceCaps (hdc, VERTRES) ;

    hemf = GetEnhMetaFile (TEXT ("..\emf8\emf8.emf")) ;

    GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

    cxImage        = emh.rclFrame.right - emh.rclFrame.left ;
    cyImage        = emh.rclFrame.bottom - emh.rclFrame.top ;

    cxImage        = cxImage * cxPix / cxMms / 100 ;
    cyImage        = cyImage * cyPix / cyMms / 100 ;

    fScale         = min ((float) cxArea / cxImage, (float) cyArea / cyImage) ;

    cxImage        = (int) (fScale * cxImage) ;
    cyImage        = (int) (fScale * cyImage) ;

    rect.left      = (cxArea - cxImage) / 2 ;
    rect.right     = (cxArea + cxImage) / 2 ;
    rect.top       = (cyArea - cyImage) / 2 ;
    rect.bottom    = (cyArea + cyImage) / 2 ;
    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
}

```

EMF10 伸展直尺图像以适应显示区域（或列印页面的可列印部分），但不会失真。通常直尺会伸展到显示区域的整个宽度，但是会上下居中对齐。如果您把视窗拉得太小，则直尺会与显示区域一般高，但是会水平居中对齐。

可能有许多种方法来计算合适的显示矩形，但是我们只根据 EMF9 的方式完成该项工作。EMF10.C 中的 PaintRoutine 函式开始部分与 EMF9.C 相同，为的装置内容计算 6 英寸长的直尺图像适当的图素大小。

然後，程式计算名为 fScale 的浮点值，它是显示区域宽度与图像宽度的比值以及显示区域高度与图像高度比值两者的最小值。这个因数在计算围绕矩形前用於增加图像的图素大小。

metafile 中的映射方式

前面绘制的直尺单位有英寸，也有毫米。这种工作使用 GDI 提供的各种映射方式似乎非常适合。但是我坚持使用图素，并「手工」完成所有必要的计算。为什么呢？

答案很简单，就是将映射方式与 metafile 一起使用会十分混乱。我们不妨

实验一下。

当使用 metafile 装置内容呼叫 SetMapMode 时, 该函式在 metafile 中像其他 GDI 函式一样被编码。如程式 18-13 EMF11 显示的那样。

程式 18-13 EMF11

```
EMF11.C
/*-----
-
    EMF11.C -- Enhanced Metafile Demo #11
                                           (c) Charles Petzold, 1998
-----*/

#include <windows.h>
TCHAR szClass      [] = TEXT ("EMF11") ;
TCHAR szTitle      [] = TEXT ("EMF11: Enhanced Metafile Demo #11") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
    int          i, iHeight ;
    LOGFONT      lf ;
    TCHAR        ch ;

    // Black pen with 1-point width

    SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
    // Rectangle surrounding entire pen (with adjustment)
    if (GetVersion () & 0x80000000) // Windows 98
        Rectangle (hdc, 0, -2, cx + 2, cy) ;
    else
// Windows NT
        Rectangle (hdc, 0, -1, cx + 1, cy) ;

    // Tick marks

    for (i = 1 ; i < 96 ; i++)
    {
        if (i % 16 == 0) iHeight = cy / 2 ; // inches
        else if (i % 8 == 0) iHeight = cy / 3 ; // half inches
        else if (i % 4 == 0) iHeight = cy / 5 ; // quarter inches
        else if (i % 2 == 0) iHeight = cy / 8 ; // eighths
        else iHeight = cy / 12 ; // sixteenths

        MoveToEx (hdc, i * cx / 96, 0, NULL) ;
        LineTo (hdc, i * cx / 96, iHeight) ;
    }

    // Create logical font
    FillMemory (&lf, sizeof (lf), 0) ;
    lf.lfHeight = cy / 2 ;
}
```

```
lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

SelectObject      (hdc, CreateFontIndirect (&lf)) ;
SetTextAlign      (hdc, TA_BOTTOM | TA_CENTER) ;
SetBkMode         (hdc, TRANSPARENT) ;

        // Display numbers

for (i = 1 ; i <= 5 ; i++)
{
    ch = (TCHAR) (i + '0') ;
    TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
}

        // Clean up
DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
    HDC          hdcEMF ;
    HENHMETAFILE hemf ;

    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf11.emf"), NULL,
                                TEXT ("EMF11\0EMF Demo #11\0")) ;
    SetMapMode (hdcEMF, MM_LOENGLISH) ;
    DrawRuler (hdcEMF, 600, 100) ;
    hemf = CloseEnhMetaFile (hdcEMF) ;
    DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER      emh ;
    HENHMETAFILE        hemf ;
    int                 cxMms, cyMms, cxPix, cyPix, cxImage, cyImage ;
    RECT                rect ;

    cxMms                = GetDeviceCaps (hdc, HORZSIZE) ;
    cyMms                = GetDeviceCaps (hdc, VERTSIZE) ;
    cxPix                = GetDeviceCaps (hdc, HORZRES) ;
    cyPix                = GetDeviceCaps (hdc, VERTRES) ;

    hemf = GetEnhMetaFile (TEXT ("emf11.emf")) ;
    GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
    cxImage              = emh.rclFrame.right - emh.rclFrame.left ;
    cyImage              = emh.rclFrame.bottom - emh.rclFrame.top ;
```

```

cxImage      = cxImage * cxPix / cxMms / 100 ;
cyImage      = cyImage * cyPix / cyMms / 100 ;

rect.left     = (cxArea - cxImage) / 2 ;
rect.top      = (cyArea - cyImage) / 2 ;
rect.right    = (cxArea + cxImage) / 2 ;
rect.bottom   = (cyArea + cyImage) / 2 ;

PlayEnhMetaFile (hdc, hemf, &rect) ;
DeleteEnhMetaFile (hemf) ;
}

```

EMF11 中的 CreateRoutine 函式比 EMF8 (最初的直尺 metafile 程式) 中的那个简单, 因为它不需要呼叫 GetDeviceCaps 来确定以每英寸点数为单位的显示器解析度。相反, EMF11 呼叫 SetMapMode 将映射方式设定为 MM_LOENGLISH, 其逻辑单位等於 0.01 英寸。因而, 直尺的大小为 600 100 个单位, 并将这些数值传递给 DrawRuler。

除了 MoveToEx 和 LineTo 呼叫绘制直尺的刻度外, EMF11 中的 DrawRuler 函式与 EMF9 中的一样。当以图素单位绘制时 (内定的 MM_TEXT 映射方式), 垂直轴上的单位沿著萤幕向下增长。对於 MM_LOENGLISH 映射方式 (以及其他度量映射方式), 则向上增长。这就需要修改程式码。同时, 也需要更改 Rectangle 函式中的调节因数。

EMF11 中的 PaintRoutine 函式基本上与 EMF9 中的相同, 那个版本的程式能在显示器和印表机上以正确尺寸显示直尺。唯一不同之处在於 EMF11 使用 EMF11.EMF 档案, 而 EMF9 使用 EMF8 建立的 EMF8.EMF 档案。

EMF11 显示的图像基本上与 EMF9 所显示的相同。因此, 在这里可以看到将 SetMapMode 呼叫嵌入 metafile 能够简化 metafile 的建立, 而且不影响以其正确大小显示 metafile 的机制。

映射与显示

在 EMF11 中计算目的矩形包括对 GetDeviceCaps 的几个呼叫。我们的第二个目的是使用映射方式代替这些呼叫。GDI 将目的矩形的座标视为逻辑座标。为这些座标使用 MM_HIMETRIC 似乎是个好方案, 因为它使用 0.01 毫米作为逻辑单位, 与增强型 metafile 表头中用於围绕矩形的单位相同。

程式 18-14 中所示的 EMF12 程式, 保留了 EMF8 中使用的 DrawRuler 处理方式, 但是使用 MM_HIMETRIC 映射方式显示 metafile。

程式 18-14 EMF12

```
EMF12.C
```

```
/*-----
```

```

-
    EMF12.C -- Enhanced Metafile Demo #12
                                   (c) Charles Petzold, 1998
-----*
/

#include <windows.h>
TCHAR szClass      [] = TEXT ("EMF12") ;
TCHAR szTitle      [] = TEXT ("EMF12: Enhanced Metafile Demo #12") ;
void DrawRuler (HDC hdc, int cx, int cy)
{
    int                iAdj, i, iHeight ;
    LOGFONT            lf ;
    TCHAR              ch ;

    iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
                // Black pen with 1-point width
    SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
                // Rectangle surrounding entire pen (with adjustment)
    Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
                // Tick marks
    for (i = 1 ; i < 96 ; i++)
    {
        if (i % 16 == 0)      iHeight = cy / 2 ;           // inches
        else if (i % 8 == 0)   iHeight = cy / 3 ;          // half inches
        else if (i % 4 == 0)   iHeight = cy / 5 ;          // quarter inches
        else if (i % 2 == 0)   iHeight = cy / 8 ;          // eighths
        else iHeight = cy / 12 ;           // sixteenths

        MoveToEx (hdc, i * cx / 96, cy, NULL) ;
        LineTo   (hdc, i * cx / 96, cy - iHeight) ;
    }

                // Create logical font
    FillMemory (&lf, sizeof (lf), 0) ;
    lf.lfHeight = cy / 2 ;
    lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

    SelectObject (hdc, CreateFontIndirect (&lf)) ;
    SetTextAlign (hdc, TA_BOTTOM | TA_CENTER) ;
    SetBkMode    (hdc, TRANSPARENT) ;

                // Display numbers

    for (i = 1 ; i <= 5 ; i++)
    {
        ch = (TCHAR) (i + '0') ;
        TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
    }
}

```

```

        // Clean up
        DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
        DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
    }

void CreateRoutine (HWND hwnd)
{
    HDC                hdcEMF ;
    HENHMETAFILE       hemf ;
    int                cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf12.emf"), NULL,
                                TEXT ("EMF13\0EMF Demo #12\0")) ;

    cxMms = GetDeviceCaps (hdcEMF, HORZSIZE) ;
    cyMms = GetDeviceCaps (hdcEMF, VERTSIZE) ;
    cxPix = GetDeviceCaps (hdcEMF, HORZRES) ;
    cyPix = GetDeviceCaps (hdcEMF, VERTRES) ;

    xDpi = cxPix * 254 / cxMms / 10 ;
    yDpi = cyPix * 254 / cyMms / 10 ;

    DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
    hemf = CloseEnhMetaFile (hdcEMF) ;
    DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER       emh ;
    HENHMETAFILE       hemf ;
    POINT                pt ;
    int                cxImage, cyImage ;
    RECT                rect ;

    SetMapMode (hdc, MM_HIMETRIC) ;
    SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
    pt.x = cxArea ;
    pt.y = 0 ;

    DPTtoLP (hdc, &pt, 1) ;
    hemf = GetEnhMetaFile (TEXT ("emf12.emf")) ;
    GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
    cxImage = emh.rclFrame.right - emh.rclFrame.left ;
    cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

    rect.left = (pt.x - cxImage) / 2 ;
    rect.top = (pt.y + cyImage) / 2 ;

```



```

    rect.right          = (pt.x + cxImage) / 2 ;
    rect.bottom         = (pt.y - cyImage) / 2 ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
}

```

EMF12 中的 PaintRoutine 函式首先将映射方式设定为 MM_HIMETRIC。像其他度量映射方式一样，y 值沿著萤幕向上增长。然而，原点座标仍在萤幕的左上角，这就意味显示区域内的 y 座标值是负数。为了纠正这个问题，程式呼叫 SetViewportOrgEx 将原点座标设定在左下角。

装置座标(cxArea, 0)位於萤幕的右上角。把该座标点传递给 DPtoLP (「装置座标点到逻辑座标点」) 函式，得到以 0.01 毫米为单位的显示区域大小。

然後，程式载入 metafile，取得档案表头，并找到以 0.01 毫米为单位的 metafile 大小。这样计算目的矩形在显示区域居中对齐的位置就变得十分简单。

现在我们看到了在建立 metafile 时能够使用映射方式，显示它时也能使用映射方式。我们能一起完成它们吗？

如程式 18-15 EMF13 展示的那样，这是可以的。

程式 18-15 EMF13

```

EMF13.C
/*-----
-
    EMF13.C -- Enhanced Metafile Demo #13
                        (c) Charles Petzold, 1998
-----*/

#include <windows.h>
TCHAR szClass      [] = TEXT ("EMF13") ;
TCHAR szTitle      [] = TEXT ("EMF13: Enhanced Metafile Demo #13") ;

void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
    ENHMETAHEADER      emh ;
    HENHMETAFILE       hemf ;
    POINT              pt ;
    int                cxImage, cyImage ;
    RECT               rect ;

    SetMapMode (hdc, MM_HIMETRIC) ;

```

```
SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
pt.x = cxArea ;
pt.y = 0 ;

DPToLP (hdc, &pt, 1) ;

hemf = GetEnhMetaFile (TEXT ("..\emf11\emf11.emf")) ;

GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

cxImage      = emh.rclFrame.right - emh.rclFrame.left ;
cyImage      = emh.rclFrame.bottom - emh.rclFrame.top ;

rect.left      = (pt.x - cxImage) / 2 ;
rect.top       = (pt.y + cyImage) / 2 ;
rect.right     = (pt.x + cxImage) / 2 ;
rect.bottom    = (pt.y - cyImage) / 2 ;

PlayEnhMetaFile (hdc, hemf, &rect) ;
DeleteEnhMetaFile (hemf) ;
}
```

在 EMF13 中, 由於直尺 metafile 已由 EMF11 建立, 所以它没有使用映射方式建立 metafile。EMF13 只是简单地载入 metafile, 然後像 EMF11 一样使用映射方式计算目的矩形。

现在, 我们可以建立一些规则。在建立 metafile 时, GDI 使用对映射方式的任意嵌入修改, 来计算以图素和毫米为单位的 metafile 图像的大小。图像的大小储存在 metafile 表头内。在显示 metafile 时, GDI 在呼叫 PlayEnhMetaFile 时根据有效的映射方式建立目的矩形的实际位置, 而本来的 metafile 中并没有任何记录去更改这个位置。

第十九章 多重文件介面

多重文件介面 (MDI) 是 Microsoft Windows 文件处理應用程式的一种规范, 该规范描述了视窗结构和允许使用者在单个應用程式中使用多个文件的使用者介面 (如文书处理程式中的文字文件和試算表程式中的試算表)。简单地说, 就像 Windows 在一个萤幕上维护多个應用程式视窗一样, MDI 應用程式在一个显示区域内维护多个文件视窗。Windows 中的第一个 MDI 應用程式是 Windows 下的 Microsoft Excel 的第一个版本。紧接著又出现了许多其他的應用程式。

MDI 概念

尽管 MDI 规范随著 Windows 2.0 的推出已经很普及, 但在那时, MDI 應用程式写起来很困难, 并且需要一些非常复杂的程式设计工作。从 Windows 3.0 起, 其中许多工作就都由 Windows 为您做好了。Windows 95 中增强的支援也已经被添加进 Windows 98 和 Microsoft Windows NT 中。

MDI 的组成

MDI 程式的主應用程式视窗是很普通的: 它有一个标题列、一个功能表、一个缩放边框、一个系统功能表图示和最大化/最小化/关闭按钮。显示区域经常被称为「工作空间」, 它不直接用於显示程式输出。这个工作空间包括零个或多个子视窗, 每个视窗都显示一个文件。

这些子视窗看起来与通常的應用程式视窗以及 MDI 程式的主视窗很相似。它们有一个标题列、一个缩放边框、一个系统功能表图示和最大化/最小化/关闭按钮, 可能还包括卷动列。但是文件视窗没有功能表, 主應用程式视窗上的功能表适用於文件视窗。

在任何时候都只能有一个文件视窗是活动的 (加亮标题列来表示), 它出现在其他所有文件视窗之前。所有文件视窗都由工作空间区域加以剪裁, 而不会出现在應用程式视窗之外。

初看起来, 对 Windows 程式写作者来说, MDI 似乎是相当简单。需要程式写作者做的工作好像就是为每个文件建立一个 WS_CHILD 视窗, 并使程式的主應用程式视窗成为文件视窗的父视窗。但对现有的 MDI 應用程式稍加研究, 就会发现一些导致程式写作困难的复杂问题。例如:

- MDI 文件视窗可以最小化。它的图示出现在工作空间的底部。一般来说, MDI 應用程式可以将不同的图示分别用於主應用程式视窗和每一类文件

应用。

- MDI 文件视窗可以最大化。在这种情况下，文件视窗的标题列（一般用来显示视窗中文件的档案名称）消失，档案名称出现在應用程式视窗标题列的應用程式名称之後，文件视窗的系统功能表图示成为應用程式视窗的顶层功能表中的第一项。关闭文件视窗按钮变成顶层功能表中的最後一项，且出现在最右边。
- 用以关闭文件视窗的系统键盘加速键与关闭主视窗的系统键盘加速键一样，只是 Ctrl 键代替了 Alt 键。这也就是说，Alt+F4 用於关闭應用程式视窗，而 Ctrl+F4 用於关闭文件视窗。此外，Ctrl+F6 可以在活动 MDI 應用程式的子文件视窗之间切换。与平时一样，Alt+空白键启动主视窗的系统功能表，Alt+-（减号）启动活动子文件视窗的系统功能表。
- 当使用游标键在功能表项间移动时，控制项权通常从系统功能表转到功能表列中的第一项。在 MDI 應用程式中，控制项权是从應用程式系统功能表转到活动文件系统功能表，然後再转到功能表列中的第一项。
- 如果應用程式能够支援若干种型态的子视窗（如 Microsoft Excel 中的工作表和图表文件），那么功能表应能反映出与这种型态的文件有关的操作。这就要求当不同的文字视窗变成活动视窗时，程式能更换功能表。此外，当没有文件视窗存在时，功能表应该被缩减到只剩下与打开新文件有关的操作。
- 顶层功能表上有一个叫做「视窗 (Window)」的功能表项。按照习惯，这是顶层功能表上「Help」之前的那一项，即倒数第二项。「视窗」子功能表上通常包含在工作空间内安排文件视窗的选项。文件视窗可以从左上方开始「平铺」或「层叠」。在前一种方式下，可以完整地看到每一个文件视窗。这个子功能表同时也包含所有文件视窗的列表。从中选择一个文件视窗，就可以把此文件视窗移到前景。

Windows 98 支援 MDI 的所有这些方面。当然，需要您做一些工作（如下面的范例程式所示），但是，这远不是要您程式写作来直接支援所有这些功能。

MDI 支援

探讨 Windows 的 MDI 支援时需要发表一些新术语。主應用程式视窗称为「框架视窗」，就像传统的 Windows 程式一样，它是 WS_OVERLAPPEDWINDOW 样式的视窗。

MDI 應用程式还根据预先定义的视窗类别 MDICLIENT 建立「客户视窗」，这一客户视窗是用这种视窗类别和 WS_CHILD 样式呼叫 CreateWindow 来建立的。

这一呼叫的最後一个参数是指向一个 CLIENTCREATESTRUCT 型态的结构的指标。这个客户视窗覆盖框架视窗的显示区域，并提供许多 MDI 支援。此客户视窗的颜色是系统颜色 COLOR_APPWORKSPACE。

文件视窗被称为「子视窗」。通过初始化一个 MDICREATESTRUCT 型态的结构，以一个指向此结构的指标为参数将讯息 WM_MDICREATE 发送给客户视窗，就可以建立这些文件视窗。

文件视窗是客户视窗的子视窗，而客户视窗又是框架视窗的子视窗。父-子视窗分层结构如图 19-1 所示。

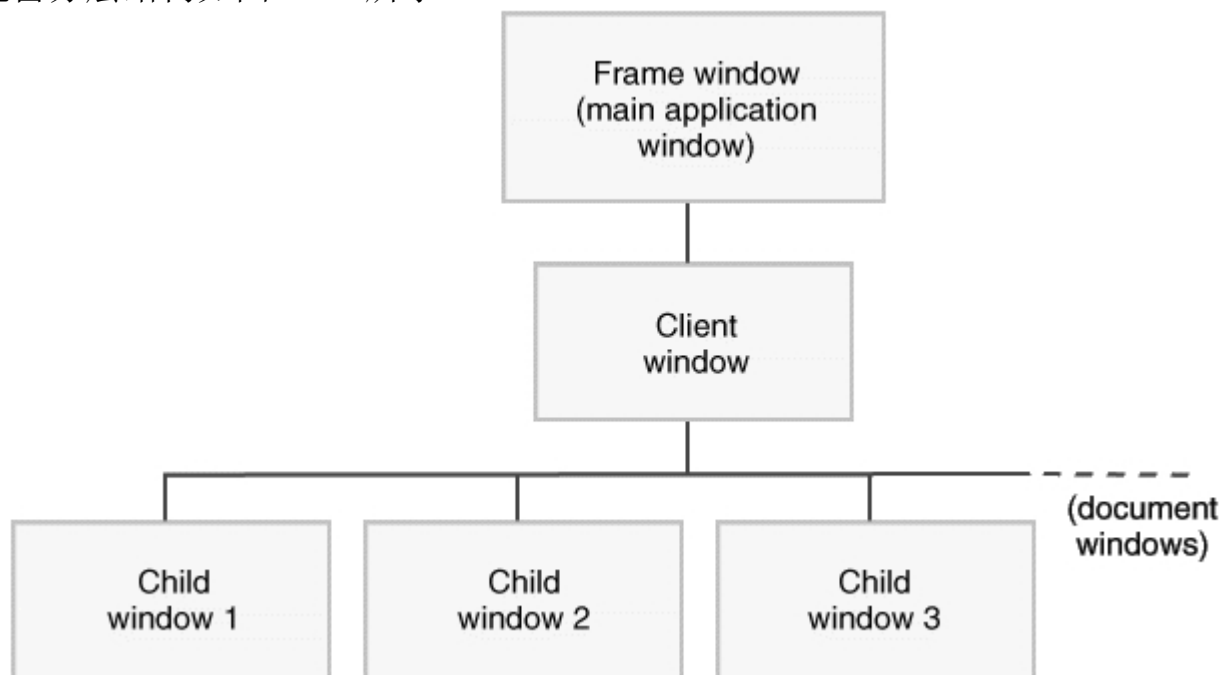


图 19-1 Windows MDI 应用程式的父-子层次图

您需要框架视窗的视窗类别（及视窗讯息处理程式）和一个由应用程式支援的每类子视窗的视窗类别（及视窗讯息处理程式）。由於已经预先注册了视窗类别，所以不需要客户视窗的视窗讯息处理程式。

Windows 98 的 MDI 支援包括一个视窗类别、五个函式、两个资料结构和 12 个讯息。前面已经提到了 MDI 视窗类别，即 MDICLIENT，以及资料结构 CLIENTCREATESTRUCT 和 MDICREATESTRUCT。在 MDI 应用程式中，这五个函式中的两个用於取代 DefWindowProc：不再将 DefWindowProc 呼叫用於所有未处理的讯息，而是由框架视窗程序呼叫 DefFrameProc，子视窗程序呼叫 DefMDIChildProc。另一个 MDI 特有的函式 TranslateMDISysAccel 与第十章中讨论的 TranslateAccelerator 的使用方式相同。MDI 支援也包括 ArrangeIconicWindows 函式，但有一条专用的 MDI 讯息使得此函式对 MDI 程式来说不再必要。

第五个 MDI 函式是 CreateMDIWindow，它使得子视窗可以在单独的执行绪中

被建立。这个函式不需要在单执行绪的程式中，我会展示这一点。

在下面的程式中，我将展示 12 条 MDI 讯息中的 9 条（其他三个讯息一般不用），这些讯息的字首是 WM_MDI。框架视窗向客户视窗发送其中某个讯息，以便在子视窗上完成一项操作或者取得关于子视窗的资讯（例如，框架视窗发送一个 WM_MDICREATE 讯息给客户视窗，以建立子视窗）。讯息 WM_MDIACTIVATE 讯息有点特别：框架视窗可以发送这个讯息给客户视窗来启动一个子视窗，而客户视窗也把这个讯息发送给将被启动或者失去活动的子视窗，以便通知它们这一变化。

MDI 的范例程式

程式 19-1 MDIDEMO 程式说明了编写 MDI 應用程式的基本方法。

程式 19-1 MDIDEMO

```
MDIDEMO.C
/*-----
-
MDIDEMO.C -- Multiple-Document Interface Demonstration
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

#define INIT_MENU_POS          0
#define HELLO_MENU_POS        2
#define RECT_MENU_POS         1

#define IDM_FIRSTCHILD         50000

LRESULT      CALLBACK FrameWndProc          (HWND, UINT, WPARAM, LPARAM) ;
BOOL         CALLBACK CloseEnumProc        (HWND, LPARAM) ;
LRESULT      CALLBACK HelloWndProc         (HWND, UINT, WPARAM, LPARAM) ;
LRESULT      CALLBACK RectWndProc          (HWND, UINT, WPARAM, LPARAM) ;

// structure for storing data unique to each Hello child window

typedef struct tagHELLODATA
{
    UINT          iColor ;
    COLORREF      clrText ;
}
HELLODATA, * PHELLODATA ;

// structure for storing data unique to each Rect child window
```

```

typedef struct tagRECTDATA
{
    short          cxClient ;
    short          cyClient ;
}
RECTDATA, * PRECTDATA ;
    // global variables
TCHAR            szAppName[]          = TEXT ("MDIDemo") ;
TCHAR            szFrameClass[]        = TEXT ("MdiFrame") ;
TCHAR            szHelloClass[]        = TEXT ("MdiHelloChild") ;
TCHAR            szRectClass[]         = TEXT ("MdiRectChild") ;
HINSTANCE        hInst ;
HMENU            hMenuInit, hMenuHello, hMenuRect ;
HMENU            hMenuInitWindow,      hMenuHelloWindow,
hMenuRectWindow ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HACCEL          hAccel ;
    HWND            hwndFrame, hwndClient ;
    MSG             msg ;
    WNDCLASS        wndclass ;

    hInst = hInstance ;
    // Register the frame window class
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = FrameWndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon           = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) (COLOR_APPWORKSPACE + 1) ;
    wndclass.lpszMenuName     = NULL ;
    wndclass.lpszClassName   = szFrameClass ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    // Register the Hello child window class
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = HelloWndProc ;

```

```

wndclass.cbClsExtra           = 0 ;
wndclass.cbWndExtra           = sizeof (HANDLE) ;
wndclass.hInstance           = hInstance ;
wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName          = NULL ;
wndclass.lpszClassName         = szHelloClass ;

RegisterClass (&wndclass) ;
        // Register the Rect child window class
wndclass.style                 = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc           = RectWndProc ;
wndclass.cbClsExtra           = 0 ;
wndclass.cbWndExtra           = sizeof (HANDLE) ;
wndclass.hInstance           = hInstance ;
wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName          = NULL ;
wndclass.lpszClassName         = szRectClass ;

RegisterClass (&wndclass) ;
        // Obtain handles to three possible menus & submenus
hMenuInit                     = LoadMenu (hInstance, TEXT ("MdiMenuInit")) ;
hMenuHello                    = LoadMenu (hInstance, TEXT ("MdiMenuHello")) ;
hMenuRect                     = LoadMenu (hInstance, TEXT ("MdiMenuRect")) ;

hMenuInitWindow                = GetSubMenu      (hMenuInit,   INIT_MENU_POS) ;
hMenuHelloWindow               = GetSubMenu      (hMenuHello,   HELLO_MENU_POS)
;
hMenuRectWindow                = GetSubMenu      (hMenuRect,    RECT_MENU_POS) ;

        // Load accelerator table

hAccel = LoadAccelerators (hInstance, szAppName) ;
        // Create the frame window
hwndFrame = CreateWindow (szFrameClass, TEXT ("MDI Demonstration"),
        WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, hMenuInit, hInstance, NULL) ;
hwndClient = GetWindow (hwndFrame, GW_CHILD) ;
ShowWindow (hwndFrame, iCmdShow) ;
UpdateWindow (hwndFrame) ;

        // Enter the modified message loop
while (GetMessage (&msg, NULL, 0, 0))

```



```

{
    if ( !TranslateMDISysAccel (hwndClient, &msg) &&
        !TranslateAccelerator (hwndFrame, hAccel, &msg))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

// Clean up by deleting unattached menus
DestroyMenu (hMenuHello) ;
DestroyMenu (hMenuRect) ;

return msg.wParam ;
}

```

```

LRESULT CALLBACK FrameWndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)

```

```

{
    static HWND          hwndClient ;
    CLIENTCREATESTRUCT    clientcreate ;
    HWND                hwndChild ;
    MDICREATESTRUCT       mdicreate ;

    switch (message)
    {
    case WM_CREATE:                // Create the client window

        clientcreate.hWindowMenu    = hMenuInitWindow ;
        clientcreate.idFirstChild    = IDM_FIRSTCHILD ;

        hwndClient = CreateWindow (    TEXT ("MDICLIENT"), NULL,
                                      WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
                                      0, 0, 0, 0, hwnd, (HMENU) 1, hInst,
                                      (PSTR) &clientcreate) ;

        return 0 ;
    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDM_FILE_NEWHELLO: // Create a Hello child window
            mdicreate.szClass      = szHelloClass ;
            mdicreate.szTitle       = TEXT ("Hello") ;
            mdicreate.hOwner        = hInst ;
            mdicreate.x              = CW_USEDEFAULT ;
            mdicreate.y              = CW_USEDEFAULT ;
            mdicreate.cx             = CW_USEDEFAULT ;
            mdicreate.cy             = CW_USEDEFAULT ;
            mdicreate.style          = 0 ;
            mdicreate.lParam         = 0 ;

```

```
        hwndChild = (HWND) SendMessage (hwndClient,
WM_MDICREATE, 0, (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
        return 0 ;

    case  IDM_FILE_NEWRECT:          // Create a Rect child window

        mdicreate.szClass            = szRectClass ;
        mdicreate.szTitle            = TEXT ("Rectangles") ;
        mdicreate.hOwner              = hInst ;
        mdicreate.x                   = CW_USEDEFAULT ;
        mdicreate.y                   = CW_USEDEFAULT ;
        mdicreate.cx                  = CW_USEDEFAULT ;
        mdicreate.cy                  = CW_USEDEFAULT ;
        mdicreate.style                = 0 ;
        mdicreate.lParam              = 0 ;

        hwndChild = (HWND) SendMessage (hwndClient,
            WM_MDICREATE, 0,
            (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
        return 0 ;

    case  IDM_FILE_CLOSE:            // Close the active window

        hwndChild = (HWND) SendMessage (hwndClient,
            WM_MDIGETACTIVE, 0, 0) ;

        if (SendMessage (hwndChild, WM_QUERYENDSESSION, 0, 0))
            SendMessage (hwndClient, WM_MDIDESTROY,
(WPARAM) hwndChild, 0) ;
        return 0 ;

    case  IDM_APP_EXIT:              // Exit the program

        SendMessage (hwnd, WM_CLOSE, 0, 0) ;
        return 0 ;

    // messages for arranging windows

    case  IDM_WINDOW_TILE:
        SendMessage (hwndClient, WM_MDITILE, 0, 0) ;
        return 0 ;

    case  IDM_WINDOW_CASCADE:
        SendMessage (hwndClient, WM_MDICASCADE, 0, 0) ;
        return 0 ;

    case  IDM_WINDOW_ARRANGE:
```

```

        SendMessage (hwndClient, WM_MDIICONARRANGE, 0, 0) ;
        return 0 ;

    case  IDM_WINDOW_CLOSEALL:  // Attempt to close all
children
        EnumChildWindows (hwndClient, CloseEnumProc, 0) ;
        return 0 ;

    default:                    // Pass to active child...

        hwndChild = (HWND) SendMessage (hwndClient,
        WM_MDIGETACTIVE, 0, 0) ;
        if (IsWindow (hwndChild))
        SendMessage (hwndChild, WM_COMMAND, wParam, lParam) ;

        break ;    // ...and then to DefFrameProc
    }
    break ;

case  WM_QUERYENDSESSION:
case  WM_CLOSE:  // Attempt to close all children

        SendMessage (hwnd, WM_COMMAND, IDM_WINDOW_CLOSEALL, 0) ;

        if (NULL != GetWindow (hwndClient, GW_CHILD))
            return 0 ;

        break ;    // i.e., call DefFrameProc
case  WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
// Pass unprocessed messages to DefFrameProc (not DefWindowProc)

return DefFrameProc (hwnd, hwndClient, message, wParam, lParam) ;
}

BOOL CALLBACK CloseEnumProc (HWND hwnd, LPARAM lParam)
{
    if (GetWindow (hwnd, GW_OWNER)) // Check for icon title
        return TRUE ;

    SendMessage (GetParent (hwnd), WM_MDIESTORE, (LPARAM) hwnd, 0) ;
    if (!SendMessage (hwnd, WM_QUERYENDSESSION, 0, 0))
        return TRUE ;
    SendMessage (GetParent (hwnd), WM_MDIESTROY, (LPARAM) hwnd, 0) ;
    return TRUE ;
}

```

```

}

LRESULT CALLBACK HelloWndProc (HWND hwnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
    static COLORREF clrTextArray[] = {      RGB (0,   0, 0), RGB (255, 0,   0),
      RGB (0, 255, 0), RGB (  0, 0, 255),
      RGB (255, 255, 255) } ;
    static HWND      hwndClient, hwndFrame ;
    HDC            hdc ;
    HMENU          hMenu ;
    PHELLODATA      pHelloData ;
    PAINTSTRUCT     ps ;
    RECT            rect ;

    switch (message)
    {
    case WM_CREATE:
        // Allocate memory for window private data

        pHelloData = (PHELLODATA) HeapAlloc (GetProcessHeap (),
            HEAP_ZERO_MEMORY, sizeof (HELLODATA)) ;
        pHelloData->iColor = IDM_COLOR_BLACK ;
        pHelloData->clrText = RGB (0, 0, 0) ;
        SetWindowLong (hwnd, 0, (long) pHelloData) ;

        // Save some window handles

        hwndClient = GetParent (hwnd) ;
        hwndFrame = GetParent (hwndClient) ;
        return 0 ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_COLOR_BLACK:
            case IDM_COLOR_RED:
            case IDM_COLOR_GREEN:
            case IDM_COLOR_BLUE:
            case IDM_COLOR_WHITE:
                // Change the text color

                pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;

                hMenu = GetMenu (hwndFrame) ;

                CheckMenuItem (hMenu, pHelloData->iColor, MF_UNCHECKED) ;
                pHelloData->iColor = wParam ;

```

```

        CheckMenuItem (hMenu, pHelloData->iColor, MF_CHECKED) ;

        pHelloData->clrText = clrTextArray[wParam - IDM_COLOR_BLACK] ;

        InvalidateRect (hwnd, NULL, FALSE) ;
    }
    return 0 ;

case WM_PAINT:
    // Paint the window

    hdc = BeginPaint (hwnd, &ps) ;

    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    SetTextColor (hdc, pHelloData->clrText) ;

    GetClientRect (hwnd, &rect) ;

    DrawText (hdc, TEXT ("Hello, World!"), -1, &rect,
DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_MDIACTIVATE:
    // Set the Hello menu if gaining focus

    if (lParam == (LPARAM) hwnd)
        SendMessage (hwndClient, WM_MDISETMENU, (WPARAM)
hMenuHello, (LPARAM) hMenuHelloWindow) ;

    // Check or uncheck menu item

    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    CheckMenuItem (hMenuHello, pHelloData->iColor,
        (lParam == (LPARAM) hwnd) ? MF_CHECKED :
MF_UNCHECKED) ;

    // Set the Init menu if losing focus

    if (lParam != (LPARAM) hwnd)
        SendMessage (hwndClient,
WM_MDISETMENU, (WPARAM)
hMenuInit, (LPARAM) hMenuInitWindow) ;

    DrawMenuBar (hwndFrame) ;
    return 0 ;

case WM_QUERYENDSESSION:

```

```

case WM_CLOSE:
    if (IDOK != MessageBox (hwnd, TEXT ("OK to close window?"),
                            TEXT ("Hello"),
                            MB_ICONQUESTION | MB_OKCANCEL))
        return 0 ;

    break ;           // i.e., call DefMDIChildProc

case WM_DESTROY:
    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    HeapFree (GetProcessHeap (), 0, pHelloData) ;
    return 0 ;
}

// Pass unprocessed message to DefMDIChildProc

return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}
LRESULT CALLBACK RectWndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND      hwndClient, hwndFrame ;
    HBRUSH          hBrush ;
    HDC              hdc ;
    PRECTDATA        pRectData ;
    PAINTSTRUCT       ps ;
    int               xLeft, xRight, yTop, yBottom ;
    short             nRed, nGreen, nBlue ;

    switch (message)
    {
    case WM_CREATE:
        // Allocate memory for window private data

        pRectData = (PRECTDATA) HeapAlloc (GetProcessHeap (),
            HEAP_ZERO_MEMORY, sizeof (RECTDATA)) ;

        SetWindowLong (hwnd, 0, (long) pRectData) ;

        // Start the timer going

        SetTimer (hwnd, 1, 250, NULL) ;

        // Save some window
handles

        hwndClient = GetParent (hwnd) ;
        hwndFrame  = GetParent (hwndClient) ;
        return 0 ;
    }
}

```

```
case WM_SIZE:    // If not minimized, save the window size

    if (wParam != SIZE_MINIMIZED)
    {
        pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;

        pRectData->cxClient = LOWORD (lParam) ;
        pRectData->cyClient = HIWORD (lParam) ;
    }

    break;    // WM_SIZE must be processed by DefMDIChildProc

case WM_TIMER:    // Display a random rectangle

    pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;
    xLeft      = rand () % pRectData->cxClient ;
    xRight     = rand () % pRectData->cxClient ;
    yTop       = rand () % pRectData->cyClient ;
    yBottom    = rand () % pRectData->cyClient ;
    nRed       = rand () & 255 ;
    nGreen     = rand () & 255 ;
    nBlue      = rand () & 255 ;

    hdc = GetDC (hwnd) ;
    hBrush = CreateSolidBrush (RGB (nRed, nGreen, nBlue)) ;
    SelectObject (hdc, hBrush) ;

    Rectangle (hdc, min (xLeft, xRight), min (yTop, yBottom),
               max (xLeft, xRight), max (yTop, yBottom)) ;

    ReleaseDC (hwnd, hdc) ;
    DeleteObject (hBrush) ;
    return 0 ;

case WM_PAINT:    // Clear the window

    InvalidateRect (hwnd, NULL, TRUE) ;
    hdc = BeginPaint (hwnd, &ps) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_MDIACTIVATE:    // Set the appropriate menu
    if (lParam == (LPARAM) hwnd)
        SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuRect,
(LPARAM) hMenuRectWindow) ;
    else
        SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuInit,
(LPARAM) hMenuInitWindow) ;
```

```

        DrawMenuBar (hwndFrame) ;
        return 0 ;

case WM_DESTROY:
        pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;
        HeapFree (GetProcessHeap (), 0, pRectData) ;
        KillTimer (hwnd, 1) ;
        return 0 ;
}

// Pass unprocessed message to DefMDIChildProc
return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}

MDIDEMO.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
MDIMENUINIT MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "New &Hello",
        IDM_FILE_NEWHELLO
        MENUITEM "New &Rectangle",
        IDM_FILE_NEWRECT
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_APP_EXIT
    END
END
MDIMENUHELLO MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "New &Hello",      IDM_FILE_NEWHELLO
        MENUITEM "New &Rectangle",  IDM_FILE_NEWRECT
        MENUITEM "&Close",          IDM_FILE_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",            IDM_APP_EXIT
    END
    POPUP "&Color"
    BEGIN
        MENUITEM "&Black",          IDM_COLOR_BLACK
        MENUITEM "&Red",             IDM_COLOR_RED
        MENUITEM "&Green",          IDM_COLOR_GREEN
    END

```



```

        MENUITEM "B&blue",          IDM_COLOR_BLUE
        MENUITEM "&White",          IDM_COLOR_WHITE
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade\tShift+F5",      IDM_WINDOW_CASCADE
        MENUITEM "&Tile\tShift+F4",          IDM_WINDOW_TILE
        MENUITEM "Arrange &Icons",          IDM_WINDOW_ARRANGE
        MENUITEM "Close &All",              IDM_WINDOW_CLOSEALL
    END
END
MDIMENURECT MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "New &Hello",              IDM_FILE_NEWHELLO
        MENUITEM "New &Rectangle",          IDM_FILE_NEWRECT
        MENUITEM "&Close",                  IDM_FILE_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                    IDM_APP_EXIT
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade\tShift+F5",      IDM_WINDOW_CASCADE
        MENUITEM "&Tile\tShift+F4",          IDM_WINDOW_TILE
        MENUITEM "Arrange &Icons",          IDM_WINDOW_ARRANGE
        MENUITEM "Close &All",              IDM_WINDOW_CLOSEALL
    END
END

```

////////////////////////////////////

/

// Accelerator

MDIDEMO ACCELERATORS DISCARDABLE

BEGIN

```

    VK_F4,    IDM_WINDOW_TILE,    VIRTKEY, SHIFT, NOINVERT
    VK_F5,    IDM_WINDOW_CASCADE, VIRTKEY, SHIFT, NOINVERT

```

END

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by MDIDemo.rc

```

```

#define IDM_FILE_NEWHELLO          40001
#define IDM_FILE_NEWRECT          40002
#define IDM_APP_EXIT               40003
#define IDM_FILE_CLOSE            40004
#define IDM_COLOR_BLACK           40005
#define IDM_COLOR_RED             40006

```

```

#define IDM_COLOR_GREEN 40007
#define IDM_COLOR_BLUE 40008
#define IDM_COLOR_WHITE 40009
#define IDM_WINDOW_CASCADE 40010
#define IDM_WINDOW_TILE 40011
#define IDM_WINDOW_ARRANGE 40012
#define IDM_WINDOW_CLOSEALL 40013

```

MDIDEMO 支援两种型态的非常简单的文件视窗：第一种视窗在它的显示区域中央显示“Hello, World!”，另一种视窗显示一系列随机矩形（在原始码列表和识别字名中，它们分别叫做「Hello」文件和「Rect」文件）。这两类文件视窗的功能表不同，显示“Hello, World!”的文件视窗有一个允许使用者修改文字颜色的功能表。

三个功能表

现在让我们先看看 MDIDEMO.RC 资源描述档，它定义了程式所使用的三个功能表模板。

当文件视窗不存在时，程式显示 MdiMenuInit 功能表，这个功能表只允许使用者建立新文件或退出程式。

MdiMenuHello 功能表与显示「Hello, World!」的文件视窗相关联。「File」子功能表允许使用者打开任何一类新文件、关闭活动文件或退出程式。「Color」子功能表允许使用者设定文字颜色。Window 子功能表包括以平铺或者重叠的方式安排文件视窗、安排文件图示或关闭所有视窗等选项，这个子功能表也列出了它们建立的所有文件视窗。

MdiMenuRect 功能表与随机矩形文件相关联。除了不包含「Color」子功能表外，它与 MdiMenuHello 功能表一样。

RESOURCE.H 表头档案定义所有的功能表识别字。另外，以下三个常数定义在 MDIDEMO.C 中：

```

#define INIT_MENU_POS 0
#define HELLO_MENU_POS 2
#define RECT_MENU_POS 1

```

这些识别字说明每个功能表模板中 Windows 子功能表的位置。程式需要这些资讯来通知客户视窗文件列表应出现在哪里。当然，MdiMenuInit 功能表没有 Windows 子功能表，所以如前所述，文件列表应附加在第一个子功能表中（位置 0）。不过，实际上永远不会在此看到文件列表（在后面讨论此程式时，您可以发现这样做的原因）。

定义在 MDIDEMO.C 中的 IDM_FIRSTCHILD 识别字不对应于功能表项，它与出现在 Windows 子功能表上的文件列表中的第一个文件视窗相关联。这个识别字

的值应当大於所有其他功能表 ID 的值。

程式初始化

在 MDIDEMO.C 中, WinMain 是从注册框架视窗和两个子视窗的视窗类别开始的。视窗讯息处理程式是 FrameWndProc、HelloWndProc 和 RectWndProc。一般来说, 这些视窗类别应该与不同的图示相关联。为了简单起见, 我们将标准 IDI_APPLICATION 图示用於框架视窗和子视窗。

注意, 我们已经定义了框架视窗类别的 WNDCLASS 结构的 hbrBackground 栏位为 COLOR_APPWORKSPACE 系统颜色。由於框架视窗的显示区域被客户视窗所覆盖并且客户视窗具有这种颜色, 所以上面的定义不是绝对必要的。但是, 在最初显示框架视窗时, 使用这种颜色似乎要好一些。

这三种视窗类别中的 lpszMenuName 栏位都设定为 NULL。对「Hello」和「Rect」子视窗类别来说, 这是很自然的。对於框架视窗类别, 我在建立框架视窗时在 CreateWindow 函式中给出功能表代号。

「Hello」和「Rect」子视窗的视窗类别将 WNDCLASS 结构中的 cbWndExtra 栏位设为非零值来为每个视窗配置额外空间, 这个空间将用於储存指向一个记忆体块的指标(HELLODATA 和 RECTDATA 结构的大小定义在 MDIDEMO.C 的开始处), 这个记忆体块被用於储存每个文件视窗特有的资讯。

下一步, WinMain 用 LoadMenu 载入三个功能表, 并把它们的代号储存到整体变数中。呼叫三次 GetSubMenu 函式可获得 Windows 子功能表(文件列表将加在它上面)的代号, 同样也把它们储存到整体变数中。LoadAccelerators 函式载入加速键表。

在 WinMain 中呼叫 CreateWindow 建立框架视窗。在 FrameWndProc 中 WM_CREATE 讯息处理期间, 框架视窗建立客户视窗。这项操作涉及到再一次呼叫函式 CreateWindow。视窗类别被设定为 MDICLIENT, 它是预先注册的 MDI 显示区域视窗类别。在 Windows 中许多对 MDI 的支援被放入了 MDICLIENT 视窗类别中。显示区域视窗讯息处理程式作为框架视窗和不同文件视窗的中间层。当呼叫 CreateWindow 建立显示区域视窗时, 最後一个参数必须被设定为指向 CLIENTCREATESTRUCT 型态结构的指标。这个结构有两个栏位:

- hWindowMenu 是要加入文件列表的子功能表的代号。在 MDIDEMO 中, 它是 hMenuInitWindow, 是在 WinMain 期间获得的。後面将看到如何修改此功能表。
- idFirstChild 是与文件列表中的第一个文件视窗相关联的功能表 ID。它就是 IDM_FIRSTCHILD。

再让我们回过头来看看 WinMain。MDIDEMO 显示新建立的框架视窗并进入讯息回圈。讯息回圈与正常的回圈稍有不同：在呼叫 GetMessage 从讯息伫列中获得讯息之後，MDI 程式把这个讯息传送给了 TranslateMDISysAccel（以及 TranslateAccelerator，如果像 MDIDEMO 程式一样，程式本身也有功能表加速键的话）。

TranslateMDISysAccel 函式把可能对应特定 MDI 加速键（例如 Ctrl-F6）的按键转换成 WM_SYSCOMMAND 讯息。如果 TranslateMDISysAccel 或 TranslateAccelerator 都传回 TRUE（表示某个讯息已被这些函式之一转换），就不能呼叫 TranslateMessage 和 DispatchMessage。

注意传递到 TranslateMDISysAccel 和 TranslateAccelerator 的两个视窗代号：hwndClient 和 hwndFrame。WinMain 函式通过用 GW_CHILD 参数呼叫 GetWindow 获得 hwndClient 视窗代号。

建立子视窗

FrameWndProc 的大部分工作是用於处理通知功能表选择的 WM_COMMAND 讯息。与平时一样，FrameWndProc 中 wParam 参数的低字组包含著功能表 ID。

在功能表 ID 的值为 IDM_FILE_NEWHELLO 和 IDM_FILE_NEWRECT 的情况下，FrameWndProc 必须建立一个新的文件视窗。这涉及到初始化 MDICREATESTRUCT 结构中的栏位（大多数栏位对应於 CreateWindow 的参数），并将讯息 WM_MDICREATE 发送给客户视窗，讯息的 lParam 参数设定为指向这个结构的指标。然後由客户视窗建立子文件视窗。（也可以使用 CreateMDIWindow 函式。）

MDICREATESTRUCT 结构中的 szTitle 栏位一般是对应於文件的档案名称。样式栏位设定为视窗样式 WS_HSCROLL、WS_VSCROLL 或这两者的组合，以便在文件视窗中包括卷动列。样式栏位也可以包括 WS_MINIMIZE 或 WS_MAXIMIZE，以便在最初时以最小化或最大化状态显示文件视窗。

MDICREATESTRUCT 结构的 lParam 栏位为框架视窗和子视窗共用某些变数提供了一种方法。这个栏位可以设定为含有一个结构的记忆体块的记忆体代号。在子文件视窗的 WM_CREATE 讯息处理期间，lParam 是一个指向 CREATESTRUCT 结构的指标，这个结构的 lpCreateParams 栏位是一个指向用於建立视窗的 MDICREATESTRUCT 结构的指标。

客户视窗一旦接收到 WM_MDICREATE 讯息就建立一个子文件视窗，并把视窗标题加到用於建立客户视窗的 MDICLIENTSTRUCT 结构中所指定的子功能表的底部。当 MDIDEMO 程式建立它的第一个文件视窗时，这个子功能表就是「MdiMenuInit」功能表中的「File」子功能表。後面将看到这个文件列表将如

何移到「MdiMenuHello」和「MdiMenuRect」功能表的「Windows」子功能表中。

功能表上可以列出 9 个文件，每个文件的前面是带有底线的数字 1 至 9。如果建立的文件视窗多於 9 个，则这个清单後跟有「More Windows」功能表项。该项启动带有清单方块的对话方块，清单方块列出了所有文件。这种文件列表的维护是 Windows MDI 支援的最好特性之一。

关于框架视窗的讯息处理

在把注意力转移到子文件视窗之前，我们先继续讨论 FrameWndProc 的讯息处理。

当从「File」功能表中选择「Close」时，MDIDEMO 关闭活动子视窗。它通过把 WM_MDIGETACTIVE 讯息发送给客户视窗，而获得活动子视窗的代号。如果子视窗以 WM_QUERYENDSESSION 讯息来回应，那么 MDIDEMO 将 WM_MDIDESTROY 讯息发送给客户视窗，从而关闭子视窗。

处理「File」功能表中的「Exit」选项只需要框架视窗讯息处理程式给自己发送一个 WM_CLOSE 讯息。

处理 Window 子功能表的「Tile」、「Cascade」和「Arrange」选项是很容易的，只需把讯息 WM_MDITILE、WM_MDICASCADE 和 WM_MDIICONARRANGE 发送给客户视窗。

处理「Close All」选项要稍微复杂一些。FrameWndProc 呼叫 EnumChildWindows，传送一个引用 CloseEnumProc 函式的指标。此函式把 WM_MDIESTORE 讯息发送给每个子视窗，紧跟著发出 WM_QUERYENDSESSION 和 WM_MDIDESTROY。对图示平铺视窗来说并不就此结束，用 GW_OWNER 参数呼叫 GetWindow 时，传回的非 NULL 值可以显示出这一点。

FrameWndProc 没有处理任何由「Color」功能表中对颜色的选择所导致的 WM_COMMAND 讯息，这些讯息应该由文件视窗负责处理。因此，FrameWndProc 把所有未经处理的 WM_COMMAND 讯息发送到活动子视窗，以便子视窗可以处理那些与它们有关的讯息。

框架视窗讯息处理程式不予处理的所有讯息都要送到 DefFrameProc，它在框架视窗讯息处理程式中取代了 DefWindowProc。即使框架视窗讯息处理程式拦截了 WM_MENUCHAR、WM_SETFOCUS 或 WM_SIZE 讯息，这些讯息也要被送到 DefFrameProc 中。

所有未经处理的 WM_COMMAND 讯息也必须送给 DefFrameProc。具体地说，FrameWndProc 并不处理任何 WM_COMMAND 讯息，即使这些讯息是使用者在 Windows 子功能表的文件列表中选择文件时产生的（这些选项的 wParam 值是以

IDM_FIRSTCHILD 开始的)。这些讯息要传送到 DefFrameProc，并在那里进行处理。

注意框架视窗并不需要维护它所建立的所有文件视窗的视窗代号清单。如果需要这些视窗代号（如处理功能表上的「Close All」选项时），可以使用 EnumChildWindows 得到它们。

子文件视窗

现在看一下 HelloWndProc，它是用於显示「Hello, World!」的子文件视窗的视窗讯息处理程式。

与用於多个视窗的视窗类别一样，所有在视窗讯息处理程式（或从该视窗讯息处理程式中呼叫的任何函式）中定义的静态变数由依据该视窗类别建立的所有视窗共用。

只有对於每个唯一於视窗的资料才必须采用非静态变数的方法来储存。这样的技术要用到视窗属性。另一种方法（我使用的方法）是使用预留的记忆体空间；可以在注册视窗类别时将 WNDCLASS 结构的 cbWndExtra 栏位设定为非零值以便预留这部分记忆体空间。

MDIDEMO 程式使用这个记忆体空间来储存一个指标，这个指标指向一块与 HELLODATA 结构大小相同的记忆体块。在处理 WM_CREATE 讯息时，HelloWndProc 配置这块记忆体，初始化它的两个栏位（它们用於指定目前选中的功能表项和文字颜色），并用 SetWindowLong 将记忆体指标储存到预留的空间中。

当处理改变文字颜色的 WM_COMMAND 讯息（回忆一下，这些讯息来自框架视窗讯息处理程式）时，HelloWndProc 使用 GetWindowLong 获得包含 HELLODATA 结构的记忆体块的指标。利用这个结构，HelloWndProc 清除原来对功能表项的选择，设定所选功能表项为选中状态，并储存新的颜色。

当视窗变成活动视窗或不活动的时候，文件视窗讯息处理程式都会收到 WM_MDIACTIVATE 讯息（lParam 的值是否为这个视窗的代号表示了该视窗是活动的还是不活动的）。您也许还能记起 MDIDEMO 程式中有三个不同的功能表：当无文件时为 MdiMenuInit；当「Hello」文件视窗是活动视窗时为 MdiMenuHello；当「Rect」文件视窗为活动视窗时为 MdiMenuRect。

WM_MDIACTIVATE 讯息为文件视窗提供了一个修改功能表的机会。如果 lParam 中含有本视窗的代号（意味著本视窗将变成活动的），那么 HelloWndProc 就将功能表改为 MdiMenuHello。如果 lParam 中包含另一个视窗的代号，那么 HelloWndProc 将功能表改为 MdiMenuInit。

HelloWndProc 经由把 WM_MDISETMENU 讯息发送给客户视窗来修改功能表，

客户视窗透过从目前功能表上删除文件列表并把它添加到一个新的功能表上来处理这个讯息。这就是文件列表从 MdiMenuInit 功能表（它在建立第一个文件时有效）传送到 MdiMenuHello 功能表中的方法。在 MDI 應用程式中不要用 SetMenu 函式改变功能表。

另一项工作涉及到「Color」子功能表上的选中旗标。像这样的程式选项对每个文件来说都是不同的，例如，可以在一个视窗中设定黑色文字，在另一个视窗中设定红色文字。功能表选中旗标应能反映出活动视窗中选择的选项。由於这种原因，HelloWndProc 在视窗变成非活动视窗时清除选中功能表项的选中旗标，而当视窗变成活动视窗时设定适当功能表项的选中旗标。

WM_MDIACTIVATE 的 wParam 和 lParam 值分别是失去活动和被启动视窗的代号。视窗讯息处理程式得到的第一个 WM_MDIACTIVATE 讯息的 lParam 参数被设定为目前视窗的代号。而当视窗被消除时，视窗讯息处理程式得到的最後一个讯息的 lParam 参数被设定为另一个值。当使用者从一个文件切换到另一个文件时，前一个文件视窗收到一个 WM_MDIACTIVATE 讯息，其 lParam 参数为第一个视窗的代号（此时，视窗讯息处理程式将功能表设定为 MdiMenuInit）；後一个文件视窗收到一个 WM_MDIACTIVATE 讯息，其 lParam 参数是第二个视窗的代号（此时，视窗讯息处理程式将功能表设定为 MdiMenuHello 或 MdiMenuRect 中适当的那个）。如果所有的视窗都关闭了，剩下的功能表就是 MdiMenuInit。

当使用者从功能表中选择「Close」或「Close All」时，FrameWndProc 给子视窗发送一个 WM_QUERYENDSESSION 讯息。HelloWndProc 将显示一个讯息方块并询问使用者是否要关闭视窗，以此来处理 WM_QUERYENDSESSION 和 WM_CLOSE 讯息（在真实的应用程式中，讯息方块会询问是否需要储存档案）。如果使用者表示不能关闭视窗，那么视窗讯息处理程式传回 0。

在 WM_DESTROY 讯息处理期间，HelloWndProc 释放在 WM_CREATE 期间配置的记忆体块。

所有未经处理的讯息必须传送到用於内定处理的 DefMDIChildProc（不是 DefWindowProc）。不论子视窗讯息处理程式是否使用了这些讯息，有几个讯息必须被传送给 DefMDIChildProc。这些讯息是：WM_CHILDACTIVATE、WM_GETMINMAXINFO、WM_MENUCHAR、WM_MOVE、WM_SETFOCUS、WM_SIZE 和 WM_SYSCOMMAND。

RectWndProc 与 HelloWndProc 非常相似，但是它比 HelloWndProc 要简单一些（不含功能表选项并且无需使用者确认是否关闭视窗），所以这里不对它进行讨论了。但应该注意到，在处理 WM_SIZE 之後 RectWndProc 使用了「break」叙述，所以 WM_SIZE 讯息被传给 DefMDIChildProc。

结束处理

在 WinMain 中, MDIDEMO 使用 LoadMenu 载入资源描述档中定义的三个功能表。一般说来, 当功能表所在的视窗被清除时, Windows 也要清除与之关联的功能表。对于 Init 功能表, 应该清除那些没有联系到视窗的功能表。由于这个原因, MDIDEMO 在 WinMain 的末尾呼叫了两次 DestroyMenu 来清除「Hello」和「Rect」功能表。

第二十章 多工和多执行绪

多工是一个作业系统可以同时执行多个程式的能力。基本上，作业系统使用一个硬体时钟为同时执行的每个程序配置「时间片段」。如果时间片段够小，并且机器也没有由於太多的程式而超出负荷时，那么在使用者看来，所有的这些程式似乎在同时执行著。

多工并不是什么新的东西。在大型电脑上，多工是必然的。这些大型主机通常有几十甚至几百个终端机和它连结，而每个终端机使用者都应该感觉到他或者她独占了整个电脑。另外，大型主机的作业系统通常允许使用者「提交工作到背景」，这些背景作业可以在使用者进行其他工作时，由机器执行完成。

个人电脑上的多工花了更长的时间才普及化。但是现在 PC 多工也被认为是正常的了。我马上就会讨论到，Microsoft Windows 的 16 位元版本支援有限度的多工，Windows 的 32 位元版本支援真正的多工，而且，还多了一种额外的优点，多执行绪。

多执行绪是在一个程式内部实作多工的能力。程式可以把它自己分隔为各自独立的「执行绪」，这些执行绪似乎也同时在执行著。这一概念初看起来似乎没有什么用处，但是它可以让程式使用多执行绪在背景执行冗长作业，从而让使用者不必长时间地无法使用其电脑进行其他工作（有时这也许不是人们所希望的，不过这种时候去冲冲凉或者到冰箱去看看总是很不错的）！但是，即使在电脑繁忙的时候，使用者也应该能够使用它。

多工的各种模式

在 PC 的早期，有人曾经提倡未来应该朝多工的方向前进，但是大多数的人还是很迷惑：在一个单使用者的个人电脑上，多工有什么用呢？好了，最後事实表示即使是不知道这一概念的使用者也都需要多工的。

DOS 下的多工

在最初 PC 上的 Intel 8088 微处理器并不是为多工而设计的。部分原因（我在上一章中讨论过）是记忆体管理不够强。当启动和结束多个程式时，多工的作业系统通常需要移动记忆体块以收集空闲记忆体。在 8088 上是不可能透明於应用系统来做到这一点的。

DOS 本身对多工没有太大的帮助，它的设计目的是尽可能小巧，并且与独立於应用程式之外，因此，除了载入程式以及对程式提供档案系统的存取功能，

它几乎没有提供任何支援。

不过，有创意的程式写作者仍然在 DOS 的早期就找到了一种克服这些缺陷的方法，大多数是使用常驻 (TSR: terminate-and-stay-resident) 程式。有些 TSR，比如背景列印伫列程式等，透过拦截硬体时钟中断来执行真正的背景处理。其他的 TSR，诸如 SideKick 等突现式工具，可以执行某种型态的工作切换——暂停目前的应用程式，执行突现式工具。DOS 也逐渐有所增强以便提供对 TSR 的支援。

一些软体厂商试图在 DOS 之上架构出工作切换或者多工的外壳程式(shell) (诸如 Quarterdeck 的 DesqView)，但是在这些环境中，仅有其中一个占据了大部分市场，当然，这就是 Windows。

非优先权式的多工

当 Microsoft 在 1985 年发表 Windows 1.0 时，它是最成熟的解决方案，目的是突破 DOS 的局限。Windows 在实际模式下执行。但是即使这样，它已可以在实体记忆体中移动记忆体块。这是多工的前提，虽然移动的方法尚未完全透明於應用程式，但是几乎可以忍受了。

在图形视窗环境中，多工比在一种命令列单使用者作业系统中显得更有意义。例如，在传统的命令列 UNIX 中，可以在命令列之外执行程式，让它们在背景执行。然而，程式的所有显示输出必须被重新转向到一个档案中，否则输出将和使用者正在做的事情混在一起。

视窗环境允许多个程式在相同萤幕上一起执行，前後切换非常容易，并且还可以快速地将资料从一个程式移动到另一个程式中。例如，将绘图程式中建立的图片嵌入由文书处理程式编辑的文字档案中。在 Windows 中，以多种方式支援资料转移，首先是使用剪贴簿，后来又使用动态资料交换 (DDE)，而现在则是透过物件连结和嵌入 (OLE)。

不过，早期 Windows 的多工实作还不是多使用者作业系统中传统的优先权式的分时多工。这些作业系统使用系统时钟周期性地中断一个工作并开始另一个工作。Windows 的这些 16 位元版本支援一种被称为「非优先权式的多工」，由於 Windows 讯息驱动的架构而使这种型态的多工成为可能。通常情况下，一个 Windows 程式将在记忆体中睡眠，直到它收到一个讯息为止。这些讯息通常是使用者的键盘或鼠标输入的直接或间接结果。当处理完讯息之後，程式将控制权返回给 Windows。

Windows 的 16 位元版本不会绝对地依据一个 timer tick 将控制权从一个 Windows 程式切换到另一个，任何的工作切换都发生在当程式完成对讯息的处理

后将控制权返回给 Windows 时。这种非优先权式的多工也被称为「合作式的多工」，因为它要求来自應用程式方面的一些合作。一个 Windows 程式可以占用整个系统，如果它要花很长一段时间来处理讯息的话。

虽然非优先权式的多工是 16 位元 Windows 的一般规则，但仍然出现了某些形式的优先权式多工。Windows 使用优先权式多工来执行 DOS 程式，而且，为了实作多媒体，还允许动态连结程式库接收硬体时钟中断。

16 位元 Windows 包括几个功能特性来帮助程式写作者解决（或者，至少可以说是对付）非优先权式多工中的局限，最显著的当然是时钟式滑鼠游标。当然，这并非一种解决方案，而仅仅是让使用者知道一个程式正在忙於处理一件冗长作业，因而让使用者在一段时间内无法使用系统。另一种解决方案是 Windows 计时器，它允许程式周期性地接收讯息并完成一些工作。计时器通常用於时钟应用和动画。

针对非优先权式多工的另一种解决方案是 PeekMessage 函式呼叫，我们曾在第五章中的 RANDRECT 程式里看到过。一个程式通常使用 GetMessage 呼叫从它的讯息伫列中找寻下一个讯息，不过，如果在讯息伫列中没有讯息，那么 GetMessage 不会传回，一直到出现一个讯息为止。而另一方面，PeekMessage 将控制权传回程式，即使没有等待的讯息。这样，一个程式可以执行一个冗长作业，并在程式码中混入 PeekMessage 呼叫。只要没有这个程式或其他任何程式的讯息要处理，那么这个冗长作业将继续执行。

Presentation Manager 和序列化的讯息伫列

Microsoft 在一种半 DOS/半 Windows 的环境下实作多工的第一个尝试（和 IBM 合作）是 OS/2 和 Presentation Manager（缩写成 PM）。虽然 OS/2 明确地支援优先权式多工，但是这种多工方式似乎并未在 Presentation Manager 中得以落实。问题在於 PM 序列化来自键盘和滑鼠的使用者输入讯息。这意味著，在前一个使用者输入讯息被完全处理以前，PM 不会将一个键盘或者滑鼠讯息传送给程式。

尽管键盘和滑鼠讯息只是一个 PM（或者 Windows）程式可以接收的许多讯息中的几个，大多数的其他讯息都是键盘或者滑鼠事件的结果。例如，功能表命令讯息是使用者使用键盘或者滑鼠进行功能表选择的结果。在处理功能表命令讯息时，键盘或者滑鼠讯息并未完全被处理。

序列化讯息伫列的主要原因是允许使用者的预先「键入」键盘按键和预先「按入」滑鼠按钮。如果一个键盘或者滑鼠讯息导致输入焦点从一个视窗切换到另一个视窗，那么接下来的键盘讯息应该进入拥有新的输入焦点的视窗中去。

因此，系统不知道将下一个使用者输入讯息发送到何处，直到前一个讯息被处理完为止。

目前的共识是不应该让一个应用系统有可能占用整个系统，而这需要非序列化的讯息伫列，32 位元版本的 Windows 支援这种讯息伫列。如果一个程式正在忙著处理一项冗长作业，那么您可以将输入焦点切换到另一个程式中。

多执行绪解决方案

我讨论 OS/2 的 Presentation Manager，只是因为它是第一个为早期的 Windows 程式写作者（比如我自己）介绍多执行绪的环境。有趣的是，PM 实作多执行绪的局限为程式写作者提供了应该如何架构多执行绪程式的必要线索。即使这些限制在 32 位元的 Windows 中已经大幅减少，但是从更有限的环境中学到的经验仍然是非常有效的。因此，让我们继续讨论下去。

在一个多执行绪环境中，程式可以将它们自己分隔为同时执行的片段（叫做执行绪）。对执行绪的支援是解决 PM 中存在的序列化讯息伫列的最好方法，并且在 Windows 中执行绪有更实际的意义。

就程式码来说，一个执行绪简单地被表示为可能呼叫程式中其他函式的函式。程式从其主执行绪开始执行，这个主执行绪是在传统的 C 程式中叫做 main 的函式，而在 Windows 中是 WinMain。一旦执行起来，程式可以通过在系统呼叫 CreateThread 中指定初始执行绪函式的名称来建立新的执行绪的执行。作业系统在执行绪之间优先权式地切换控制项，和它在程序之间切换控制权的方法非常类似。

在 OS/2 的 Presentation Manager 中，每个执行绪可以建立一个讯息伫列，也可以不建立。如果希望从执行绪建立视窗，那么一个 PM 执行绪必须建立讯息伫列。否则，如果只是进行许多的资料处理或者图形输出，那么执行绪不需要建立讯息伫列。因为无讯息伫列的程序不处理讯息，所以它们将不会当住系统。唯一的限制是一个无讯息伫列执行绪无法向一个讯息伫列执行绪中的视窗发送讯息，或者呼叫任何发送讯息的函式（不过，它们可以将讯息递送给讯息伫列执行绪）。

这样，PM 程式写作者学会了如何将它们的程式分隔为一个讯息伫列执行绪（在其中建立所有的视窗并处理传送给视窗的讯息）和一个或者多个无讯息伫列执行绪，在其中执行冗长的背景工作。PM 程式写作者还了解到「1/10 秒规则」，大体上，程式写作者被告知，一个讯息伫列执行绪处理任何讯息都不应该超过 1/10 秒，任何花费更长时间的事情都应该在另一个执行绪中完成。如果所有的程式写作者都遵循这一规则，那么将没有 PM 程式会将系统当住超过 1/10 秒。

多执行绪架构

我已经说过 PM 的限制让程式写作者理解如何在图形环境中执行的程式里头使用多个执行绪提供了必要的线索。因此在这里我将为您的程式建议一种架构：您的主执行绪建立您程式所需要的所有视窗，并在其中包含所有的视窗讯息处理程式，以便处理这些视窗的所有讯息；所有其他执行绪只进行一些背景处理，除了和主执行绪通讯，它们不和使用者进行交流。

可以把这种架构想像成：主执行绪处理使用者输入（和其他讯息），并建立程序中的其他执行绪，这些附加的执行绪完成与使用者无关的工作。

换句话说，您程式的主执行绪是一个老板，而您的其他执行绪是老板的职员。老板将大的工作丢给职员处理，而他自己保持和外界的联系。因为那些执行绪仅仅是职员，所以其他执行绪不会举行它们自己的记者招待会。它们会认真地完成自己的工作，将结果报告给老板，并等待他们的下一个任务。

一个程式中的执行绪是同一程序的不同部分，因此他们共用程序的资源，如记忆体和打开的档案。因为执行绪共用程式的记忆体，所以他们还共用静态变数。然而，每个执行绪都有他们自己的堆叠，因此动态变数对每个执行绪是唯一的。每个执行绪还有各自的处理器状态（和数学辅助运算器状态），这个状态在进行执行绪切换期间被储存和恢复。

执行绪间的「争吵」

正确地设计、写作和测试一个复杂的多执行绪應用程式显然是 Windows 程式写作者可能遇到的最困难的工作之一。因为优先权式多工系统可以在任何时刻中断一个执行绪，并将控制权切换到另一个执行绪中，在两个执行绪之间可能有无法预料的随机交互作用的情况。

多执行绪程式中的一个常见的错误被称为「竞争状态 (race condition)」，这发生在程式写作者假设一个执行绪在另一个执行绪需要某资料之前已经完成了某些处理（如准备资料）的时候。为了帮助协调执行绪的活动，作业系统要求各种形式的同步。一种是同步信号 (semaphore)，它允许程式写作者在程式码中的某一点阻止一个执行绪的执行，直到另一个执行绪发信号让它继续为止。类似於同步信号的是「临界区域 (critical section)」，它是程式码中不可中断的部分。

但是同步信号还可能产生称为「锁死 (deadlock)」的常见执行绪错误，这发生在两个执行绪互相阻止了另一个的执行，而继续执行的唯一办法又是它们继续向前执行。

幸运的是，32 位元程式比 16 位元程式更能抵抗执行绪所涉及的某些问题。例如，假定一个执行绪执行下面的简单叙述：

```
lCount++ ;
```

其中 lCount 是由其他执行绪使用的一个 32 位元的 long 型态变数，C 中的这个叙述被编译为两条机械码指令，第一条将变数的低 16 位元加 1，而第二条指令将任何可能的进位加到高 16 位上。假定作业系统在这两个机械码指令之间中断了执行绪。如果 lCount 在第一条机械码指令之前是 0x0000FFFF，那么 lCount 在执行绪被中断时为 0，而这正是另一个执行绪将看到的值。只有当执行绪继续执行时，lCount 才会增加到正确的值 0x00010000。

这是那些偶尔会导致操作问题的错误之一。在 16 位元程式中，解决此问题正确的方法是将叙述包含在一个临界区域中，在这期间执行绪不会被中断。然而，在一个 32 位元程式中，该叙述是正确的，因为它被编译为一条机械码指令。

Windows 的好处

32 位元 Windows 版本（包括 Windows NT 和 Windows 98）有一个非序列化的讯息伫列。这种实作似乎非常好：如果一个程式正在花费一段长时间处理一个讯息，那么滑鼠位於该程式的视窗上时，滑鼠游标将呈现为一个时钟，但是当将滑鼠移到另一个程式的视窗上时，滑鼠游标将变为正常的箭头形状。只需按一下就可以将另一个视窗提到前面来。

然而，使用者仍然不能使用正在处理大量工作的那个程式，因为那些工作会阻止程式接收其他讯息，这不是我们所希望的。一个程式应该总是能随时处理讯息的，所以这时就需要使用从属执行绪了。

在 Windows NT 和 Windows 98 中，没有讯息伫列执行绪和无讯息伫列执行绪的区别，每个执行绪在建立时都会有它自己的讯息伫列，从而减少了 PM 程式中关于执行绪的一些不便规定（然而，在大多数情况下，您仍然想通过一条专门处理讯息的执行绪中的讯息程序处理输入，而将冗长作业交给那些不包含视窗的执行绪处理，这种结构几乎总是最容易理解的，我们将看到这一点）。

还有更好的事情：Windows NT 和 Windows 98 中还有个函式允许执行绪杀死同一程序中的另一个执行绪。当您开始编写多执行绪程式码时，您将会发现这种功能在有时是很方便的。OS/2 的早期版本没有「杀死执行绪」的函式。

最後的好讯息（至少对这里的话题是好讯息）是 Windows NT 和 Windows 98 实作了一些被称为「执行绪区域储存空间 (TLS: thread local storage)」的功能。为了了解这一点，回顾一下我在前面提到过的，静态变数（对一个函式来说，既是整体又是区域变数）在执行绪之间是被共用的，因为它们位於程序

的资料储存空间中。动态变数（对一个函式来说总是区域变数）对每一个执行绪则是唯一的，因为它们占据堆叠上的空间，而每个执行绪都有它自己的堆叠。

有时让两个或多个执行绪使用相同的函式，而让这些执行绪使用唯一於执行绪的静态变数，那会带来很大便利。这就是执行绪区域储存空间，其中涉及一些 Windows 函式呼叫，但是 Microsoft 还为 C 编译器进行扩展，使执行绪区域储存空间的使用更透明于程式写作者。

新改良过的！支援多执行绪了！

既然已经介绍了执行绪的现状，让我们来展望一下执行绪的未来。有时，有人会出现一种使用作业系统所提供的每一种功能特性的冲动。最坏的情况是，当您的老板走到您的桌前并说：「我听说这种新功能非常炫，让我们在自己的程式中用一些这种新功能吧。」然後您将花费一个星期的时间，试图去了解您的应用程式如何从这种新功能获益。

应该注意的是，在并不需要多执行绪的应用系统中加入多执行绪是没有任何意义的。如果您的程式显示沙漏游标的时间太长，或者如果它使用 PeekMessage 呼叫来避免沙漏游标的出现，那么请重新规划您的程式架构，使用多执行绪可能会是一个好主意。其他情形，您是在为难您自己，并可能会在程式码中产生新的错误。

在某些情况下，沙漏游标的出现可能是完全适当的。我在前面提到过「1/10 秒规则」，而将一个大档案载入记忆体可能会花费多於 1/10 秒的时间，这是否意味著档案载入常式应该在分离的执行绪中实作呢？没有必要。当使用者命令一个程式打开档案时，他或者她通常想立即完成该操作。将档案载入常式放在分离的执行绪中只会增加额外的负担。即使您想向您的朋友夸耀您在编写多执行绪程式，也完全不值得这样做！

WINDOWS 的多执行绪处理

建立新的执行绪的 API 函式是 CreateThread，它的语法如下：

```
hThread = CreateThread (&security_attributes, dwStackSize, ThreadProc,  
                        pParam, dwFlags, &idThread) ;
```

第一个参数是指向 SECURITY_ATTRIBUTES 型态的结构体的指标。在 Windows 98 中忽略该参数。在 Windows NT 中，它被设为 NULL。第二个参数是用於新执行绪的初始堆叠大小，预设值为 0。在任何情况下，Windows 根据需要动态延长堆叠的大小。

CreateThread 的第三个参数是指向执行绪函式的指标。函式名称没有限制，

但是必须以下列形式宣告：

```
DWORD WINAPI ThreadProc (PVOID pParam) ;
```

CreateThread 的第四个参数为传递给 ThreadProc 的参数。这样主执行绪和从属执行绪就可以共用资料。

CreateThread 的第五个参数通常为 0，但当建立的执行绪不马上执行时为旗标 CREATE_SUSPENDED。执行绪将暂停直到呼叫 ResumeThread 来恢复执行绪的执行为止。第六个参数是一个指标，指向接受执行绪 ID 值的变数。

大多数 Windows 程式写作者喜欢用在 PROCESS.H 表头档案中宣告的 C 执行时期程式库函式_beginthread。它的语法如下：

```
hThread = _beginthread (ThreadProc, uiStackSize, pParam) ;
```

它更简单，对于大多数应用程式很完美，这个执行绪函式的语法为：

```
void __cdecl ThreadProc (void * pParam) ;
```

再论随机矩形

程式 20-1 RNDRCTMT 是第五章里的 RANDRECT 程式的多执行绪版本，您将回忆起 RANDRECT 使用的是 PeekMessage 回圈来显示一系列的随机矩形。

程式 20-1 RNDRCTMT

```
RNDRCTMT.C
/*-----
-
-      RNDRCTMT.C --      Displays Random Rectangles
-                               (c) Charles Petzold, 1998
-
-----*/

#include <windows.h>
#include <process.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HWND  hwnd ;
int   cxClient, cyClient ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("RndRctMT") ;
    MSG              msg ;
    WNDCLASS          wndclass ;

    wndclass.style
                        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
                        = WndProc ;
    wndclass.cbClsExtra
                        = 0 ;
```



```
wndclass.cbWndExtra          = 0 ;
wndclass.hInstance          = hInstance ;
wndclass.hIcon              = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor            = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName        = NULL ;
wndclass.lpszClassName       = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("Random Rectangles"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

VOID Thread (PVOID pvoid)
{
    HBRUSH      hBrush ;
    HDC          hdc ;
    int          xLeft, xRight, yTop, yBottom, iRed, iGreen, iBlue ;

    while (TRUE)
    {
        if (cxClient != 0 || cyClient != 0)
        {
            xLeft          = rand () % cxClient ;
            xRight         = rand () % cxClient ;
            yTop           = rand () % cyClient ;
            yBottom        = rand () % cyClient ;
            iRed            = rand () & 255 ;
            iGreen         = rand () & 255 ;
            iBlue          = rand () & 255 ;
        }
    }
}
```

```

        hdc = GetDC (hwnd) ;
        hBrush = CreateSolidBrush (RGB (iRed, iGreen, iBlue)) ;
        SelectObject (hdc, hBrush) ;

        Rectangle (hdc,  min (xLeft, xRight), min (yTop, yBottom),
max (xLeft, xRight), max (yTop, yBottom)) ;

        ReleaseDC (hwnd, hdc) ;
        DeleteObject (hBrush) ;
    }
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
        case WM_CREATE:
            _beginthread (Thread, 0, NULL) ;
            return 0 ;

        case WM_SIZE:
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;
            return 0 ;

        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

在建立多执行绪的 Windows 程式时，需要在「Project Settings」对话方块中做一些修改。选择「C/C++」页面标签，然後在「Category」下拉式清单方块中选择「Code Generation」。在「Use Run-Time Library」下拉式清单方块中，可以看到用於「Release」设定的「Single-Threaded」和用於 Debug 设定的「Debug Single-Threaded」。将这些分别改为「Multithreaded」和「Debug Multithreaded」。这将把编译器旗标改为/MT，它是编译器在编译多执行绪的应用程式所需要的。具体地说，编译器将在.OBJ 档案中插入 LIBCMT.LIB 档案名，而不是 LIBC.LIB。连结程式使用这个名称与执行期程式库函式连结。

LIBC.LIB 和 LIBCMT.LIB 档案包含 C 语言程式库函式，有些 C 语言程式库函式包含静态资料。例如，由於 strtok 函式可能被连续地多次呼叫，所以它在静态记忆体中储存了一个指标。在多执行绪程式中，每个执行绪必须在 strtok 函

式中有它自己的静态指标。因此，这个函式的多执行绪版本稍微不同於单执行绪的 strtok 函式。

同时请注意，我在 RNDRCTMT.C 中包含了表头档案 PROCESS.H，这个档案定义一个名为_beginthread 的函式，它启动一个新的执行绪。只有定义了_MT 识别字，才会宣告这个函式，这是/MT 旗标的另一个结果。

在 RNDRCTMT.C 的 WinMain 函式中，由 CreateWindow 传回的 hwnd 值被储存在一个整体变数中，因此 cxClient 和 cyClient 值也可以由视窗讯息处理程式的 WM_SIZE 讯息获得。

视窗讯息处理程式以最容易的方法呼叫_beginthread——简单地以执行绪函式的位址（称为 Thread）作为第一个参数，其他参数使用 0，执行绪函式传回 VOID 并有一个参数，该参数是一个指向 VOID 的指标。在 RNDRCTMT 中的 Thread 函式不使用这个参数。

在呼叫了_beginthread 函式之後，执行绪函式（以及该执行绪函式可能呼叫的其他任何函式）中的程式码和程式中的其他程式码同时执行。两个或者多个执行绪使用一个程序中的同一函式，在这种情况下，动态区域变数（储存在堆叠上）对每个执行绪是唯一的。对程序中的所有执行绪来说，所有的静态变数都是一样的。这就是视窗讯息处理程式设定整体的 cxClient 和 cyClient 变数并由 Thread 函式使用的方式。

有时您需要唯一於各个执行绪的持续储存性资料。通常，这种资料是静态变数，但在 Windows 98 中，您可以使用「执行绪区域储存空间」，我将在本章後面进行讨论。

程式设计竞赛的问题

1986 年 10 月 3 日，Microsoft 举行了为期一天，针对电脑杂志出版社的技术编辑和作者的简短的记者招待会，来讨论他们当时的一组语言产品，包括他们的第一个交谈式开发环境，QuickBASIC 2.0。当时，Windows 1.0 出现还不到一年，但是没有人知道我们什么时候能得到与该环境类似的东西（这花了好几年）。这一事件与众不同的部分原因是由於 Microsoft 的公关人员所举办的「Storm the Gates」程式设计竞赛。Bill Gates 使用 QuickBASIC 2.0，而电脑出版社的人员可以使用他们选择的任何语言产品。

竞赛的问题是从公众提出的题目中挑选出来的（挑选那些需要写大约半小时程式来解决的问题），问题如下：

建立一个包含四个视窗的多工模拟程式。第一个视窗必须显示一系列的递增数，第二个必须显示一系列的递增质数，而第三个必须显示 Fibonacci 数列

(Fibonacci 数列以数字 0 和 1 开始, 後头每一个数都是其前两个数的和——即 0、1、1、2、3、5、8 等等)。这三个视窗应该在数字达到视窗底部时或者进行滚动, 或者自行清除视窗内容。第四个视窗必须显示任意半径的圆, 而程式必须在按下下一个 Escape 键时终止。

当然, 在 1986 年 10 月, 在 DOS 下执行的这样一个程式最多只能是模拟多工而已, 而且没有一个竞赛者具有足够的勇气——并且其中大多数也没有足够的知识——来为 Windows 编写这个程式。再者, 如果真要这么做, 当然不会只花半小时了!

参加这次竞赛的大多数人编写了一个程式来将萤幕分为四个区域, 程式中包含一个回圈, 依次更新每个视窗, 然後检查是否按下了 Escape 键。如同 DOS 环境下的传统习惯, 程式占用了百分之百的 CPU 处理时间。

如果在 Windows 1.0 中写程式, 那么结果将是类似程式 20-2 MULTI1 的结果。我说「类似」, 是因为我编写的程式是 32 位元的, 但程式结构和相当多的程式码——除了变数和函式参数定义以及 Unicode 支援——都是相同的。

程式 20-2 MULTI1

```
MULTI1.C
/*-----
MULTI1.C --      Multitasking Demo
(c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <math.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int cyChar ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Multi1") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
```

```

    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
            szAppName, MB_ICONERROR) ;
        return 0 ;
    }
    hwnd = CreateWindow (    szAppName, TEXT ("Multitasking Demo"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

int CheckBottom (HWND hwnd, int cyClient, int iLine)
{
    if (iLine * cyChar + cyChar > cyClient)
    {
        InvalidateRect (hwnd, NULL, TRUE) ;
        UpdateWindow (hwnd) ;
        iLine = 0 ;
    }
    return iLine ;
}

// -----
// Window 1: Display increasing sequence of numbers
// -----

LRESULT APIENTRY WndProc1 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static int      iNum, iLine, cyClient ;
    HDC          hdc ;
    TCHAR        szBuffer[16] ;

```

```

switch (message)
{
case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;

case WM_TIMER:
    if (iNum < 0)
        iNum = 0 ;

    iLine = CheckBottom (hwnd, cyClient, iLine) ;
    hdc = GetDC (hwnd) ;

    TextOut (hdc, 0, iLine * cyChar, szBuffer,
        wsprintf (szBuffer, TEXT ("%d"), iNum++)) ;

    ReleaseDC (hwnd, hdc) ;
    iLine++ ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// -----
// Window 2: Display increasing sequence of prime numbers
// -----

LRESULT APIENTRY WndProc2 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int      iNum = 1, iLine, cyClient ;
    HDC            hdc ;
    int            i, iSqrt ;
    TCHAR          szBuffer[16] ;

    switch (message)
    {
case WM_SIZE:
        cyClient = HIWORD (lParam) ;
        return 0 ;

case WM_TIMER:
        do
        {
            if (++iNum < 0)
                iNum = 0 ;

            iSqrt = (int) sqrt (iNum) ;

```

```

                                for (i = 2 ; i <= iSqrt ; i++)
                                    if (iNum % i == 0)
                                        break ;
                                }
                                while (i <= iSqrt) ;

                                iLine = CheckBottom (hwnd, cyClient, iLine) ;
                                hdc = GetDC (hwnd) ;

                                TextOut (  hdc, 0, iLine * cyChar, szBuffer,
                                            wsprintf (szBuffer, TEXT ("%d"), iNum)) ;
                                ReleaseDC (hwnd, hdc) ;
                                iLine++ ;
                                return 0 ;
                            }
                            return DefWindowProc (hwnd, message, wParam, lParam) ;
                        }

// -----
// Window 3: Display increasing sequence of Fibonacci numbers
// -----

LRESULT APIENTRY WndProc3 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int      iNum = 0, iNext = 1, iLine, cyClient ;
    HDC             hdc ;
    int             iTemp ;
    TCHAR           szBuffer[16] ;

    switch (message)
    {
    case WM_SIZE:
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_TIMER:
        if (iNum < 0)
        {
            iNum = 0 ;
            iNext = 1 ;
        }

        iLine = CheckBottom (hwnd, cyClient, iLine) ;
        hdc = GetDC (hwnd) ;

        TextOut (  hdc, 0, iLine * cyChar, szBuffer,

```

```

wsprintf (szBuffer, "%d",
iNum)) ;

        ReleaseDC (hwnd, hdc) ;
        iTemp      =      iNum ;
        iNum       =      iNext ;
        iNex       +=      iTemp ;
        iLine++ ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// -----
// Window 4: Display circles of random radii
//
// -----

LRESULT APIENTRY WndProc4 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static int      cxClient, cyClient ;
    HDC             hdc ;
    int             iDiameter ;

    switch (message)
    {
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_TIMER:
        InvalidateRect (hwnd, NULL, TRUE) ;
        UpdateWindow (hwnd) ;

        iDiameter = rand() % (max (1, min (cxClient, cyClient))) ;
        hdc = GetDC (hwnd) ;

        Ellipse (hdc,
                 (cxClient - iDiameter) / 2,
                 (cyClient - iDiameter) / 2,
                 (cxClient + iDiameter) / 2,
                 (cyClient + iDiameter) / 2) ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```



```
// -----
// Main window to create child windows
// -----

LRESULT APIENTRY WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND          hwndChild[4] ;
    static TCHAR *      szChildClass[] = { TEXT ("Child1"), TEXT ("Child2"),
    TEXT ("Child3"), TEXT ("Child4") } ;
    static WNDPROC      ChildProc[] =    { WndProc1, WndProc2, WndProc3,
WndProc4 } ;
    HINSTANCE           hInstance ;
    int                  i, cxClient, cyClient ;
    WNDCLASS             wndclass ;

    switch (message)
    {
    case WM_CREATE:
        hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

        wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
        wndclass.cbClsExtra      = 0 ;
        wndclass.cbWndExtra      = 0 ;
        wndclass.hInstance      = hInstance ;
        wndclass.hIcon           = NULL ;
        wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground = (HBRUSH) GetStockObject
(WHITE_BRUSH) ;

        wndclass.lpszMenuName = NULL ;

        for (i = 0 ; i < 4 ; i++)
        {
            wndclass.lpfnWndProc = ChildProc[i] ;
            wndclass.lpszClassName = szChildClass[i] ;

            RegisterClass (&wndclass) ;

            hwndChild[i] = CreateWindow (szChildClass[i], NULL,
            WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE,
            0, 0, 0, 0,
            hwnd, (HMENU) i, hInstance, NULL) ;
        }

        cyChar = HIWORD (GetDialogBaseUnits ()) ;
        SetTimer (hwnd, 1, 10, NULL) ;
        return 0 ;
    }
}
```

```

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    for (i = 0 ; i < 4 ; i++)
        MoveWindow (hwndChild[i], (i % 2) * cxClient / 2,
                    (i > 1) * cyClient / 2,
                    cxClient / 2, cyClient / 2, TRUE) ;
    return 0 ;

case WM_TIMER:
    for (i = 0 ; i < 4 ; i++)
        SendMessage (hwndChild[i], WM_TIMER, wParam, lParam) ;

    return 0 ;

case WM_CHAR:
    if (wParam == '\\xB')
        DestroyWindow (hwnd) ;

    return 0 ;

case WM_DESTROY:
    KillTimer (hwnd, 1) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

在这个程式里实际上没有什么我们没见过的东西。主视窗建立四个子视窗，每个子视窗占据显示区域的一个象限。主视窗还设定一个 Windows 计时器并发送 WM_TIMER 讯息给四个子视窗中的每一个。

通常一个 Windows 程式应该保留足够的资讯以便在 WM_PAINT 讯息处理期间重建其视窗中的内容。MULTI1 没有这么做，既然它绘制和清除视窗的速度如此之快，所以我认为那是不必要的。

WndProc2 中的质数产生器的效率并不很高，但是有效。如果一个数除了 1 和它自身以外没有别的因数，那么这个数就是质数。当然，要检查一个数是否是质数并不要求使用小於被检查数的所有数来除这个数并检查余数，而只需使用所有小於被检查数的平方根的数。平方根计算是发表浮点数的原因，否则，该程式将是完全依据整数的程式。

MULTI1 程式没有什么不好的地方。使用 Windows 计时器是在 Windows 的早期（和目前）版本中模拟多工的一种好方法，然而，计时器的使用有时限制了

程式的速度。如果程式可以在 WM_TIMER 讯息处理中更新它的所有视窗而还有时间剩余下来的话，那就意味著它并没有充分利用我们的机器资源。

一种可能的解决方案是在单个 WM_TIMER 讯息处理期间进行两次或者更多次的更新，但是到底多少次呢？这不得不依赖于机器的速度，而有很大的变动性。您当然不会想编写一个只能适用于 25MHz 的 386 或 50MHz 的 486 或 100-GHz 的 Pentium VII 上的程式吧。

多执行绪解决方案

让我们来看一看关于这个程式设计问题的一种多执行绪解决方案。如程式 20-3 MULTI2 所示。

程式 20-3 MULTI2

```
MULTI2.C
/*-----
-
MULTI2.C --      Multitasking Demo
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include <math.h>
#include <process.h>

typedef struct
{
    HWND      hwnd ;
    int       cxClient ;
    int       cyClient ;
    int       cyChar ;
    BOOL bKill ;
}
PARAMS, *PPARAMS ;
LRESULT APIENTRY WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("Multi2") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
                    = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
                    = WndProc ;
```

```
wndclass.cbClsExtra          = 0 ;
wndclass.cbWndExtra          = 0 ;
wndclass.hInstance          = hInstance ;
wndclass.hIcon               = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor             = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
wndclass.lpszMenuName         = NULL ;
wndclass.lpszClassName       = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
        szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow ( szAppName, TEXT ("Multitasking Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

int CheckBottom (HWND hwnd, int cyClient, int cyChar, int iLine)
{
    if (iLine * cyChar + cyChar > cyClient)
    {
        InvalidateRect (hwnd, NULL, TRUE) ;
        UpdateWindow (hwnd) ;
        iLine = 0 ;
    }
    return iLine ;
}

// -----
// Window 1: Display increasing sequence of numbers
// -----
```

```

void Thread1 (PVOID pvoid)
{
    HDC                hdc ;
    int                iNum = 0, iLine = 0 ;
    PPARAMS            pparams ;
    TCHAR              szBuffer[16] ;

    pparams            = (PPARAMS) pvoid ;

    while (!pparams->bKill)
    {
        if (iNum < 0)
            iNum = 0 ;

        iLine = CheckBottom ( pparams->hwnd,
        pparams->cyClient,
        pparams->cyChar,    iLine) ;

        hdc = GetDC (pparams->hwnd) ;

        TextOut (  hdc, 0, iLine * pparams->cyChar, szBuffer,
                  wsprintf (szBuffer, TEXT ("%d"), iNum++)) ;

        ReleaseDC (pparams->hwnd, hdc) ;
        iLine++ ;
    }
    _endthread () ;
}

LRESULT APIENTRY WndProc1 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static PARAMS params ;
    switch (message)
    {
    case WM_CREATE:
        params.hwnd = hwnd ;
        params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
        _beginthread (Thread1, 0, 耗s) ;
        return 0 ;

    case WM_SIZE:
        params.cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_DESTROY:
        params.bKill = TRUE ;
        return 0 ;
    }
}

```

```

        return DefWindowProc (hwnd, message, wParam, lParam) ;
    }

// -----
// Window 2: Display increasing sequence of prime numbers
// -----

void Thread2 (PVOID pvoid)
{
    HDC                hdc ;
    int                iNum = 1, iLine = 0, i, iSqrt ;
    PPARAMS            pparams ;
    TCHAR              szBuffer[16] ;

    pparams = (PPARAMS) pvoid ;
    while (!pparams->bKill)
    {
        do
        {
            if (++iNum < 0)
                iNum = 0 ;
            iSqrt = (int) sqrt (iNum) ;
            for (i = 2 ; i <= iSqrt ; i++)
                if (iNum % i == 0)
                    break ;
        }
        while (i <= iSqrt) ;
        iLine = CheckBottom      (      pparams->hwnd,
        pparams->cyClient,
        pparams->cyChar,    iLine) ;

        hdc = GetDC (pparams->hwnd) ;

        TextOut (  hdc, 0, iLine * pparams->cyChar, szBuffer,
                  wsprintf (szBuffer, TEXT ("%d"), iNum)) ;

        ReleaseDC (pparams->hwnd, hdc) ;
        iLine++ ;
    }
    _endthread () ;
}

LRESULT APIENTRY WndProc2 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static PARAMS params ;
    switch (message)
    {

```

```

    case WM_CREATE:
        params.hwnd = hwnd ;
        params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
        _beginthread (Thread2, 0, 耗s) ;
        return 0 ;

    case WM_SIZE:
        params.cyClient = HIWORD (lParam) ;
        return 0 ;

    case WM_DESTROY:
        params.bKill = TRUE ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// Window 3: Display increasing sequence of Fibonacci numbers
// -----

void Thread3 (PVOID pvoid)
{
    HDC                hdc ;
    int                iNum = 0, iNext = 1, iLine = 0, iTemp ;
    PPARAMS            pparams ;
    TCHAR              szBuffer[16] ;

    pparams = (PPARAMS) pvoid ;
    while (!pparams->bKill)
    {
        if (iNum < 0)
        {
            iNum = 0 ;
            iNext = 1 ;
        }
        iLine = CheckBottom ( pparams->hwnd, pparams->cyClient,
            pparams->cyChar, iLine) ;

        hdc = GetDC (pparams->hwnd) ;

        TextOut (hdc, 0, iLine * pparams->cyChar, szBuffer,
            wsprintf (szBuffer, TEXT ("%d"), iNum)) ;

        ReleaseDC (pparams->hwnd, hdc) ;
        iTemp = iNum ;
        iNum = iNext ;
        iNext += iTemp ;
        iLine++ ;
    }
}

```

```

    }
    _endthread () ;
}

LRESULT APIENTRY WndProc3 ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static PARAMS params ;
    switch (message)
    {
        case WM_CREATE:
            params.hwnd = hwnd ;
            params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
            _beginthread (Thread3, 0, 耗s) ;
            return 0 ;

        case WM_SIZE:
            params.cyClient = HIWORD (lParam) ;
            return 0 ;

        case WM_DESTROY:
            params.bKill = TRUE ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// -----
// Window 4: Display circles of random radii
// -----

void Thread4 (PVOID pvoid)
{
    HDC          hdc ;
    int          iDiameter ;
    PPARAMS      pparams ;

    pparams = (PPARAMS) pvoid ;
    while (!pparams->bKill)
    {
        InvalidateRect (pparams->hwnd, NULL, TRUE) ;
        UpdateWindow (pparams->hwnd) ;

        iDiameter =      rand() % (max (1,
min (pparams->cxCClient, pparams->cyClient))) ;

        hdc = GetDC (pparams->hwnd) ;
    }
}

```



```

        Ellipse (hdc,      (pparams->cxCClient - iDiameter) / 2,
                    (pparams->cyCClient - iDiameter) / 2,
                    (pparams->cxCClient + iDiameter) / 2,
                    (pparams->cyCClient + iDiameter) / 2) ;

        ReleaseDC (pparams->hwnd, hdc) ;
    }
    _endthread () ;
}

LRESULT APIENTRY WndProc4 (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static PARAMS params ;
    switch (message)
    {
        case WM_CREATE:
            params.hwnd = hwnd ;
            params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
            _beginthread (Thread4, 0, 耗s) ;
            return 0 ;

        case WM_SIZE:
            params.cxCClient = LOWORD (lParam) ;
            params.cyCClient = HIWORD (lParam) ;
            return 0 ;

        case WM_DESTROY:
            params.bKill = TRUE ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// -----
// Main window to create child windows
// -----

LRESULT APIENTRY WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND          hwndChild[4] ;
    static TCHAR * szChildClass[] = { TEXT ("Child1"), TEXT ("Child2"),
        TEXT ("Child3"), TEXT ("Child4") } ;
    static WNDPROC      ChildProc[] = { WndProc1, WndProc2, WndProc3, WndProc4 } ;
    HINSTANCE           hInstance ;
    int                  i, cxCClient, cyCClient ;
    WNDCLASS             wndclass ;

```

```

switch (message)
{
case WM_CREATE:
    hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;
    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = NULL ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;

    for (i = 0 ; i < 4 ; i++)
    {
        wndclass.lpfnWndProc = ChildProc[i]
        wndclass.lpszClassName = szChildClass[i] ;

        RegisterClass (&wndclass) ;

        hwndChild[i] = CreateWindow (szChildClass[i], NULL,
        WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE,
        0, 0, 0, 0,
        hwnd, (HMENU) i, hInstance, NULL) ;
    }

    return 0 ;

case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    for (i = 0 ; i < 4 ; i++)
        MoveWindow (hwndChild[i], (i % 2) * cxClient / 2,
                    (i > 1) * cyClient / 2,
                    cxClient / 2, cyClient / 2, TRUE) ;
    return 0 ;

case WM_CHAR:
    if (wParam == '\x1B')
        DestroyWindow (hwnd) ;

    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}

```

```

    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

MULTI2.C 的 WinMain 和 WndProc 函式非常类似於 MULTI1.C 中的同名函式。WndProc 为四个视窗注册了四种视窗类别，建立了这些视窗，并在 WM_SIZE 讯息处理期间缩放这些视窗。WndProc 的唯一不同是它不再设定 Windows 计时器，也不再处理 WM_TIMER 讯息。

MULTI2 中较大的改变是每个子视窗讯息处理程式透过在 WM_CREATE 讯息处理期间呼叫 `_beginthread` 函式来建立另一个执行绪。总括来说，MULTI2 程式有五个同时执行的执行绪，主执行绪包含主视窗讯息处理程式和四个子视窗讯息处理程式，其余的四个执行绪使用名为 Thread1、Thread2 等的函式，这四个执行绪负责绘制四个视窗。

我在 RNDRCTMT 程式中给出的多执行绪程式码没有使用 `_beginthread` 的第三个参数，这个参数允许一个建立另一个执行绪的执行绪在 32 位元变数中将资讯传递给其他执行绪。通常，这个变数是一个指标，而且是指向一个结构的指标，这允许原来的执行绪和新执行绪共用资讯，而不必借助於整体变数。您可以看到，在 MULTI2 中没有整体变数。

对 MULTI2 程式，我在程式开头定义了一个名为 PARAMS 的结构和一个名为 PPARAMS 的指向结构的指标，这个结构有五个栏位——视窗代号、视窗的宽度和高度、字元的高度和名为 bKill 的布林变数。最後的结构栏位允许建立执行绪告知被建立执行绪何时终止。

让我们来看一看 WndProc1，这是显示增加数序列的子视窗讯息处理程式。视窗讯息处理程式变得非常简单，唯一的区域变数是一个 PARAMS 结构。在 WM_CREATE 讯息处理期间，它设定这个结构的 hwnd 和 cyChar 栏位，呼叫 `_beginthread` 来建立一个使用 Thread1 函式的新执行绪，并传递给新执行绪一个指向该结构的指标。在 WM_SIZE 讯息处理期间，WndProc1 设定结构的 cyClient 栏位，而在 WM_DESTROY 讯息处理期间，它将 bKill 栏位设定为 TRUE。Thread1 函式通过对 `_endthread` 的呼叫而告结束。这并不是绝对必要的，因为执行绪将在退出执行绪函式之後被清除。不过，要退出一个深陷入复杂的处理程序的执行绪时，`_endthread` 是很有用的。

Thread1 函式完成在视窗上的实际绘图，并且和程式的其他四个执行绪同时执行。函式接收指向 PARAMS 结构的一个指标，并进入一个 while 回圈，不断检查 bKill 是 TRUE 还是 FALSE。如果是 FALSE，那么函式必须进行 MULTI1.C 中的 WM_TIMER 讯息处理期间所作的同样处理——格式化数字、取得装置内容代号并使用 TextOut 显示数字。

当您在 Windows 98 中执行 MULTI2 时，将会看到，视窗更新要比在 MULTI1 中快得多，这表示程式在更加有效地利用处理器的资源。在 MULTI1 和 MULTI2 之间还有另一种区别：通常，当您移动或者缩放一个视窗时，内定视窗讯息处理程式进入一种模态回圈，而视窗的所有输出都将停止。在 MULTI2 中，输出将继续。

有问题吗？

似乎 MULTI2 程式并没有达到它应该有的稳固性。我为什么会这样认为呢？让我们来看一看 MULTI2.C 中的一些多执行绪「缺陷」，以 WndProc1 和 Thread1 为例。

WndProc1 在 MULTI2 的主执行绪中执行，而 Thread1 与它同时执行，Windows 98 在这两个执行绪之间进行切换是不可预测的。假定 Thread1 正在执行，并且刚好执行了检查 PARAMS 结构的 bKill 栏位是否为 TRUE 的程式码。发现不为 TRUE，但是这之後 Windows 98 将控制权切换到主执行绪，这时使用者终止了程式，WndProc1 收到一个 WM_DESTROY 讯息并将 bKill 参数设为 TRUE。哦，这参数设定得太晚了！作业系统突然切换到 Thread1 中，而该函式会试图取得一个不存在的视窗的装置内容代号。

事实证明，这不是一个问题。Windows 98 够稳固，以致另一条执行绪呼叫的图形处理函式只是失败而已，而不会引起任何问题。

正确的多执行绪程式写作技术涉及执行绪同步的使用（尤其是临界区域的使用），我将马上加以详细地讨论。大体上，临界区域通过对 EnterCriticalSection 和 LeaveCriticalSection 的呼叫而加以界定。如果一个执行绪进入一个临界区域，那么另一个执行绪将无法再进入这个临界区域。後一个执行绪被阻挡在对 EnterCriticalSection 的呼叫上，直到第一个执行绪呼叫 LeaveCriticalSection 时为止。

在 MULTI2 中的另一个可能存在的问题是，当另外一个执行绪显示其输出时，主执行绪可能会收到一个 WM_ERASEBKGD 或 WM_PAINT 讯息。这里，使用临界区域有助於避免当两个程序试图在同一个视窗上绘图时可能导致的任何问题。但是，经验显示，Windows 98 很恰当地序列化了对图形绘制函式的存取。亦即，当另一个执行绪正在绘图的时候，一个执行绪不能在同一个视窗上绘图。

Windows 98 文件提醒说，有一种未进行图形函式序列化的情形，这就是 GDI 物件（如画笔、画刷、字体、点阵图、区域和调色盘等）的使用。有可能发生一个执行绪清除了一个物件，而另一个执行绪仍然在使用它的情况。解决这个问题的方法要求使用临界区域，或者最好不要在执行绪之间共用 GDI 物件。

Sleep 的好处

我曾经提到，我认为对一个多执行绪程式来说，最好的架构是主执行绪建立程式中的所有视窗，以及所有的视窗讯息处理程式，并处理所有的视窗讯息。其他执行绪完成背景工作或者冗长作业。

不过，假设您想在另一个执行绪中做动画。通常，Windows 中的动画是使用 WM_TIMER 讯息来实作的。如果这个执行绪没有建立视窗，那么它也不会收到这些讯息。如果没有计时器，动画又可能会执行得太快。

解决方案是 Sleep 函式。实际上，执行绪呼叫 Sleep 函式来自动暂停执行，该函式唯一的一个参数是以毫秒计的时间。Sleep 函式呼叫在指定的时间过去以前不会传回控制权。在这段时间内，执行绪被暂停，并且不会被配置给时间片段（尽管该执行绪显然仍然要求在 tick 时给予一小段的处理时间，因为系统必须确定执行绪是否应该重新开始执行）。给 Sleep 一个值为 0 的参数将导致执行绪交回它尚未使用完的时间片段。

当一个执行绪呼叫 Sleep 时，只是该执行绪被暂停指定的时间。系统仍然执行其他的执行绪，这些执行绪和暂停的执行绪可以是在同一个程序中，也可以是在另一个程序中。我在第十四章中的 SCRAMBLE 程式中使用了 Sleep 函式，以放慢画面清除的操作。

通常，您不应该在您的主执行绪中使用 Sleep 函式，因为这会减慢对讯息的处理速度，但是因为 SCRAMBLE 没有建立任何视窗，因此在那里使用 Sleep 应该没有问题。

执行绪同步

大约每年一次，在我公寓窗外的交通繁忙地段的红绿灯会停止工作。结果是造成交通的混乱，虽然轿车一般能避免撞上别的轿车，但是这些车经常挤在一起。

我用术语称两条路相交的十字路口为「临界区域」。一辆向南的车和一辆向西的车不可能同时通过一个十字路口而不撞著对方。依赖於交通流量，可以采用不同的方法来解决这个问题。对于视野清楚车辆稀少的路口，可以相信司机有处理的能力。车辆增多可能会要求一个停车号志，而更加繁忙的交通则将要求有红绿灯，红绿灯有助於协调路口的交通（当然，这些灯号必须正常工作）。

临界区域

在单工作业系统中，传统的电脑程式不需要红绿灯来帮助协调它们之间的

行为。它们在执行时似乎独占了整条路，而且也确实是这样，没有什么会干扰它们的工作。

即使在多作业系统中，大多数的程式也似乎各自独立地在执行，但是可能会发生一些问题。例如，两个程式可能会需要同时从同一个档案中读或者对同一档案进行写。在这种情况下，作业系统提供了一种共用档案和记录上锁的技术来帮助解决这个问题。

然而，在支援多执行绪的作业系统中，情况会变得混乱而且存在潜在的危险。两个或多个执行绪共用某些资料的情况并不罕见。例如，一个执行绪可以更新一个或者多个变数，而另一个执行绪可以使用这些变数。有时这会引发一个问题，有时又不会（记住作业系统将控制权从一个执行绪切换到另一个执行绪的操作，只能在机器码指令之间发生。如果只是一个整数被执行绪共用，那么对这个变数的改变通常发生在单个指令中，因此潜在的问题被最小化了）。

然而，假设执行绪共用几个变数或者资料结构。通常，这么多个变数或者结构的栏位在它们之间必须是一致的。作业系统可以在更新这些变数的程序中间中断一个执行绪，那么使用这些变数的执行绪得到的将是不一致的资料。

结果是冲突发生了，并且通常不难想像这样的错误将对程式造成怎样的破坏。我们需要的是类似於红绿灯的程式写作技术，以帮助我们对执行绪交通进行协调和同步，这就是临界区域。大体上，一个临界区域就是一块不可中断的程式码。

有四个函式用於临界区域。要使用这些函式，您必须定义一个临界区域物件，这是一个型态为 `CRITICAL_SECTION` 的整体变数。例如：

```
CRITICAL_SECTION cs ;
```

这个 `CRITICAL_SECTION` 资料型态是一个结构，但是其中的栏位只能由 Windows 内部使用。这个临界区域物件必须先被程式中的某个执行绪初始化，通过呼叫：

```
InitializeCriticalSection (&cs) ;
```

这样就建立了一个名为 `cs` 的临界区域物件。该函式的线上辅助说明包含下面的警告：「临界区域物件不能被移动或者复制，程序也不能修改该物件，但必须在逻辑上把它视为不透明的。」这句话，可以被解释为：「不要干扰它，甚至不要看它。」

当临界区域物件被初始化之後，执行绪可以通过下面的呼叫进入临界区域：

```
EnterCriticalSection (&cs) ;
```

在这时，执行绪被认为「拥有」临界区域物件。两个执行绪不可以同时拥有同一个临界区域物件，因此，如果一个执行绪进入了临界区域，那么下一个使用同一临界区域物件呼叫 `EnterCriticalSection` 的执行绪将在函式呼叫中被

暂停。只有当第一个执行绪通过下面的呼叫离开临界区域时，函式才会传回控制权：

```
LeaveCriticalSection (&cs) ;
```

这时，在 EnterCriticalSection 呼叫中被停住的那个执行绪将拥有临界区域，其函式呼叫也将传回，允许执行绪继续执行。

当临界区域不再被程式所需要时，可以通过呼叫

```
DeleteCriticalSection (&cs) ;
```

将其删除，该函式释放所有被配置来维护此临界区域物件的系统资源。

这种临界区域技术涉及「互斥」（此术语在我们继续讨论执行绪同步时将再次出现）。在任何时刻，只有一个执行绪能拥有一个临界区域。因此，一个执行绪可以进入一个临界区域，设定一个结构的栏位，然後退出临界区域。另一个使用该结构的执行绪在存取结构中的栏位之前也要先进入该临界区域，然後再退出临界区域。

注意，您可以定义多个临界区域物件，比如 cs1 和 cs2。例如，如果一个程式有四个执行绪，而前两个执行绪共用一些资料，那么它们可以使用一个临界区域物件，而另外两个执行绪共用一些其他的资料，那么它们可以使用另一个临界区域物件。

您在主执行绪中使用临界区域时应该小心。如果从属执行绪在它自己的临界区域中花费了一段很长的时间，那么它可能会将主执行绪的执行阻碍很长一段时间。从属执行绪可能只是使用临界区域复制该结构的栏位到自己的区域变数中。

临界区域的一个限制是它们只能用於在同一程序内的执行绪之间的协调。但是在某些情况下，您需要协调两个不同程序对同一资源的共用（如共用记忆体等）。在此其况下不能使用临界区域，但是可以使用一种被称为「互斥物件（mutex object）」的技术。「mutex」是个合成字，代表「mutual exclusion（互斥）」，它在这里精确地表达了我们的目的。我们想防止一个程式的执行绪在更新资料或者使用共用记忆体与其他资源时被中断。

事件信号

多执行绪通常是用於那些必须执行长时间处理的程式。我们可以将一个「大作业」定义为一个可能会违反 1/10 秒规则的程式。显然大作业包括文书处理程式中的拼写检查、资料库程式中的档案排序或者索引、试算表的重新计算、列印，甚至包括复杂的绘图。当然，迄今为止我们知道，遵循 1/10 秒规则的最好方法是将大作业放到另一个执行绪去执行。这些额外的执行绪不会建立视窗，因此它们不受 1/10 秒规则的限制。

通常希望这些额外的执行绪在完成其任务时能够通知主执行绪，或者主执行绪能够停止其他执行绪正在进行的作业。这就是我们下面将要讨论的。

BIGJOB1 程式

作为一个想像的大作业，我将使用一系列浮点运算，有时这种运算被称为「暴力的」性能测试指标。这种计算以一种间接的方式递增一个整数的值：它求一个数的平方，再对结果取平方根（得到原来的整数），然後使用 log 和 exp 函式（同样得到原来的整数），接著使用 atan 和 tan 函式（还是得到原来的整数），最後对结果加 1。

BIGJOB1 程式如程式 20-4 所示。

程式 20-4 BIGJOB1

```

BIGJOB1.C
/*-----
-
-      BIGJOB1.C -- Multithreading Demo
-
-                                     (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include <math.h>
#include <process.h>

#define REP                        1000000

#define STATUS_READY              0
#define STATUS_WORKING            1
#define STATUS_DONE               2

#define WM_CALC_DONE              (WM_USER + 0)
#define WM_CALC_ABORTED          (WM_USER + 1)

typedef struct
{
    HWND hwnd ;
    BOOL bContinue ;
}
PARAMS, *PPARAMS ;
LRESULT APIENTRY WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    szCmdLine, int iCmdShow)
{
    static TCHAR                szAppName[] = TEXT ("BigJob1") ;

```



```

    HWND                hwnd ;
    MSG                msg ;
    WNDCLASS            wndclass ;
    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Multithreading Demo"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void Thread (PVOID pvoid)
{
    double          A = 1.0 ;
    INT             i ;
    LONG            lTime ;
    volatile        PPARAMS pparams ;

    pparams = (PPARAMS) pvoid ;
    lTime = GetCurrentTime () ;
    for (i = 0 ; i < REP && pparams->bContinue ; i++)

```

```

        A = tan (atan (exp (log (sqrt (A * A))))) + 1.0 ;

if (i == REP)
{
    lTime = GetCurrentTime () - lTime ;
    SendMessage (pparams->hwnd, WM_CALC_DONE, 0, lTime) ;
}
else
    SendMessage (pparams->hwnd, WM_CALC_ABORTED, 0, 0) ;
_endthread () ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      INT      iStatus ;
    static      LONG     lTime ;
    static      PARAMS    params ;
    static      TCHAR *   szMessage[] = { TEXT ("Ready (left mouse button
begins)"),
        TEXT ("Working (right mouse button ends)"),
        TEXT ("%d repetitions in %ld msec") } ;

    HDC          hdc ;
    PAINTSTRUCT   ps ;
    RECT          rect ;
    TCHAR         szBuffer[64] ;

    switch (message)
    {
    case WM_LBUTTONDOWN:
        if (iStatus == STATUS_WORKING)
        {
            MessageBeep (0) ;
            return 0 ;
        }

        iStatus = STATUS_WORKING ;

        params.hwnd = hwnd ;
        params.bContinue = TRUE ;

        _beginthread (Thread, 0, 耗s) ;

        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

    case WM_RBUTTONDOWN:
        params.bContinue = FALSE ;
        return 0 ;
    }
}

```

```
case WM_CALC_DONE:
    lTime = lParam ;
    iStatus = STATUS_DONE ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_CALC_ABORTED:
    iStatus = STATUS_READY ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    wsprintf (szBuffer, szMessage[iStatus], REP, lTime) ;
    DrawText (hdc, szBuffer, -1, &rect,
        DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这是一个相当简单的程式，但是我认为您将看到它如何展示在多执行绪程式中完成大作业的通用方法。为了使用 BIGJOB1 程式，在视窗的显示区域中按下鼠标左键，从而开始暴力的性能测试计算的 1,000,000 次重复，这在一台 300MHz 的 Pentium II 机器上将花费 2 秒。当完成计算时，花费的时间将显示在视窗上。当正在进行计算时，您可以通过在显示区域中按下鼠标右键来终止它。

让我们来看一看这是如何实作的：

视窗讯息处理程式拥有了一个被叫做 iStatus 的静态变数（该变数可以被设定为在程式开始处定义三个常数之一，常数以 STATUS 为字首），该变数表示程式是否准备好进行一次计算，是否正在进行一次计算，或者是否完成了计算。程式在 WM_PAINT 讯息处理期间使用 iStatus 变数在显示区域的中央显示一个适当的字符串。

视窗讯息处理程式还拥有一个静态结构（型态为 PARAMS，也定义在程式的顶部），该结构是在视窗讯息处理程式和其他执行绪之间的共用资料。结构只

有两个栏位——hwnd (程式视窗的代号) 和 bContinue, 这是一个布林变数, 用於指示执行绪是否继续计算或者停止。

当您在显示区域中按下鼠标左键时, 视窗讯息处理程式将 iStatus 变数设为 STATUS_WORKING, 并设定 PARAMS 结构中的两个栏位。结构的 hwnd 栏位被设定为视窗代号, 当然, bContinue 被设定为 TRUE。

然後视窗程序呼叫 _beginthread 函式。执行绪函式 Thread 以呼叫 GetCurrentTime 开始, GetCurrentTime 取得以毫秒计的 Windows 启动以来已经执行了的时间。然後它进入一个 for 回圈, 重复 1,000,000 次的暴力测试计算。还要注意, 如果 bContinue 被设为了 FALSE, 那么执行绪将退出回圈。

在 for 回圈之後, 执行绪函式检查它是否确实完成了 1,000,000 次计算。如果是, 那么它再次呼叫 GetCurrentTime 获得所经过的时间, 然後使用 SendMessage 向视窗讯息处理程式发送一个由程式定义的 WM_USER_DONE 讯息, 并以经过的时间作为 lParam 参数。如果计算是在未完成之前被终止的 (即, 如果在回圈期间 PARAMS 结构的 bContinue 栏位变为 FALSE), 那么执行绪将发送给视窗讯息处理程式一个 WM_USER_ABORTED 讯息。然後, 执行绪通过呼叫 _endthread 正常地结束。

在视窗讯息处理程式中, 当您在显示区域中按下鼠标右键时, PARAMS 结构的 bContinue 栏位被设为 FALSE。这是如何在完成计算之前结束计算的方法。

注意 Thread 中的 pparams 变数定义为 volatile, 这种型态限定字向编译器指出变数可能会在实际的程式叙述外被修改 (例如被另一个执行绪)。否则, 最佳化的编译器会假设 pparams->bContinue 不能被 for 回圈内的程式码修改, 没有必要在每层回圈中检查变数。volatile 关键字防止这样的最佳化进行。

视窗讯息处理程式处理 WM_USER_DONE 讯息时, 首先储存经过的时间。对 WM_USER_DONE 和 WM_USER_ABORTED 讯息的处理都是透过对 InvalidateRect 的呼叫产生 WM_PAINT 讯息并在显示区域显示一个新的字串。

提供一个方法 (如结构中的 bContinue 栏位) 允许执行绪正常终止, 通常是一个好主意。KillThread 函式只有在正常终止执行绪比较困难时才应该使用, 原因是执行绪可以配置资源, 如记忆体等。如果当执行绪终止时没有释放所配置的记忆体, 那么记忆体将仍然是被配置了的。执行绪不是程序: 所配置的资源在一个程序的所有执行绪之间是共用的, 因此当执行绪终止时, 资源不会被自动释放。好的程式结构要求一个执行绪释放由它配置的所有资源。

您还应该知道当第二个执行绪仍在执行时, 可以建立第三个执行绪。如果 Windows 在 SendMessage 呼叫和 _endthread 呼叫之间, 将控制权从第二个执行绪切换到第一个执行绪, 那么视窗讯息处理程式就可能回应鼠标按键而建立一

个新的执行绪，从而出现了上述的情况。这不是什么问题，但是如果这对您自己的应用来说是一个问题的话，那么您可能会考虑使用临界区域来避免执行绪之间的冲突。

事件物件

BIGJOB1 在每次需要执行暴力测试计算时，就建立一个执行绪。执行绪在完成计算之後自动终止。

另一种可用的方法是在程式的整个生命周期内保持执行绪的执行，但是只在必要时才启动它。这是一个应用事件物件的理想情况。

事件物件可以是「有信号的」（也称为「被设立的」）或「没信号的」（也称为「被重置的」）。您可以通过下面呼叫来建立事件物件：

```
hEvent = CreateEvent (&sa, fManual, fInitial, pszName) ;
```

第一个参数（指向一个 SECURITY_ATTRIBUTES 结构的指标）和最後一个参数（一个事件物件的名字）只有在事件物件被多个程序共用时才有意义。在同一程序中，这些参数通常被设定为 NULL。如果您希望事件物件被初始化为有信号的，那么将 fInitial 参数设定为 TRUE。而如果希望事件物件被初始化为无信号的，则将 fInitial 参数设定为 FALSE。稍後，我将简短地描述 fManual 参数。

要设立一个现存的事件物件，呼叫

```
SetEvent (hEvent) ;
```

要重置一个事件物件，呼叫

```
ResetEvent (hEvent) ;
```

一个程式通常呼叫：

```
WaitForSingleObject (hEvent, dwTimeOut) ;
```

并且将第二个参数设定为 INFINITE。如果事件物件目前是被设立的，那么函式将立即传回，否则，函式将暂停执行绪直到事件物件被设立。如果您将第二个参数设定为一个以毫秒计的超时时间值，这样函式也可能在事件物件被设立之前传回。

如果最初的 CreateEvent 呼叫的 fManual 参数被设定为 FALSE，那么事件物件将在 WaitForSingleObject 函式传回时自动重置。这种功能特性通常使得事件物件没有必要使用 ResetEvent 函式。

现在，我们可以来看一看程式 20-5 所示的 BIGJOB2.C 程式。

程式 20-5 BIGJOB2

```
BIGJOB2.C
```

```
/*-----  
--  
BIGJOB2.C -- Multithreading Demo  
  
(c) Charles Petzold, 1998
```

```

-----
-*/

#include <windows.h>
#include <math.h>
#include <process.h>

#define REP                                1000000

#define STATUS_READY                      0
#define STATUS_WORKING                    1
#define STATUS_DONE                       2

#define WM_CALC_DONE                      (WM_USER + 0)
#define WM_CALC_ABORTED                  (WM_USER + 1)

typedef struct
{
    HWND          hwnd ;
    HANDLE         hEvent ;
    BOOL          bContinue ;
}
PARAMS, *PPARAMS ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    static TCHAR    szAppName[] = TEXT ("BigJob2") ;
    HWND           hwnd ;
    MSG            msg ;
    WNDCLASS        wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
        return 0 ;
    }
}

```

```

}

hwnd = CreateWindow (  szAppName, TEXT ("Multithreading Demo"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

void Thread (PVOID pvoid)
{
    double          A = 1.0 ;
    INT              i ;
    LONG             lTime ;
    volatile         PPARAMS pparams ;

    pparams = (PPARAMS) pvoid ;
    while (TRUE)
    {
        WaitForSingleObject (pparams->hEvent, INFINITE) ;
        lTime = GetCurrentTime () ;
        for (i = 0 ; i < REP && pparams->bContinue ; i++)
            A = tan (atan (exp (log (sqrt (A * A))))) + 1.0 ;
        if (i == REP)
        {
            lTime = GetCurrentTime () - lTime ;
            PostMessage (pparams->hwnd, WM_CALC_DONE, 0, lTime) ;
        }
        else
            PostMessage (pparams->hwnd, WM_CALC_ABORTED, 0, 0) ;
    }
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static          HANDLE      hEvent ;
    static          INT         iStatus ;

```

```
static      LONG      lTime ;
static      PARAMS    params ;
static      TCHAR *    szMessage[] = { TEXT ("Ready (left mouse button
begins)"),
                                TEXT ("Working (right mouse button ends)"),
                                TEXT ("%d repetitions in %ld msec") } ;

HDC          hdc ;
PAINTSTRUCT  ps ;
RECT         rect ;
TCHAR        szBuffer[64] ;

switch (message)
{
case WM_CREATE:
    hEvent = CreateEvent (NULL, FALSE, FALSE, NULL) ;

    params.hwnd = hwnd ;
    params.hEvent = hEvent ;
    params.bContinue = FALSE ;

    _beginthread (Thread, 0, 耗s) ;

    return 0 ;

case WM_LBUTTONDOWN:
    if (iStatus == STATUS_WORKING)
    {
        MessageBeep (0) ;
        return 0 ;
    }
    iStatus = STATUS_WORKING ;
    params.bContinue = TRUE ;

    SetEvent (hEvent) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_RBUTTONDOWN:
    params.bContinue = FALSE ;
    return 0 ;

case WM_CALC_DONE:
    lTime = lParam ;
    iStatus = STATUS_DONE ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
```



```

case WM_CALC_ABORTED:
    iStatus = STATUS_READY ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    wsprintf ( szBuffer, szMessage[iStatus], REP, lTime) ;
    DrawText ( hdc, szBuffer, -1, &rect,
               DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

处理 WM_CREATE 讯息时，视窗讯息处理程式首先建立一个初始化为没信号的自动重置事件物件，然後建立执行绪。

Thread 函式进入一个无限的 while 回圈，在回圈开始时首先呼叫 WaitForSingleObject（注意 PARAMS 结构包括一个包含事件物件代号的栏位）。因为事件被初始化为重置的，所以执行绪的执行被阻挡在函式呼叫中。按下滑鼠左键将导致视窗程序呼叫 SetEvent，这将释放由 WaitForSingleObject 呼叫产生的第二个执行绪，并开始暴力测试计算。当计算完之後，执行绪再次呼叫 WaitForSingleObject，但是由於第一次呼叫已经使事件物件重置，因此，执行绪将被暂停，直到再次按下滑鼠。

在其他方面，程式几乎和 BIGJOB1 完全一样。

执行绪区域储存空间 (TLS)

多执行绪程式中的整体变数（以及任何被配置的记忆体）被程式中的所有执行绪共用。在一个函式中的局部静态变数也被使用函式的所有执行绪共用。一个函式中的局部动态变数是唯一於各个执行绪的，因为它们被储存在堆叠上，而每个执行绪有它自己的堆叠。

对各个执行绪唯一的持续性储存空间有存在的必要。例如，我在本章前面提到过的 C 中的 strtok 函式要求这种型态的储存空间。不幸的是，C 语言不支

援这类储存空间。但是 Windows 中提供了四个函式，它们实作了一种技术来做到这一点，并且 Microsoft 对 C 的扩充语法也支援它，这就叫做执行绪区域储存空间。

下面是 API 工作的方法：

首先，定义一个包含需要唯一於执行绪的所有资料的结构，例如：

```
typedef struct
{
    int a ;
    int b ;
}
DATA, * PDATA ;
```

主执行绪呼叫 TlsAlloc 获得一个索引值：

```
dwTlsIndex = TlsAlloc () ;
```

这个值可以储存在一个整体变数中或者通过参数结构传递给执行绪函式。

执行绪函式首先为该资料结构配置记忆体，并使用上面所获得的索引值呼叫 TlsSetValue：

```
TlsSetValue (dwTlsIndex, GlobalAlloc (GPTR, sizeof (DATA)) ;
```

该函式将一个指标和某个执行绪及某个执行绪索引相关联。现在，任何需要使用这个指标的函式（包括最初的执行绪函式本身）都可以包含如下所示的程式码：

```
PDATA pdata ;
...
pdata = (PDATA) TlsGetValue (dwTlsIndex) ;
```

现在函式可以设定或者使用 pdata->a 和 pdata->b 了。在执行绪函式终止以前，它释放配置的记忆体：

```
GlobalFree (TlsGetValue (dwTlsIndex)) ;
```

当使用该资料的所有执行绪都终止之时，主执行绪将释放索引：

```
TlsFree (dwTlsIndex) ;
```

这个程序刚开始可能令人有些迷惑，因此如果能看一看如何实作执行绪区域储存空间可能会有帮助（我不知道 Windows 实际上是如何实作的，但下面的方案是可能的）。首先，TlsAlloc 可能只是配置一块记忆体（长度为 0）并传回一个索引值，即指向这块记忆体的一个指标。每次使用该索引呼叫 TlsSetValue 时，通过重新配置将记忆体块增大 8 个位元组。在这 8 个位元组中储存的是呼叫函式的执行绪 ID（通过 GetCurrentThreadId 来获得）以及传递给 TlsSetValue 函式的指标。TlsSetValue 简单地使用执行绪 ID 来搜寻作业系统管理的执行绪区域储存空间位址表，然後传回指标。TlsFree 将释放记忆体块。所以您看，这可能是一件容易得可以由您自己来实作的事情。不过，既然已经有工具为您做好了这些工作，那也不错。

Microsoft 对 C 的扩充功能使这件工作更加容易。只要在要对每个执行绪都保留不同内容的变数前加上__declspec (thread)就好了。对于任何函式的外部静态变数, 则为:

```
__declspec (thread) int iGlobal = 1 ;
```

对于函式内部的静态变数, 则为:

```
__declspec (thread) static int iLocal = 2 ;
```

第二十一章 动态连结程式库

动态连结程式库（也称为 DLL）是 Microsoft Windows 最重要的组成要素之一。大多数与 Windows 相关的磁碟档案如果不是程式模组，就是动态连结程式。迄今为止，我们都是开发 Windows 应用程序；现在是尝试编写动态连结程式库的时候了。许多您已经学会的编写应用程序的规则同样适用于编写这些动态连结程式库模组，但也有一些重要的不同。

动态连结程式库的基本知识

正如前面所看到的，Windows 应用程序是一个可执行档案，它通常建立一个或几个视窗，并使用讯息回圈接收使用者输入。通常，动态连结程式库并不能直接执行，也不接收讯息。它们是一些独立的档案，其中包含能被程式或其他 DLL 呼叫来完成一定作业的函式。只有在其他模组呼叫动态连结程式库中的函式时，它才发挥作用。

所谓「动态连结」，是指 Windows 把一个模组中的函式呼叫连结到动态连结程式库模组中的实际函式上的程序。在程式开发中，您将各种目的模组（.OBJ）、执行时期程式库（.LIB）档案，以及经常是已编译的资源（.RES）档案连结在一起，以便建立 Windows 的 .EXE 档案，这时的连结是「静态连结」。动态连结与此不同，它发生在执行时期。

KERNEL32.DLL、USER32.DLL 和 GDI32.DLL、各种驱动程序档案如 KEYBOARD.DRV、SYSTEM.DRV 和 MOUSE.DRV 和视讯及印表机驱动程序都是动态连结程式库。这些动态连结程式库能被所有 Windows 应用程序使用。

有些动态连结程式库（如字体档案等）被称为「纯资源」。它们只包含资料（通常是资源的形式）而不包含程式码。由此可见，动态连结程式库的目的之一就是提供能被许多不同的应用程序所使用的函式和资源。在一般的作业系统中，只有作业系统本身才包含其他应用程序能够呼叫来完成某一作业的常式。在 Windows 中，一个模组呼叫另一个模组函式的程序被推广了。结果使得编写一个动态连结程式库，也就是在扩充 Windows。当然，也可认为动态连结程式库（包括构成 Windows 的那些动态连结程式库常式）是对使用者程式的扩充。

尽管一个动态连结程式库模组可能有其他副档名（如 .EXE 或 .FON），但标准副档名是 .DLL。只有带 .DLL 副档名的动态连结程式库才能被 Windows 自动载入。如果档案有其他副档名，则程式必须另外使用 LoadLibrary 或者 LoadLibraryEx 函式载入该模组。

您通常会发现，动态连结程式库在大型应用程序中最有意义。例如，假设要为 Windows 编写一个由几个不同的程式组成的大型财务套装软体，就会发现这些应用程序会使用许多共同的常式。可以把这些公共常式放入一个一般性的目的码程式库（带.LIB 副档名）中，并在使用 LINK 静态连结时把它们加入各程式模组中。但这种方法是很浪费的，因为套装软体中的每个程式都包含与公共常式相同的程式码。而且，如果修改了程式库中的某个常式，就要重新连结使用此常式的所有程式。然而，如果把这些公共常式放到称为 ACCOUNT.DLL 的动态连结程式库中，就可解决这两个问题。只有动态连结程式库模组才包含所有程式都要用到的常式。这样能为储存档案节省磁碟空间，并且在同时执行多个应用程序时节省记忆体，而且，可以修改动态连结程式库模组而不用重新连结各个程式。

动态连结程式库实际上是可以独立存在的。例如，假设您编写了一系列 3D 绘图常式，并把它们放入名为 GDI3.DLL 的 DLL 中。如果其他软体发展者对此程式库很感兴趣，您就可以授权他们将其加入他们的图形程式中。使用多个这样的图形程式的使用者只需要一个 GDI3.DLL 档案。

程式库：一词多义

动态连结程式库有著令人困惑的印象，部分原因是由於「程式库」这个词被放在几种不同的用语之後。除了动态连结程式库之外，我们也用它来称呼「目的码程式库」或「引用程式库」。

目的码程式库是带.LIB 副档名的档案。在使用连结程式进行静态连结时，它的程式码就会加到程式的.EXE 档案中。例如，在 Microsoft Visual C++中，连同程式连结的一般 C 执行目的码程式库被称为 LIBC.LIB。

引用程式库是目的码程式库档案的一种特殊形式。像目的码程式库一样，引用程式库有.LIB 副档名，并且被连结器用来确定程式码中的函式呼叫来源。但引用程式库不含程式码，而是为连结程式提供资讯，以便在.EXE 档案中建立动态连结时要用到的重定位表。包含在 Microsoft 编译器中的 KERNEL32.LIB、USER32.LIB 和 GDI32.LIB 档案是 Windows 函式的引用程式库。如果一个程式呼叫 Rectangle 函式，Rectangle 将告诉 LINK，该函式在 GDI32.DLL 动态连结程式库中。该资讯被记录在.EXE 档案中，使得程式执行时，Windows 能够和 GDI32.DLL 动态连结程式库进行动态连结。

目的码程式库和引用程式库只用在程式开发期间使用，而动态连结程式库在执行期间使用。当一个使用动态连结程式库的程式执行时，该动态连结程式库必须在磁片上。当 Windows 要执行一个使用了动态连结程式库的程式而需要

载入该程式库时，动态连结程式库档案必须储存在含有该 .EXE 程式的目录下、目前的目录下、Windows 系统目录下、Windows 目录下，或者是在通过 MS-DOS 环境中的 PATH 可以存取到的目录下（Windows 会按顺序搜索这些目录）。

一个简单的 DLL

虽然动态连结程式库的整体概念是它们可以被多个應用程式所使用，但您通常最初设计的动态连结程式库只与一个應用程式相联系，可能是一个「测试」程式在使用 DLL。

下面就是我们要做的。我们建立一个名为 EDRLIB.DLL 的 DLL。档案名中的「EDR」代表「简便的绘图常式 (easy drawing routines)」。这里的 EDRLIB 只含有一个函式（名称为 EdrCenterText），但是您还可以将應用程式中其他简单的绘图函式添加进去。應用程式 EDRTTEST.EXE 将通过呼叫 EDRLIB.DLL 中的函式来利用它。

要做到这一点，需要与我们以前所做的略有不同的方法，也包括 Visual C++ 中我们没有看过的特性。在 Visual C++ 中「工作空间 (workspaces)」和「专案 (projects)」不同。专案通常与建立的应用程式 (.EXE) 或者动态连结程式库 (.DLL) 相联系。一个工作空间可以包含一个或多个专案。迄今为止，我们所有的工作空间都只包含一个专案。我们现在就建立一个包含两个专案的工作空间 EDRTTEST——一个用於建立 EDRTTEST.EXE，而另一个用於建立 EDRLIB.DLL，即 EDRTTEST 使用的动态连结程式库。

现在就开始。在 Visual C++ 中，从「File」功能表选择「New」，然後选择「Workspaces」页面标签。（我们以前从来没有选择过。）在「Location」栏选择工作空间要储存的目录，然後在「Workspace Name」栏输入「EDRTTEST」，按 Enter 键。

这样就建立了一个空的工作空间。Developer Studio 还建立了一个名为 EDRTTEST 的子目录，以及工作空间档案 EDRTTEST.DSW（就像两个其他档案）。

现在让我们在此工作空间里建立一个专案。从「File」功能表选择「New」，然後选择「Projects」页面标签。尽管过去您选择「Win32 Application」，但现在「Win32 Dynamic-Link Library」。另外，单击单选按钮「Add To Current Workspace」，这使得此专案是「EDRTTEST」工作空间的一部分。在「Project Name」栏输入 EDRLIB，但先不要按「OK」按钮。当您在 Project Name 栏输入 EDRLIB 时，Visual C++ 将改变「Location」栏，以显示 EDRLIB 作为 EDRTTEST 的一个子目录。这不是我们要的，所以接著在「Location」栏删除 EDRLIB 子目录以便专案建立在 EDRTTEST 目录。现在按「OK」。萤幕将显示一个对话方块，询问您建

立什么型态的 DLL。选择「An Empty DLL Project」，然後按「Finish」。Visual C++ 将建立一个专案档案 EDRLIB.DSP 和一个构造档案 EDRLIB.MAK (如果「Tools Options」对话方块的 Build 页面标签中选择了「Export Makefile」选项)。

现在您已经在此专案中添加了一对档案。从「File」功能表选择「New」，然後选择「Files」页面标签。选择「C/C++ Header File」，然後输入档案名 EDRLIB.H。输入程式 21-1 所示的档案 (或者从本书光碟中复制)。再次从「File」功能表中选择「New」，然後选择「Files」页面标签。这次选择「C++ Source File」，然後输入档案名 EDRLIB.C。继续输入程式 21-1 所示的程式。

程式 21-1 EDRLIB 动态连结程式库

```
EDRLIB.H
/*-----
    EDRLIB.H header file
-----
*/

#ifdef    __cplusplus
#define    EXPORT extern "C" __declspec (dllexport)
#else
#define    EXPORT __declspec (dllexport)
#endif

EXPORT    BOOL CALLBACK EdrCenterTextA (HDC, PRECT, PCSTR) ;
EXPORT    BOOL CALLBACK EdrCenterTextW (HDC, PRECT, PCWSTR) ;

#ifdef    UNICODE
#define    EdrCenterText EdrCenterTextW
#else
#define    EdrCenterText EdrCenterTextA
#endif
EDRLIB.C
/*-----
-
    EDRLIB.C -- Easy Drawing Routine Library module
                                     (c) Charles Petzold, 1998
-----
-*/
#include windows.h>
#include "edrlib.h"

int WINAPI DllMain (    HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
    return TRUE ;
}

EXPORT BOOL CALLBACK EdrCenterTextA (    HDC hdc, PRECT prc, PCSTR pString)
```

```

{
    int iLength ;
    SIZE size ;

    iLength = lstrlenA (pString) ;
    GetTextExtentPoint32A (hdc, pString, iLength, &size) ;
    return TextOutA (hdc, ( prc->right - prc->left - size.cx) / 2,
                     (    prc->bottom - prc->top - size.cy) / 2,
                     pString, iLength) ;
}

EXPORT BOOL CALLBACK EdrCenterTextW (HDC hdc, PRECT prc, PCWSTR pString)
{
    int iLength ;
    SIZE size ;

    iLength = lstrlenW (pString) ;
    GetTextExtentPoint32W (hdc, pString, iLength, &size) ;
    return TextOutW (hdc, (    prc->right - prc->left - size.cx) / 2,
                     (    prc->bottom - prc->top - size.cy) / 2,
                     pString, iLength) ;
}

```

这里您可以按 Release 设定, 或者也可以按 Debug 设定来建立 EDRLIB.DLL。之後, RELEASE 和 DEBUG 目录将包含 EDRLIB.LIB (即动态连结程式库的引用程式库) 和 EDRLIB.DLL (动态连结程式库本身)。

纵观全书, 我们建立的所有程式都可以根据 UNICODE 识别字来编译成使用 Unicode 或非 Unicode 字串的程式码。当您建立一个 DLL 时, 它应该包括处理字元和字串的 Unicode 和非 Unicode 版的所有函式。因此, EDRLIB.C 就包含函式 EdrCenterTextA (ANSI 版) 和 EdrCenterTextW (宽字元版)。EdrCenterTextA 定义为带有参数 PCSTR (指向 const 字串的指标), 而 EdrCenterTextW 则定义为带有参数 PCWSTR (指向 const 宽字串的指标)。EdrCenterTextA 函式将呼叫 lstrlenA、GetTextExtentPoint32A 和 TextOutA。EdrCenterTextW 将呼叫 lstrlenW、GetTextExtentPoint32W 和 TextOutW。如果定义了 UNICODE 识别字, 则 EDRLIB.H 将 EdrCenterText 定义为 EdrCenterTextW, 否则定义为 EdrCenterTextA。这样的做法很像 Windows 表头档案。

EDRLIB.H 也包含函式 DllMain, 取代了 DLL 中的 WinMain。此函式用於执行初始化和未初始化 (deinitialization), 我将在下一节讨论。我们现在所需要的就是从 DllMain 传回 TRUE。

在这两个档案中, 最後一点神秘之处就是定义了 EXPORT 识别字。DLL 中应用程式使用的函式必须是「输出 (exported)」的。这跟税务或者商业制度无

关，只是确保函式名添加到 EDRLIB.LIB 的一个关键字（以便连结程式在连结使用此函式的应用程式时，能够解析出函式名称），而且该函式在 EDRLIB.DLL 中也是看得到的。EXPORT 识别字包括储存方式限定词 `__declspec (dllexport)` 以及在表头档案按 C++ 模式编译时附加的「C」。这将防止编译器使用 C++ 的名称轧压规则 (name mangling) 来处理函式名称，使 C 和 C++ 程式都能使用这个 DLL。

程式库入口 / 出口点

当动态连结程式库首次启动和结束时，我们呼叫了 DllMain 函式。DllMain 的第一个参数是程式库的执行实体代号。如果您的程式库需要使用执行实体代号（诸如 DialogBox）的资源，那么您应该将 hInstance 储存为一个整体变数。DllMain 的最后一个参数由系统保留。

fdwReason 参数可以是四个值之一，说明为什么 Windows 要呼叫 DllMain 函式。在下面的讨论中，请记住一个程式可以被载入多次，并在 Windows 下一起执行。每当一个程式载入时，它都被认为是一个独立的程序 (process)。

fdwReason 的一个值 DLL_PROCESS_ATTACH 表示动态连结程式库被映射到一个程序的位址空间。程式库可以根据这个线索进行初始化，为以后来自该程序的请求提供服务。例如，这类初始化可能包括记忆体配置。在一个程序的生命周期内，只有一次对 DllMain 的呼叫以 DLL_PROCESS_ATTACH 为参数。使用同一 DLL 的其他任何程序都将导致另一个使用 DLL_PROCESS_ATTACH 参数的 DllMain 呼叫，但这是对新程序的呼叫。

如果初始化成功，DllMain 应该传回一个非 0 值。传回 0 将导致 Windows 不执行该程式。

当 fdwReason 的值为 DLL_PROCESS_DETACH 时，意味著程序不再需要 DLL 了，从而提供给程式库自己清除自己的机会。在 32 位元的 Windows 下，这种处理并不是严格必须的，但这是一种良好的程式写作习惯。

类似地，当以 DLL_THREAD_ATTACH 为 fdwReason 参数呼叫 DllMain 时，意味著某个程序建立了一个新的执行绪。当执行绪中止时，Windows 以 DLL_THREAD_DETACH 为 fdwReason 参数呼叫 DllMain。请注意，如果动态连结程式库是在执行绪被建立之后和一个程序连结的，那么可能会得到一个没有事先对应一个 DLL_THREAD_ATTACH 呼叫的 DLL_THREAD_DETACH 呼叫。

当使用一个 DLL_THREAD_DETACH 参数呼叫 DllMain 时，执行绪仍然存在。动态连结程式库甚至可以在这个程序期间发送执行绪讯息。但是它不应该使用 PostMessage，因为执行绪可能在此讯息被处理到之前就已经退出执行了。

测试程式

现在让我们在 EDRTEST 工作空间里建立第二个专案，程式名称为 EDRTEST，而且使用 EDRLIB.DLL。在 Visual C++ 中载入 EDRTEST 工作空间时，请从「File」功能表选择「New」，然後在「New」对话方块中选择「Projects」页面标签。这次选择「Win32 Application」，并确保选中了「Add To Current Workspace」按钮。输入专案名称 EDRTEST。再在「Locations」栏删除第二个 EDRTEST 子目录。按下「OK」，然後在下一个对话方块选择「An Empty Project」，按「Finish」。

从「File」功能表再次选择「New」。选择「Files」页面标签然後选择「C++ Source File」。确保「Add To Project」清单方块显示「EDRTEST」而不是「EDRLIB」。输入档案名称 EDRTEST.C，然後输入程式 21-2 所示的程式。此程式用 EdrCenterText 函式将显示区域中的字串居中对齐。

程式 21-2 EDRTEST

```
EDRTEST.C
/*-----
-
    EDRTEST.C -- Program using EDRLIB dynamic-link library
                                   (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "edrlib.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName[] = TEXT ("StrProg") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance   = hInstance ;
    wndclass.hIcon        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
```

```
wndclass.lpszClassName = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                  szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("DLL Demonstration Program"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

LRESULT CALLBACK WndProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    HDC          hdc ;
    PAINTSTRUCT  ps ;
    RECT         rect ;

    switch (message)
    {
    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        GetClientRect (hwnd, &rect) ;

        EdrCenterText (    hdc, &rect,
        TEXT ("This string was displayed by a DLL")) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY:
        PostQuitMessage (0) ;
```

```
        return 0 ;  
    }  
    return DefWindowProc (hwnd, message, wParam, lParam) ;  
}
```

注意，为了定义 EdrCenterText 函式，EDRTEST.C 包括 EDRLIB.H 表头档案，此函式将在 WM_PAINT 讯息处理期间呼叫。

在编译此程式之前，您可能希望做以下几件事。首先，在「Project」功能表选择「Select Active Project」。这时您将看到「EDRLIB」和「EDRTEST」，选择「EDRTEST」。在重新编译此工作空间时，您真正要重新编译的是程式。另外，在「Project」功能表中，选择「Dependencies」，在「Select Project To Modify」清单方块中选择「EDRTEST」。在「Dependent On The Following Project(s)」列表选中「EDRLIB」。此操作的意思是：EDRTEST 需要 EDRLIB 动态连结程式库。以後每次重新编译 EDRTEST 时，如果必要的话，都将在编译和连结 EDRTEST 之前重新编译 EDRLIB。

从「Project」功能表选择「Settings」，单击「General」标签。当您在左边的窗格中选择「EDRLIB」或者「EDRTEST」专案时，如果设定为「Win32 Release」，则显示在右边窗格中的「Intermediate Files」和「Output Files」将位於 RELEASE 目录；如果设定为「Win32 Debug」，则位於 DEBUG 目录。如果不是，请按此修改。这样可确保 EDRLIB.DLL 与 EDRTEST.EXE 在同一个目录中，而且程式在使用 DLL 时也不会产生问题。

在「Project Setting」对话方块中依然选中「EDRTEST」，单击「C/C++」页面标签。按本书的惯例，在「Preprocessor Definitions」中，将「UNICODE」添加到 Debug 设定。

现在您就可以在「Debug」或「Release」设定中重新编译 EDRTEST.EXE 了。必要时，Visual C++ 将首先编译和连结 EDRLIB。RELEASE 和 DEBUG 目录都包含 EDRLIB.LIB（引用程式库）和 EDRLIB.DLL。当 Developer Studio 连结 EDRTEST 时，将自动包含引用程式库。

了解 EDRTEST.EXE 档案中不包含 EdrCenterText 程式码很重要。事实上，要证明执行了 EDRLIB.DLL 档案和 EdrCenterText 函式很简单：执行 EDRTEST.EXE 需要 EDRLIB.DLL。

执行 EDRTEST.EXE 时，Windows 按外部程式库模组执行固定的函式。其中许多函式都在一般 Windows 动态连结程式库中。但 Windows 也看到程式从 EDRLIB 呼叫了函式，因此 Windows 将 EDRLIB.DLL 档案载入到记忆体，然後呼叫 EDRLIB 的初始化常式。EDRTEST 呼叫 EdrCenterText 函式是动态连结到 EDRLIB 中函式的。

在 EDRTTEST.C 原始码档案中包含 EDRLIB.H 与包含 WINDOWS.H 类似。连结 EDRLIB.LIB 与连结 Windows 引用程式库 (例如 USER32.LIB) 类似。当您的程式执行时, 它连结 EDLIB.DLL 的方式与连结 USER32.DLL 的方式相同。恭喜您! 您已经扩展了 Windows 的功能!

在继续之前, 我还要对动态连结程式库多说明一些:

首先, 虽然我们将 DLL 作为 Windows 的延伸, 但它也是您的應用程式的延伸。DLL 所完成的每件工作对于應用程式来说都是應用程式所交代要完成的。例如, 應用程式拥有 DLL 配置的全部记忆体、DLL 建立的全部视窗以及 DLL 打开的所有档案。多个應用程式可以同时使用同一个 DLL, 但在 Windows 下, 这些應用程式不会相互影响。

多个程序能够共用一个动态连结程式库中相同的程式码。但是, DLL 为每个程序所储存的资料都不同。每个程序都为 DLL 所使用的全部资料配置了自己的位址空间。我们将在下以节看到, 共用记忆体需要额外的工作。

在 DLL 中共用记忆体

令人兴奋的是, Windows 能够将同时使用同一个动态连结程式库的應用程式分开。不过, 有时却不太令人满意。您可能希望写一个 DLL, 其中包含能够被不同應用程式或者同一个程式的不同常式所共用的记忆体。这包括使用共用记忆体。共用记忆体实际上是一种记忆体映射档案。

让我们测试一下, 这项工作是如何在程式 STRPROG (「字串程式 (string program)」) 和动态连结程式库 STRLIB (「字串程式库 (string library)」) 中完成的。STRLIB 有三个输出函式被 STRPROG 呼叫, 我们只对此感兴趣, STRLIB 中的一个函式使用了在 STRPROG 定义的 callback 函式。

STRLIB 是一个动态连结程式库模组, 它储存并排序了最多 256 个字串。在 STRLIB 中, 这些字串均为大写, 并由共用记忆体维护。利用 STRLIB 的三个函式, STRPROG 能够添加字串、删除字串以及从 STRLIB 获得目前的所有字串。STRPROG 测试程式有两个功能表项 (「Enter」和「Delete」), 这两个功能表项将启动不同的对话方块来添加或删除字串。STRPROG 在其显示区域列出目前储存在 STRLIB 中的所有字串。

下面这个函式在 STRLIB 定义, 它将一个字串添加到 STRLIB 的共用记忆体。

```
EXPORT BOOL CALLBACK AddString (pStringIn)
```

参数 pStringIn 是字串的指标。字串在 AddString 函式中变成大写。如果在 STRLIB 的列表中有一个相同的字串, 那么此函式将添加一个字串的复本。如果成功, AddString 传回「TRUE」(非 0), 否则传回「FALSE」(0)。如果字

串的长度为 0，或者不能配置储存字串的记忆体，或者已经储存了 256 个字串，则传回值将都是 FALSE。

STRLIB 函式从 STRLIB 的共用记忆体中删除一个字串。

```
EXPORT BOOL CALLBACK DeleteString (pStringIn)
```

另外，参数 pStringIn 是一个字串指标。如果有多个相同内容字串，则删除第一个。如果成功，那么 DeleteString 传回「TRUE」（非 0），否则传回「FALSE」（0）。传回「FALSE」表示字串长度为 0，或者找不到相同内容的字串。

STRLIB 函式使用了呼叫程式中的一个 callback 函式，以便列出目前储存在 STRLIB 共用记忆体中的字串：

```
EXPORT int CALLBACK GetStrings (pfnGetStrCallBack, pParam)
```

在呼叫程式中，callback 函式必须像下面这样定义：

```
EXPORT BOOL CALLBACK GetStrCallBack (PSTR pString, PVOID pParam)
```

GetStrings 的参数 pfnGetStrCallBack 指向 callback 函式。直到 callback 函式传回「FALSE」（0），GetStrings 将为每个字串都呼叫一次 GetStrCallBack。GetStrings 传回传递给 callback 函式的字串数。pParam 参数是一个远程指标，指向程式写作者定义的资料。

当然，此程式可以编译成 Unicode 程式，或者在 STRLIB 的支援下，编译成 Unicode 和非 Unicode 应用程式。与 EDRLIB 一样，所有的函式都有「A」和「W」两种版本。在内部，STRLIB 以 Unicode 储存所有的字串。如果非 Unicode 程式使用了 STRLIB（也就是说，程式将呼叫 AddStringA、DeleteStringA 和 GetStringsA），字串将在 Unicode 和非 Unicode 之间转换。

与 STRPROG 和 STRLIB 专案相关的工作空间名为 STRPROG。此档案按 EDRTST 工作空间的方式组合。程式 21-3 显示了建立 STRLIB.DLL 动态连结程式库所必须的两个档案。

程式 21-3 STRLI

```
STRLIB.H
/*-----
--
    STRLIB.H header file
-----
*/

#ifdef __cplusplus
#define EXPORT extern "C" __declspec (dlllexport)
#else
#define EXPORT __declspec (dlllexport)
#endif

// The maximum number of strings STRLIB will store and their lengths
```

```

#define      MAX_STRINGS 256
#define      MAX_LENGTH  63

        // The callback function type definition uses generic strings

typedef BOOL (CALLBACK * GETSTRCB) (PCTSTR, PVOID) ;

        // Each function has ANSI and Unicode versions

EXPORT      BOOL CALLBACK AddStringA (PCSTR) ;
EXPORT      BOOL CALLBACK AddStringW (PCWSTR) ;

EXPORT      BOOL CALLBACK DeleteStringA (PCSTR) ;
EXPORT      BOOL CALLBACK DeleteStringW (PCWSTR) ;

EXPORT      int CALLBACK GetStringsA (GETSTRCB, PVOID) ;
EXPORT      int CALLBACK GetStringsW (GETSTRCB, PVOID) ;

        // Use the correct version depending on the UNICODE identifier

#ifdef      UNICODE
#define      AddString          AddStringW
#define      DeleteString      DeleteStringW
#define      GetStrings        GetStringsW
#else
#define      AddString          AddStringA
#define      DeleteString      DeleteStringA
#define      GetStrings        GetStringsA
#endif
STRLIB.C
/*-----
-
        STRLIB.C - Library module for STRPROG program
                        (c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include <wchar.h>                                // for wide-character string
functions
#include "strlib.h"

        // shared memory section (requires /SECTION:shared,RWS in link options)
#pragma      data_seg ("shared")
int          iTotal = 0 ;
WCHAR       szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma      data_seg ()

```

```
#pragma          comment(linker, "/SECTION:shared,RWS")

int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
    return TRUE ;
}

EXPORT BOOL CALLBACK AddStringA (PCSTR pStringIn)
{
    BOOL          bReturn ;
    int           iLength ;
    PWSTR         pWideStr ;

    // Convert string to Unicode and call AddStringW
    iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
    pWideStr = malloc (iLength) ;
    MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
    bReturn = AddStringW (pWideStr) ;
    free (pWideStr) ;

    return bReturn ;
}

EXPORT BOOL CALLBACK AddStringW (PCWSTR pStringIn)
{
    PWSTR         pString ;
    int           i, iLength ;

    if (iTotal == MAX_STRINGS - 1)
        return FALSE ;
    if ((iLength = wcslen (pStringIn)) == 0)
        return FALSE ;
    // Allocate memory for storing string, copy it, convert to
uppercase
    pString = malloc (sizeof (WCHAR) * (1 + iLength)) ;
    wcscpy (pString, pStringIn) ;
    _wcsupr (pString) ;

    // Alphabetize the strings
    for (i = iTotal ; i > 0 ; i-)
    {
        if (wcscmp (pString, szStrings[i - 1]) >= 0)
            break ;
        wcscpy (szStrings[i], szStrings[i - 1]) ;
    }
    wcscpy (szStrings[i], pString) ;
    iTotal++ ;
}
```



```
    free (pString) ;
    return TRUE ;
}

EXPORT BOOL CALLBACK DeleteStringA (PCSTR pStringIn)
{
    BOOL        bReturn ;
    int         iLength ;
    PWSTR        pWideStr ;

    // Convert string to Unicode and call DeleteStringW

    iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
    pWideStr = malloc (iLength) ;
    MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
    bReturn = DeleteStringW (pWideStr) ;
    free (pWideStr) ;

    return bReturn ;
}

EXPORT BOOL CALLBACK DeleteStringW (PCWSTR pStringIn)
{
    int i, j ;
    if (0 == wcslen (pStringIn))
        return FALSE ;
    for (i = 0 ; i < iTotal ; i++)
    {
        if (_wcsicmp (szStrings[i], pStringIn) == 0)
            break ;
    }

    // If given string not in list, return without taking action
    if (i == iTotal)
        return FALSE ;
    // Else adjust list downward
    for (j = i ; j < iTotal ; j++)
        wcscpy (szStrings[j], szStrings[j + 1]) ;
    szStrings[iTotal-1][0] = '\\0' ;
    return TRUE ;
}

EXPORT int CALLBACK GetStringsA (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
    BOOL        bReturn ;
    int         i, iLength ;
    PSTR        pAnsiStr ;

    for (i = 0 ; i < iTotal ; i++)
```

```

{
    // Convert string from Unicode
    iLength = WideCharToMultiByte (    CP_ACP, 0, szStrings[i], -1, NULL, 0, NULL,
    NULL) ;

    pAnsiStr = malloc (iLength) ;
    WideCharToMultiByte (    CP_ACP, 0,  szStrings[i],  -1,  pAnsiStr,
    iLength, NULL, NULL) ;

    // Call callback function

    bReturn = pfnGetStrCallBack (pAnsiStr, pParam) ;

    if (bReturn == FALSE)
        return i + 1 ;

    free (pAnsiStr) ;
}
return iTotat ;
}

EXPORT int CALLBACK GetStringsW (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
    BOOL        bReturn ;
    int         i ;

    for (i = 0 ; i < iTotat ; i++)
    {
        bReturn = pfnGetStrCallBack (szStrings[i], pParam) ;
        if (bReturn == FALSE)
            return i + 1 ;
    }
    return iTotat ;
}

```

除了 DllMain 函式以外，STRLIB 中只有六个函式供其他函式输出用。所有这些函式都按 EXPORT 定义。这会使 LINK 在 STRLIB.LIB 引用程式库中列出它们。

STRPROG 程式

STRPROG 程式如程式 21-4 所示，其内容相当浅显易懂。两个功能表选项 (Enter 和 Delete) 启动一个对话方块，让您输入一个字串，然後 STRPROG 呼叫 AddString 或者 DeleteString。当程式需要更新它的显示区域时，呼叫 GetStrings 并使用函式 GetStrCallBack 来列出所列举的字串。

程式 21-4 STRPROG

STRPROG.C

/*-----

```

--
    STRPROG.C - Program using STRLIB dynamic-link library
                                   (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "strlib.h"
#include "resource.h"

typedef struct
{
    HDC    hdc ;
    int    xText ;
    int    yText ;
    int    xStart ;
    int    yStart ;
    int    xIncr ;
    int    yIncr ;
    int    xMax ;
    int    yMax ;
}
CBPARAM ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR      szAppName [] = TEXT ("StrProg") ;
TCHAR szString [MAX_LENGTH + 1] ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
        = szAppName ;
    wndclass.lpszClassName
        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),

```

```

szAppName, MB_ICONERROR) ;

    return 0 ;
}

hwnd = CreateWindow (  szAppName, TEXT ("DLL Demonstration Program"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

BOOL CALLBACK DlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_INITDIALOG:
        SendDlgItemMessage (hDlg, IDC_STRING, EM_LIMITTEXT, MAX_LENGTH, 0) ;
        return TRUE ;

    case WM_COMMAND:
        switch (wParam)
        {
        case IDOK:
            GetDlgItemText (hDlg, IDC_STRING, szString, MAX_LENGTH) ;
            EndDialog (hDlg, TRUE) ;
            return TRUE ;

        case IDCANCEL:
            EndDialog (hDlg, FALSE) ;
            return TRUE ;

        }

    }
    return FALSE ;
}

BOOL CALLBACK GetStrCallBack (PTSTR pString, CBPARAM * pcbp)
{
    TextOut (  pcbp->hdc, pcbp->xText, pcbp->yText,

```

```

        pString, lstrlen (pString)) ;

if ((pcbp->yText += pcbp->yIncr) > pcbp->yMax)
{
    pcbp->yText = pcbp->yStart ;
    if ((pcbp->xText += pcbp->xIncr) > pcbp->xMax)
        return FALSE ;
}
return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HINSTANCE          hInst ;
    static int                cxChar, cyChar, cxClient, cyClient ;
    static UINT               iDataChangeMsg ;
    CBPARAM                   cbparam ;
    HDC                        hdc ;
    PAINTSTRUCT                ps ;
    TEXTMETRIC                tm ;

    switch (message)
    {
    case WM_CREATE:
        hInst          = ((LPCREATESTRUCT) lParam)->hInstance ;
        hdc = GetDC (hwnd) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar      = (int) tm.tmAveCharWidth ;
        cyChar      = (int) (tm.tmHeight + tm.tmExternalLeading) ;
        ReleaseDC (hwnd, hdc) ;

        // Register message for notifying instances of data changes

        iDataChangeMsg = RegisterWindowMessage (TEXT ("StrProgDataChange")) ;
        return 0 ;
    case WM_COMMAND:
        switch (wParam)
        {
        case IDM_ENTER:
            if (DialogBox (hInst, TEXT ("EnterDlg"), hwnd, &DlgProc))
            {
                if (AddString (szString))
                    PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
                else
                    MessageBeep (0) ;
            }
            break ;
        }
    }
}

```

```

        case IDM_DELETE:
            if (DialogBox (hInst, TEXT ("DeleteDlg"), hwnd, &DlgProc))
            {
                if (DeleteString (szString))
                    PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
                else
                    MessageBeep (0) ;
            }
            break ;
        }
        return 0 ;

case WM_SIZE:
    cxClient = (int) LOWORD (lParam) ;
    cyClient = (int) HIWORD (lParam) ;
    return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    cbparam.hdc          = hdc ;
    cbparam.xText        = cbparam.xStart = cxChar ;
    cbparam.yText        = cbparam.yStart = cyChar ;
    cbparam.xIncr        = cxChar * MAX_LENGTH ;
    cbparam.yIncr        = cyChar ;
    cbparam.xMax         = cbparam.xIncr * (1 + cxClient / cbparam.xIncr) ;
    cbparam.yMax         = cyChar * (cyClient / cyChar - 1) ;

    GetStrings ((GETSTRCB) GetStrCallBack, (PVOID) &cbparam) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;

default:
    if (message == iDataChangeMsg)
        InvalidateRect (hwnd, NULL, TRUE) ;
    break ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

[STRPROG.RC \(摘录\)](#)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"

```

```

#include "afxres.h"

////////////////////////////////////
/
// Dialog
ENTERDLG DIALOG DISCARDABLE 20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Enter"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT                                "&Enter:", IDC_STATIC, 7, 7, 26, 9
    EDITTEXT
    IDC_STRING, 31, 7, 148, 12, ES_AUTOHSCROLL
    DEFPUSHBUTTON        "OK", IDOK, 32, 26, 50, 14
    PUSHBUTTON           "Cancel", IDCANCEL, 104, 26, 50, 14
END
DELETEDLG DIALOG DISCARDABLE 20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Delete"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT                                "&Delete:", IDC_STATIC, 7, 7, 26, 9
    EDITTEXT
    IDC_STRING, 31, 7, 148, 12, ES_AUTOHSCROLL
    DEFPUSHBUTTON        "OK", IDOK, 32, 26, 50, 14
    PUSHBUTTON           "Cancel", IDCANCEL, 104, 26, 50, 14
END

////////////////////////////////////
/
// Menu
STRPROG MENU DISCARDABLE
BEGIN
    MENUITEM "&Enter!",                                IDM_ENTER
    MENUITEM "&Delete!",                                IDM_DELETE
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by StrProg.rc

#define IDC_STRING 1000
#define IDM_ENTER 40001
#define IDM_DELETE 40002
#define IDC_STATIC -1

```

STRPROG.C 包含 STRLIB.H 表头档案，其中定义了 STRPROG 将使用的 STRLIB 中的三个函式。

当您执行 STRPROG 的多个执行实体的时候，本程式的奥妙之处就会显露出来。STRLIB 将在共用记忆体中储存字串及其指标，并允许 STRPROG 中的所有执行实体共用此资料。让我们看一下它是如何执行的吧。

在 STRPROG 执行实体之间共用资料

Windows 在一个 Win32 程序的位址空间周围筑了一道墙。通常，一个程序的位址空间中的资料是私有的，对别的程序而言是不可见的。但是执行 STRPROG 的多个执行实体表示了 STRLIB 在程式的所有执行实体之间共用资料是毫无问题的。当您在一个 STRPROG 视窗中增加或者删除一个字串时，这种改变将立即反映在其他的视窗中。

在全部常式之间，STRLIB 共用两个变数：一个字元阵列和一个整数（记录已储存的有效字串的个数）。STRLIB 将这两个变数储存在共用的一个特殊记忆体区段中：

```
#pragma    data_seg ("shared")
int        iTotal = 0 ;
WCHAR      szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma    data_seg ()
```

第一个#pragma 叙述建立资料段，这里命名为 shared。您可以将这段命名为任何一个您喜欢的名字。在这里的#pragma 叙述之後的所有初始化了的变数都放在 shared 资料段中。第二个#pragma 叙述标示段的结束。对变数进行专门的初始化是很重要的，否则编译器将把它们放在普通的未初始化资料段中而不是放在 shared 中。

连结器必须知道有一个「shared」共享资料段。在「Project Settings」对话方块选择「Link」页面标签。选中「STRLIB」时在「Project Options」栏位（在 Release 和 Debug 设定中均可），包含下面的连结叙述：

```
/SECTION:shared,RWS
```

字母 RWS 表示段具有读、写和共用属性。或者，您也可以直接用 DLL 原始码指定连结选项，就像我们在 STRLIB.C 那样：

```
#pragma comment(linker, "/SECTION:shared,RWS")
```

共用的记忆体段允许 iTotal 变数和 szStrings 字串阵列在 STRLIB 的所有常式之间共用。因为 MAX_STRINGS 等於 256，而 MAX_LENGTH 等於 63，所以，共用记忆体段的长度为 32,772 位元组——iTotal 变数需要 4 位元组，256 个指标中的每一个都需要 128 位元组。

使用共用记忆体段可能是在多个应用程式间共用资料的最简单的方法。如果需要动态配置共用记忆体空间，您应该查看记忆体映射档案物件的用法，文件 在 /Platform SDK/Windows Base Services/Interprocess

Communication/File Mapping。

各式各样的 DLL 讨论

如前所述，动态连结程式库模组不接收讯息，但是，动态连结程式库模组可呼叫 GetMessage 和 PeekMessage。实际上，从讯息伫列中得到的讯息是发给呼叫程式库函式的程式的。一般来说，程式库是替呼叫它的程式工作的，这是一项对程式库所呼叫的大多数 Windows 函式都适用的规则。

动态连结程式库可以从程式库档案或者从呼叫程式库的程式档案中载入资源（如图示、字串和点阵图）。载入资源的函式需要执行实体代号。如果程式库使用它自己的执行实体代号（初始化期间传给程式库的），则程式库能从它自己的档案中获得资源。为了从呼叫程式的 .EXE 档案中得到资源，程式库程式库函式需要呼叫该函式的程式的执行实体代号。

在程式库中登录视窗类别和建立视窗需要一点技巧。视窗类别结构和 CreateWindow 呼叫都需要执行实体代号。尽管在建立视窗类别和视窗时可使用动态连结程式库模组的执行实体代号，但在程式库建立视窗时，视窗讯息仍会发送到呼叫程式库中程式的讯息伫列。如果使用者必须在程式库中建立视窗类别和视窗，最好的方法可能是使用呼叫程式的执行实体代号。

因为模态对话方块的讯息是在程式的讯息回圈之外接收到的，因此使用者可以在程式库中呼叫 DialogBox 来建立模态对话方块。执行实体代号可以是程式库代号，并且 DialogBox 的 hwndParent 参数可以为 NULL。

不用输入引用资讯的动态连结

除了在第一次把使用者程式载入记忆体时，由 Windows 执行动态连结外，程式执行时也可以把程式同动态连结程式库模组连结到一起。例如，您通常会这样呼叫 Rectangle 函式：

```
Rectangle (hdc, xLeft, yTop, xRight, yBottom) ;
```

因为程式和 GDI32.LIB 引用程式库连结，该程式库提供了 Rectangle 的地址，因此这种方法有效。

您也可以用更迂回的方法呼叫 Rectangle。首先用 typedef 为 Rectangle 定义一个函式型态：

```
typedef BOOL (WINAPI * PFNRECT) (HDC, int, int, int, int) ;
```

然後定义两个变数：

```
HANDLE      hLibrary ;  
PFNRECT     pfnRectangle ;
```

现在将 hLibrary 设定为程式库代号，将 lpfnRectangle 设定为 Rectangle

函式的位址：

```
hLibrary = LoadLibrary (TEXT ("GDI32.DLL"))
pfnRectangle = (PFNPRECT) GetProcAddress (hLibrary, TEXT ("Rectangle"))
```

如果找不到程式库档案或者发生其他一些错误，LoadLibrary 函式传回 NULL。现在您可以呼叫函式然後释放程式库：

```
pfnRectangle (hdc, xLeft, yTop, xRight, yBottom) ;
FreeLibrary (hLibrary) ;
```

尽管这项执行时期动态连结的技术并没有为 Rectangle 函式增加多大好处，但它肯定是有用的，如果直到执行时还不知道程式动态连结程式库模組的名称，这时就需要使用它。

上面的程式码使用了 LoadLibrary 和 FreeLibrary 函式。Windows 为所有的动态连结程式库模組提供「引用计数」，LoadLibrary 使引用计数递增。当 Windows 载入任何使用了程式库的程式时，引用计数也会递增。FreeLibrary 使引用计数递减，在使用了程式库的程式执行实体结束时也是如此。当引用计数为零时，Windows 将从记忆体中把程式库删除掉，因为不再需要它了。

纯资源程式库

可由 Windows 程式或其他程式库使用的动态连结程式库中的任何函式都必须被输出。然而，DLL 也可以不包含任何输出函式。那么，DLL 到底包含什么呢？答案是资源。

假设使用者正在使用需要几幅点阵图的 Windows 應用程式进行工作。通常要在程式的资源描述档中列出资源，并用 LoadBitmap 函式把它们载入记忆体。但使用者可能希望建立若干套点阵图，每一套均适用於 Windows 所使用的不同显示卡。将不同套的点阵图存放到不同档案中可能是明智的，因为只需要在硬碟上保留一套点阵图。这些档案就是纯资源档案。

程式 21-5 说明如何建立包含 9 幅点阵图的名为 BITLIB.DLL 的纯资源程式库档案。BITLIB.RC 档案列出了所有独立的点阵图档案并为每个档案赋予一个序号。为了建立 BITLIB.DLL，需要 9 幅名为 BITMAP1.BMP、BITMAP2.BMP 等等的点阵图。您可以使用附带的光碟上提供的点阵图或者在 Visual C++中建立这些点阵图。它们与 ID 从 1 到 9 相对应。

程式 21-5 BITLIB

```
BITLIB.C
/*-----
   BITLIB.C -- Code entry point for BITLIB dynamic-link library
                                   (c) Charles Petzold, 1998
   -----*/
```

```

#include <windows.h>
int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
    return TRUE ;
}

BITLIB.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////
/
// Bitmap
1          BITMAP  DISCARDABLE      "bitmap1.bmp"
2          BITMAP  DISCARDABLE      "bitmap2.bmp"
3          BITMAP  DISCARDABLE      "bitmap3.bmp"
4          BITMAP  DISCARDABLE      "bitmap4.bmp"
5          BITMAP  DISCARDABLE      "bitmap5.bmp"
6          BITMAP  DISCARDABLE      "bitmap6.bmp"
7          BITMAP  DISCARDABLE      "bitmap7.bmp"
8          BITMAP  DISCARDABLE      "bitmap8.bmp"
9          BITMAP  DISCARDABLE      "bitmap9.bmp"

```

在名为 SHOWBIT 的工作空间中建立 BITLIB 专案。在名为 SHOWBIT 的另一个专案中，建立程式 21-6 所示的 SHOWBIT 程式，这与前面的一样。不过，不要使 BITLIB 依赖於 SHOWBIT；否则，连结程序中将需要 BITLIB.LIB 档案，并且因为 BITLIB 没有任何输出函式，它也不会建立 BITLIB.LIB。事实上，要分别重新编译 BITLIB 和 SHOWBIT，可以交替设定其中一个为「Active Project」然後再重新编译。

SHOWBIT.C 从 BITLIB 读取点阵图资源，然後在其显示区域显示。按键盘上的任意键可以循环显示。

程式 21-6 SHOWBIT

```

SHOWBIT.C
/*-----
    SHOWBIT.C -- Shows bitmaps in BITLIB dynamic-link library
                                   (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("ShowBit") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int

```

```

iCmdShow)
{
    HWND                hwnd ;
    MSG                 msg ;
    WNDCLASS             wndclass ;

    wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance  = hInstance ;
    wndclass.hIcon       = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName,
        TEXT ("Show Bitmaps from BITLIB (Press Key)"),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    if (!hwnd)
        return 0 ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void DrawBitmap (HDC hdc, int xStart, int yStart, HBITMAP hBitmap)
{
    BITMAP          bm ;
    HDC              hMemDC ;

```

```
POINT                pt ;

hMemDC = CreateCompatibleDC (hdc) ;
SelectObject (hMemDC, hBitmap) ;
GetObject (hBitmap, sizeof (BITMAP), &bm) ;
pt.x = bm.bmWidth ;
pt.y = bm.bmHeight ;

BitBlt (hdc, xStart, yStart, pt.x, pt.y, hMemDC, 0, 0, SRCCOPY) ;
DeleteDC (hMemDC) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HINSTANCE      hLibrary ;
    static int             iCurrent = 1 ;
    HBITMAP                hBitmap ;
    HDC                    hdc ;
    PAINTSTRUCT             ps ;

    switch (message)
    {
    case WM_CREATE:
        if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)
        {
            MessageBox ( hwnd, TEXT ("Can't load BITLIB.DLL."),
                szAppName, 0) ;
            return -1 ;
        }
        return 0 ;

    case WM_CHAR:
        if (hLibrary)
        {
            iCurrent ++ ;
            InvalidateRect (hwnd, NULL, TRUE) ;
        }
        return 0 ;

    case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        if (hLibrary)
        {
            hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;

            if (!hBitmap)
            {

```

```

        iCurrent = 1 ;
        hBitmap = LoadBitmap ( hLibrary,
        MAKEINTRESOURCE (iCurrent)) ;
    }
    if (hBitmap)
    {
        DrawBitmap (hdc, 0, 0, hBitmap) ;
        DeleteObject (hBitmap) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY:
    if (hLibrary)
        FreeLibrary (hLibrary) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

在处理 WM_CREATE 讯息处理期间, SHOWBIT 获得了 BITLIB.DLL 的代号:

```
if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)
```

如果 BITLIB.DLL 与 SHOWBIT.EXE 不在同一个目录, Windows 将按本章前面讨论的方法搜索。如果 LoadLibrary 传回 NULL, SHOWBIT 显示一个讯息方块来报告错误, 并从 WM_CREATE 讯息传回-1。这将导致 WinMain 中的 CreateWindow 呼叫传回 NULL, 而且程式终止程式。

SHOWBIT 透过程式库代号和点阵图号码来呼叫 LoadBitmap, 从而得到一个点阵图代号:

```
hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;
```

如果号码 iCurrent 对应的点阵图无效或者没有足够的记忆体载入点阵图, 则传回一个错误。

在处理 WM_DESTROY 讯息时, SHOWBIT 释放程式库:

```
FreeLibrary (hLibrary) ;
```

当 SHOWBIT 的最後一个执行实体终止时, BITLIB.DLL 的引用计数变为 0, 并且释放所占用的记忆体。这就是实作「图片剪辑」程式的一种简单方法, 所谓的「图片剪辑」程式就是能够将预先建立的点阵图 (或者 metafile、增强型 metafile) 载入到剪贴簿, 以供其他程式使用的程式。

第二十二章 声音与音乐

在 Microsoft Windows 中, 声音、音乐与视讯的综合运用是一个重要的进步。对多媒体的支援起源於 1991 年所谓的 Microsoft Windows 多媒体延伸功能 (Multimedia Extensions to Microsoft Windows)。1992 年, Windows 3.1 的发布使得对多媒体的支援成为另一类 API。最近几年, CD-ROM 驱动器和音效卡——在 90 年代初期还很少见——已成为新 PC 的标准配备。现在, 几乎所有的人们都深信: 多媒体在很大程度上有益於 Windows 的视觉化图形, 从而使电脑摆脱了其只是处理数字和文字的机器的传统角色。

WINDOWS 和多媒体

从某种意义上来说, 多媒体就是透过与装置无关的函式呼叫来获得对各种硬体的存取。让我们首先看一下硬体, 然後再看看 Windows 多媒体 API 的结构。

多媒体硬体

或许最常用的多媒体硬体就是波形声音设备, 也就是平常所说的音效卡。波形声音设备将麦克风的输入或其他声音输入转换为数位取样, 并将其储存到记忆体或者储存到以 .WAV 为副档名的磁碟档案中。波形声音设备还将波形转换回类比声音, 以便通过 PC 扩音器来播放。

音效卡通常还包含 MIDI 设备。MIDI 是符合工业标准的乐器数位化介面 (Musical Instrument Digital Interface)。这类硬体播放音符以回应短的二进位命令讯息。MIDI 硬体通常还可以通过电缆连结到如音乐键盘等的 MIDI 输入设备上。通常, 外部的 MIDI 合成器也能够添加到音效卡上。

现在, 大多数 PC 上的 CD-ROM 驱动器都具备播放普通音乐 CD 的能力。这就是平常所说的「CD 声音」。来自波形声音设备、MIDI 设备以及 CD 声音设备的输出, 一般在使用者的控制下用「音量控制」程式混合在一起。

另外几种普遍的多媒体「设备」不需要额外的硬体。Windows 视讯设备 (也称作 AVI 视讯设备) 播放副档名为 .AVI (audio-video interleave: 声音视频插格) 的电影或动画档案。「ActiveMovie 控制项」可以播放其他型态的电影, 包括 QuickTime 和 MPEG。PC 上的显示卡需要特定的硬体来协助播放这些电影。

还有个别 PC 使用者使用某种 Pioneer 雷射影碟机或者 Sony VISCA 系列录放影机。这些设备都有序列埠介面, 因此可由 PC 软体来控制。某些显示卡具有一种称为「视窗影像 (video in a window)」的功能, 此功能允许一个外部的

视讯信号与其他应用程式一起出现在 Windows 的萤幕上。这也可认为是一种多媒体设备。

API 概述

在 Windows 中,API 支援的多媒体功能主要分成两个集合。它们通常称为「低阶」和「高阶」介面。

低阶介面是一系列函式,这些函式以简短的说明性字首开头,而且在 /Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Functions (与高阶函式一起) 中列出。

低阶的波形声音输入输出函式的字首是 waveIn 和 waveOut。我们将在本章看到这些函式。另外,本章还讨论用 midiOut 函式来控制 MIDI 输出设备。这些 API 还包括 midiIn 和 midiStream 函式。

本章还使用字首为 time 的函式,这些函式允许设定一个高解析度的计时器常式,其计时器的时间间隔速率最低能够到 1 毫秒。此程式主要用於播放 MIDI 音乐。其他几组函式包括声音压缩、视讯压缩以及动画和视讯序列,可惜的是本章不包括这些函式。

您还会注意到多媒体函式列表中七个带有字首 mci 的函式,它们允许存取媒体控制介面 (MCI: Media Control Interface)。这是一个高阶的开放介面,用於控制多媒体 PC 中所有的多媒体硬体。MCI 包括所有多媒体硬体都共有的许多命令,因为多媒体的许多方面都以磁带答录机这类设备播放/记录方式为模型。您为输入或输出而「打开」一台设备,进而可以「录音」(对於输入)或者「播放」(对於输出),并且结束後可以「关闭」设备。

MCI 本身分为两种形式。一种形式下,可以向 MCI 发送讯息,这类似於 Windows 讯息。这些讯息包括位元编码标记和 C 资料结构。另一种形式下,可以向 MCI 发送文字字串。这个程式主要用於描述命令语言,此语言具有灵活的字串处理函式,但支援呼叫 Windows API 的函式不多。字串命令版的 MCI 还有利於交互研究和学习 MCI,我们马上就举一个例子。MCI 中的设备名称包括 CD 声音 (cdaudio)、波形音响 (waveaudio)、MIDI 编曲器 (sequencer)、影碟机 (videodisc)、vcr、overlay (视窗中的类比视频)、dat (digital audio tape: 数位式录频磁带) 以及数位视频 (digitalvideo)。MCI 设备分为「简单型」和「混合型」。简单型设备 (如 CD 声音) 不使用档案。混合型设备 (如波形音响) 则使用档案。使用波形音响时,这些档案的副档名是 .WAV。

存取多媒体硬体的另一种方法包括 DirectX API,它超出了本书的范围。

另外两个高阶多媒体函式也值得一提: MessageBeep 和 PlaySound,它们在

第三章有示范。MessageBeep 播放「控制台」的「声音」中指定的声音。PlaySound 可播放磁碟上、记忆体中或者作为资源载入的.WAV 档案。本章的後面还会用到 PlaySound 函式。

用 TESTMCI 研究 MCI

在 Windows 多媒体的早期, 软体开发套件含有一个名为 MCITEST 的 C 程式, 它允许程式写作者交谈式输入 MCI 命令并学习这些命令的工作方式。这个程式, 至少是 C 语言版, 显然已经消失了。因此, 我又重新建立了它, 即程式 22-1 所示的 TESTMCI 程式。虽然我不认为目前程式码与旧的程式码有什么区别, 但现在的使用者介面还是依据以前的 MCITEST 程式, 并且没有使用现在的程式码。

程式 22-1 TESTMCI

```
TESTMCI.C
/*-----
-
TESTMCI.C -- MCI Command String Tester
(c) Charles Petzold, 1998
-----
*/

#include <windows.h>
#include "resource.h"

#define ID_TIMER 1
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("TestMci") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
    }
    return 0 ;
}

BOOL CALLBACK DlgProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND hwndEdit ;
    int          iCharBeg, iCharEnd, iLineBeg, iLineEnd, iChar, iLine, iLength ;
    MCIERROR      error ;
    RECT          rect ;
```

```
TCHAR                szCommand [1024], szReturn [1024],
                    szError [1024], szBuffer [32] ;

switch (message)
{
case WM_INITDIALOG:

    // Center the window on screen

    GetWindowRect (hwnd, &rect) ;
    SetWindowPos (hwnd, NULL,
        (GetSystemMetrics (SM_CXSCREEN) - rect.right + rect.left) / 2,
        (GetSystemMetrics (SM_CYSCREEN) - rect.bottom + rect.top) / 2,
        0, 0, SWP_NOZORDER | SWP_NOSIZE) ;

    hwndEdit = GetDlgItem (hwnd, IDC_MAIN_EDIT) ;
    SetFocus (hwndEdit) ;
    return FALSE ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDOK:
        // Find the line numbers corresponding to the selection

        SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iCharBeg,
            (LPARAM) &iCharEnd) ;

        iLineBeg = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharBeg, 0) ;
        iLineEnd = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharEnd, 0) ;

        // Loop through all the lines

        for (iLine = iLineBeg ; iLine <= iLineEnd ; iLine++)
        {
            // Get the line and terminate it; ignore if blank

            * (WORD *) szCommand = sizeof (szCommand) / sizeof (TCHAR) ;

            iLength = SendMessage (hwndEdit, EM_GETLINE, iLine,
                (LPARAM) szCommand) ;
            szCommand [iLength] = '\\0' ;

            if (iLength == 0)
                continue ;

            // Send the MCI command

            error = mciSendString (szCommand, szReturn,
```

```
sizeof (szReturn) / sizeof (TCHAR), hwnd) ;

    // Set the Return String field

SetDlgItemText (hwnd, IDC_RETURN_STRING, szReturn) ;

    // Set the Error String field (even if no error)

mciGetErrorString (error, szError, sizeof (szError) / sizeof (TCHAR)) ;

SetDlgItemText (hwnd, IDC_ERROR_STRING, szError) ;
}
// Send the caret to the end of the last selected line

iChar = SendMessage (hwndEdit, EM_LINEINDEX, iLineEnd, 0) ;
iChar += SendMessage (hwndEdit, EM_LINELENGTH, iCharEnd, 0) ;
SendMessage (hwndEdit, EM_SETSEL, iChar, iChar) ;

// Insert a carriage return/line feed combination

SendMessage (hwndEdit, EM_REPLACESEL, FALSE,
    (LPARAM) TEXT ("\r\n")) ;
    SetFocus (hwndEdit) ;
    return TRUE ;

case IDCANCEL:
    EndDialog (hwnd, 0) ;
    return TRUE ;

case IDC_MAIN_EDIT:
    if (HIWORD (wParam) == EN_ERRSPACE)
    {
        MessageBox (hwnd, TEXT ("Error control out of space."),
            szAppName, MB_OK | MB_ICONINFORMATION) ;
        return TRUE ;
    }
    break ;
}
break ;

case MM_MCINOTIFY:
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), TRUE) ;

    wsprintf (szBuffer, TEXT ("Device ID = %i"), lParam) ;
    SetDlgItemText (hwnd, IDC_NOTIFY_ID, szBuffer) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), TRUE) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL),
        wParam & MCI_NOTIFY_SUCCESSFUL) ;
```

```

        EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED),
                        wParam & MCI_NOTIFY_SUPERSEDED) ;

        EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED),
                        wParam & MCI_NOTIFY_ABORTED) ;

        EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE),
                        wParam & MCI_NOTIFY_FAILURE) ;

        SetTimer (hwnd, ID_TIMER, 5000, NULL) ;
        return TRUE ;

case WM_TIMER:
    KillTimer (hwnd, ID_TIMER) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE), FALSE) ;
    return TRUE ;

case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
        case SC_CLOSE:
            EndDialog (hwnd, 0) ;
            return TRUE ;
    }
    break ;
}
return FALSE ;
}

```

TESTMCI.RC (摘录)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
TESTMCI    DIALOG DISCARDABLE  0, 0, 270, 276
STYLE      WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION    "MCI Tester"
FONT 8,    "MS Sans Serif"
BEGIN

```

```

        EDITTEXT                IDC_MAIN_EDIT,8,8,254,100,ES_MULTILINE
ES_AUTOHSCROLL |
                                WS_VSCROLL
        LTEXT                    "Return String:",IDC_STATIC,8,114,60,8
        EDITTEXT                IDC_RETURN_STRING,8,126,120,50,ES_MULTILINE |
                                ES_AUTOVSCROLL | ES_READONLY | WS_GROUP
| NOT WS_TABSTOP
        LTEXT                    "Error String:",IDC_STATIC,142,114,60,8
        EDITTEXT                IDC_ERROR_STRING,142,126,120,50,ES_MULTILINE |
                                ES_AUTOVSCROLL | ES_READONLY | NOT
WS_TABSTOP
        GROUPBOX                "MM_MCINOTIFY Message",IDC_STATIC,9,186,254,58
        LTEXT                    "",IDC_NOTIFY_ID,26,198,100,8
        LTEXT
        "MCI_NOTIFY_SUCCESSFUL",IDC_NOTIFY_SUCCESSFUL,26,212,100,
                                8,WS_DISABLED
        LTEXT
        "MCI_NOTIFY_SUPERSEDED",IDC_NOTIFY_SUPERSEDED,26,226,100,
                                8,WS_DISABLED
        LTEXT
        "MCI_NOTIFY_ABORTED",IDC_NOTIFY_ABORTED,144,212,100,8,
                                WS_DISABLED
        LTEXT
        "MCI_NOTIFY_FAILURE",IDC_NOTIFY_FAILURE,144,226,100,8,
                                WS_DISABLED
        DEFPUSHBUTTON            "OK",IDOK,57,255,50,14
        PUSHBUTTON               "Close",IDCANCEL,162,255,50,14
END

```

RESOURCE.H (摘录)

```

// Microsoft Developer Studio generated include file.
// Used by TestMci.rc

```

```

#define IDC_MAIN_EDIT                1000
#define IDC_NOTIFY_MESSAGE           1005
#define IDC_NOTIFY_ID                1006
#define IDC_NOTIFY_SUCCESSFUL        1007
#define IDC_NOTIFY_SUPERSEDED        1008
#define IDC_NOTIFY_ABORTED           1009
#define IDC_NOTIFY_FAILURE           1010
#define IDC_SIGNAL_MESSAGE           1011
#define IDC_SIGNAL_ID                1012
#define IDC_SIGNAL_PARAM             1013
#define IDC_RETURN_STRING            1014
#define IDC_ERROR_STRING             1015
#define IDC_DEVICES                  1016
#define IDC_STATIC                   -1

```

与本章的大多数程式一样，TESTMCI 使用非模态对话方块作为它的主视窗。

与本章所有的程式一样，TESTMCI 要求 WINMM.LIB 引用程式库在 Microsoft Visual C++ 「Projects Settings」对话方块的「Links」页列出。

此程式用到了两个最重要的多媒体函式：mciSendString 和 mciGetErrorText。在 TESTMCI 的主编辑视窗输入一些内容然後按下 Enter 键(或「OK」按钮)後，程式将输入的字串作为第一个参数传递给 mciSendString 命令：

```
error = mciSendString (szCommand, szReturn,
                      sizeof (szReturn) / sizeof (TCHAR), hwnd) ;
```

如果在编辑视窗选择了不止一行，则程式将按顺序将它们发送给 mciSendString 函式。第二个参数是字串位址，此字串取得从函式传回的资讯。程式将此资讯显示在视窗的「Return String」区域。从 mciSendString 传回的错误代码传递给 mciGetErrorString 函式，以获得文字错误说明；此说明显示在 TESTMCI 视窗的「Error String」区域。

MCITEXT 和 CD 声音

通过控制 CD-ROM 驱动器和播放声音 CD，您会对 MCI 命令字串留下很好的印象。因为这些命令字串一般都非常简单，并且更重要的是您可以听到一些音乐，所以这是好的起点。您可以在 /Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Command Strings 中获得 MCI 命令字串的参考，以方便本练习。

请确认 CD-ROM 驱动器的声音输出已连结到扩音器或耳机，然後放入一张声音 CD，如 Bruce Springsteen 的「Born to Run」。Windows 98 中，「CD 播放程式」将启动并开始播放此唱片。如果是这样的话，终止「CD 播放程式」，然後可以叫出 TESTMCI 并且键入命令：

```
open cdaudio
```

然後按 Enter 键。其中 open 是 MCI 命令，cdaudio 是 MCI 认定的 CD-ROM 驱动器的设备名称（假定您的系统中只有一个 CD-ROM 驱动器。要获得多个 CD-ROM 驱动器名称需使用 sysinfo 命令）。

TESTMCI 中的「Return String」区域显示 mciSendString 函式中系统传回给程式的字串。如果执行了 open 命令，则此值是 1。TESTMCI 在「Error String」区域中显示 mciGetErrorString 依据 mciSendString 传回值所传回的资讯。如果 mciSendString 没有传回错误代码，则「Error String」区域显示文字“The specified command was carried out”。

假定执行了 open 命令，现在就可以输入：

```
play cdaudio
```

CD 将开始播放唱片上的第一首乐曲「Thunder Road」。输入下面的命令可以暂停播放：

```
pause cdaudio
```

或者

```
stop cdaudio
```

对于 CD 声音设备来说，这些叙述的功能相同。您可用下面的叙述重新播放：

```
play cdaudio
```

迄今为止，我们使用的全部字串都由命令和设备名称组成。其中有些命令带有选项。例如，键入：

```
status cdaudio position
```

根据收听时间的长短，「Return String」区域将显示类似下面的一些字元：

01:15:25

这是些什么？很显然不是小时、分钟和秒，因为 CD 没有那么长。要找出时间格式，请键入：

```
status cdaudio time format
```

现在「Return String」区域显示下面的字串：

msf

这代表「分-秒-格」。CD 声音中，每秒有 75 格。时间格式的讯格部分可在 0 到 74 之间的范围内变化。

状态命令有一连串的选项。使用下面的命令，您可以确定 msf 格式的 CD 全部长度：

```
status cdaudio length
```

对于「Born to Run」，「Return String」区域将显示：

39:28:19

这指的是 39 分 28 秒 19 格。

现在试一下

```
status cdaudio number of tracks
```

「Return String」区域将显示：

8

我们从 CD 封面上知道「Born to Run」CD 上第五首乐曲是主题曲。MCI 命令中的乐曲从 1 开始编号。要想知道乐曲「Born to Run」的长度，可以键入下面的命令：

```
status cdaudio length track 5
```

「Return String」区域将显示：

04:30:22

我们还可确定此乐曲从盘上的哪个位置开始：

```
status cdaudio position track 5
```

「Return String」区域将显示：

```
17:36:35
```

根据这条资讯，我们可以直接跳到乐曲标题：

```
play cdaudio from 17:36:35 to 22:06:57
```

此命令只播放一首乐曲，然後停止。最後的值是由 4:30:22（乐曲长度）加 17:36:35 得到的。或者，也可以用下面的命令确定：

```
status cdaudio position track 6
```

或者，也可以将时间格式设定为乐曲-分-秒-格：

```
set cdaudio time format tmsf
```

然後

```
play cdaudio from 5:0:0:0 to 6:0:0:0
```

或者，更简单地

```
play cdaudio from 5 to 6
```

如果时间的尾部是 0，那么您可去掉它们。还可以用毫秒设定时间格式。

每个 MCI 命令字串都可以在字串的後面包括选项 wait 和 notify（但不是同时使用）。例如，假设您只想播放「Born to Run」的前 10 秒，而且播放後，您还想让程式完成其他工作。您可按下面的方法进行（假定您已经将时间格式设定为 tmsf）：

```
play cdaudio from 5:0:0 to 5:0:10 wait
```

这种情况下，直到函式执行结束，也就是说，直到播放完「Born to Run」的前 10 秒，mciSendString 函式才传回。

现在很明显，一般来说，在单执行绪的应用程式中这不是一件好事。如果不小心键入：

```
play cdaudio wait
```

直到整个唱片播放完以後，mciSendString 函式才将控制权传回给程式。如果必须使用 wait 选项（在只要执行 MCI 描述档案而不管其他事情的时候，这么做很方便，与我将展示的一样），首先使用 break 命令。此命令可设定一个虚拟键码，此码将中断 mciSendString 命令并将控制权传回给程式。例如，要设定 Escape 键来实作此目的，可用：

```
break cdaudio on 27
```

这里，27 是十进位的 VK_ESCAPE 值。

比 wait 选项更好的是 notify 选项：

```
play cdaudio from 5:0:0 to 5:0:10 notify
```


这种情况下, `mciSendString` 函式立即传回, 但如果该操作在 MCI 命令的尾部定义, 则 `mciSendString` 函式的最後一个参数所指定代号的视窗会收到 `MM_MCINOTIFY` 讯息。TESTMCI 程式在 `MM_MCINOTIFY` 框中显示此讯息的结果。为避免与其他可能键入的命令混淆, TESTMCI 程式在 5 秒後停止显示 `MM_MCINOTIFY` 讯息的结果。

您可以同时使用 `wait` 和 `notify` 关键字, 但没有理由这么做。不使用这两个关键字, 内定的操作就既不是 `wait`, 也不是您通常所希望的 `notify`。

用这些命令结束播放时, 可键入下面的命令来停止 CD:

```
stop cdaudio
```

如果在关闭之前没有停止 CD-ROM 设备, 那么甚至在关闭设备之後还会继续播放 CD。

另外, 您还可以试试您的硬体允许或者不允许的一些命令:

```
eject cdaudio
```

最後按下面的方法关闭设备:

```
close cdaudio
```

虽然 TESTMCI 自己不能储存或载入文字档案, 但可以在编辑控制项和剪贴簿之间复制文字: 先从 TESTMCI 选择一些内容, 将其复制到剪贴簿(用 `Ctrl-C`), 再将这些文字从剪贴簿复制到「记事本」, 然後储存。相反的操作, 可以将一系列的 MCI 命令载入到 TESTMCI。如果选择了一系列命令然後按下「OK」按钮(或者 `Enter` 键), 则 TESTMCI 将每次执行一条命令。这就允许您编写 MCI 的「描述档案」, 即 MCI 命令的简单列表。

例如, 假设您想听歌曲「Jungleland」(唱片中的最後一首)、「Thunder Road」和「Born to Run」, 并按此顺序听, 可以编写如下的描述命令:

```
open cdaudio
set cdaudio time format tmsf
break cdaudio on 27
play cdaudio from 8 wait
play cdaudio from 1 to 2 wait
play cdaudio from 5 to 6 wait
stop cdaudio
eject cdaudio
close cdaudio
```

不用 `wait` 关键字, 就不能正常工作, 因为 `mciSendString` 命令会立即传回, 然後执行下一条命令。

此时, 如何编写模拟 CD 播放程式的简单应用程式, 就应该相当清楚了。程式可以确定乐曲数量、每个乐曲的长度并能显示允许使用者从任意位置开始播放(不过, 请记住: `mciSendString` 总是传回文字字串资讯, 因此您需要编写解

析处理程式来将这些字符串转换成数字)。可以肯定,这样的程式还要使用 Windows 计时器,以产生大约 1 秒的时间间隔。在 WM_TIMER 讯息处理期间,程式将呼叫:

```
status cdaudio mode
```

来查看 CD 是暂停还是在播放。

```
status cdaudio position
```

命令允许程式更新显示以给使用者显示目前的位置。但可能还存在更令人感兴趣的事:如果程式知道音乐音调部分的节拍位置,那么就可以使萤幕上的图形与 CD 同步。这对于音乐指令或者建立自己的图形音乐视讯程式极为有用。

波形声音

波形声音是最常用的 Windows 多媒体特性。波形声音设备可以通过麦克风捕捉声音,并将其转换为数值,然后把它们储存到记忆体或者磁碟上的波形档案中,波形档案的副档名是.WAV。这样,声音就可以播放了。

声音与波形

在接触波形声音 API 之前,具备一些预备知识很重要,这些知识包括物理学、听觉以及声音进出电脑的程序。

声音就是振动。当声音改变了鼓膜上空气的压力时,我们就感觉到了声音。麦克风可以感应这些振动,并且将它们转换为电流。同样,电流再经过放大器和扩音器,就又变成了声音。传统上,声音以类比方式储存(例如录音磁带和唱片),这些振动储存在磁气脉冲或者轮廓凹槽中。当声音转换为电流时,就可以用随时间振动的波形来表示。振动最自然的形式可以用正弦波表示,它的一个周期如图 5-5 所示。

正弦波有两个参数——振幅(也就是一个周期中的最大振幅)和频率。我们已知振幅就是音量,频率就是音调。一般来说人耳可感受的正弦波的范围是从 20Hz(每秒周期)的低频声音到 20,000Hz 的高频声,但随著年龄的增长,对高频声音的感受能力会逐年退化。

人感受频率的能力与频率是对数关系而不是线性关系。也就是说,我们感受 20Hz 到 40Hz 的频率变化与感受 40Hz 到 80Hz 的频率变化是一样的。在音乐中,这种加倍的频率定义为八度音阶。因此,人耳可感觉到大约 10 个八度音阶的声音。钢琴的范围是从 27.5 Hz 到 4186 Hz 之间,略小于 7 个八度音阶。

虽然正弦波代表了振动的大多数自然形式,但纯正弦波很少在现实生活中单独出现,而且,纯正弦波并不动听。大多数声音都很复杂。

任何周期的波形(即,一个回圈波形)可以分解成多个正弦波,这些正弦

波的频率都是整倍数。这就是所谓的 Fourier 级数，它以法国数学家和物理学家 Jean Baptiste Joseph Fourier (1768-1830) 的名字命名。周期的频率是基础。级数中其他正弦波的频率是基础频率的 2 倍、3 倍、4 倍（等等）。这些频率的声音称为泛音。基础频率也称作一级谐波。第一泛音是二级谐波，以此类推。

正弦波谐波的相对强度给每个周期的波形唯一的聲音。这就是「音质」，它使得喇叭吹出喇叭声，钢琴弹出钢琴声。

人们一度认为电子合成乐器仅仅需要将声音分解成谐波并且与多个正弦波重组即可。不过，事实证明现实世界中的声音并不是这么简单。代表现实世界中声音的波形都没有严格的周期。乐器之间谐波的相对强度是不同的，并且谐波也随著每个音符的演奏时间改变。特别是乐器演奏音符的开始位置——我们称作起奏 (attack) ——相当复杂，但这个位置又对我们感受音质至关重要。

由於近年来数位储存能力的提高，我们可以将声音直接以数位形式储存而不用复杂的重组。

脉冲编码调制 (Pulse Code Modulation)

电脑处理的是数值，因此要使声音进入电脑，就必须设计一种能将声音与数位信号相互转换的机制。

不压缩资料就完成此功能的最常用方法称作「脉冲编码调制」(PCM: pulse code modulation)。PCM 可用在光碟、数位式录音磁带以及 Windows 中。脉冲编码调制其实只是一种概念上很简单的处理步骤的奇怪代名词而已。

利用脉冲编码调制，波形可以按固定的周期频率取样，其频率通常是每秒几万次。對於每个样本都测量其波形的振幅。完成将振幅转换成数位信号工作的硬体是类比数位转换器 (ADC: analog-to-digital converter)。类似地，通过数位类比转换器 (DAC: digital-to-analog converter) 可将数位信号转换回波形电子信号。但这样转换得到的波形与输入的并不完全相同。合成的波形具有由高频组成的尖锐边缘。因此，播放硬体通常在数位类比转换器後还包括一个低通滤波器。此滤波器滤掉高频，并使合成後的波形更平滑。在输入端，低通滤波器位於 ADC 前面。

脉冲编码调制有两个参数：取样频率，即每秒内测量波形振幅的次数；样本大小，即用於储存振幅级的位元数。与您想像的一样：取样频率越高，样本大小越大，原始声音的复制品才更好。不过，存在一个提高取样频率和样本大小的极点，超过这个极点也就超过了人类分辨声音的极限。另外，如果取样频率和样本大小过低，将导致不能精确地复制音乐以及其他声音。

取样频率

取样频率决定声音可被数位化和储存的最大频率。尤其是，取样频率必须是样本声音最高频率的两倍。这就是「Nyquist 频率 (Nyquist Frequency)」，以 30 年代研究取样程序的工程师 Harry Nyquist 的名字命名。

以过低的取样频率对正弦波取样时，合成的波形比最初的波形频率更低。这就是所说的失真信号。为避免失真信号的发生，在输入端使用低通滤波器以阻止频率大於半个取样频率的所有波形。在输出端，数位类比转换器产生的粗糙的波形边缘实际上是由频率大於半个取样频率的波形组成的泛音。因此，位於输出端的低通滤波器也阻止频率大於半个取样频率的所有波形。

声音 CD 中使用的取样频率是每秒 44,100 个样本，或者称为 44.1kHz。这个特有的数值是这样产生的：

人耳可听到最高 20kHz 的声音，因此要拦截人能听到的整个声音范围，就需要 40kHz 的取样频率。然而，由於低通滤波器具有频率下滑效应，所以取样频率应该再高出大约百分之十才行。现在，取样频率就达到了 44kHz。这时，我们要与视讯同时记录数位声音，於是取样频率就应该是美国、欧洲电视显示格速率的整数倍，这两种视讯格速率分别是 30Hz 和 25Hz。这就使取样频率升高到了 44.1kHz。

取样频率为 44.1kHz 的光碟会产生大量的资料，这對於一些应用程式来说实在是太多了，例如對於录制声音而不是录制音乐时就是这样。把取样频率减半到 22.05 kHz，可由一个 10 kHz 的泛音来简化复制声音的上半部分。再将其减半到 11.025 kHz 就向我们提供了 5 kHz 频率范围。44.1 kHz、22.05 kHz 和 11.025 kHz 的取样频率，以及 8 kHz 都是波形声音设备普遍支援的标准。

因为钢琴的最高频率为 4186 Hz，所以您可能会认为给钢琴录音时，11.025 kHz 的取样频率就足够了。但 4186 Hz 只是钢琴最高的基础频率而已，滤掉大於 5000Hz 的所有正弦波将减少可被复制的泛音，而这样将不能精确地捕捉和复制钢琴的声音。

样本大小

脉冲编码调制的第二个参数是按位元计算的样本大小。样本大小决定了可供录制和播放的最低音与最高音之间的区别。这就是通常所说的动态范围。

声音强度是波形振幅的平方（即每个正弦波一个周期中最大振幅的合成）。与频率一样，人对声音强度的感受也呈对数变化。

两个声音在强度上的区别是以贝尔（以电话发明人 Alexander Graham Bell

的名字命名) 和分贝 (dB) 为单位进行测量的。1 贝尔在声音强度上呈 10 倍增加。1dB 就是以相同的乘法步骤成为 1 贝尔的十分之一。由此, 1dB 可增加声音强度的 1.26 倍 (10 的 10 次方根), 或者增加波形振幅的 1.12 倍 (10 的 20 次方根)。1 分贝是耳朵可感觉出的声强的最小变化。从开始能听到的声音极限到让人感到疼痛的声音极限之间的声强差大约是 100 dB。

可用下面的公式来计算两个声音间的动态范围, 单位是分贝:

$$dB = 20 \cdot \log \left(\frac{A_1}{A_2} \right)$$

其中 A1 和 A2 是两个声音的振幅。因为只可能有一个振幅, 所以样本大小是 1 位元, 动态范围是 0。

如果样本大小是 8 位元, 则最大振幅与最小振幅之间的比例就是 256。这样, 动态范围就是:

$$dB = 20 \cdot \log (256)$$

或者 48 分贝。48 的动态范围大约相当於非常安静的房屋与电动割草机之间的差别。将样本大小加倍到 16 位元产生的动态范围是:

$$dB = 20 \cdot \log (65536)$$

或者 96 分贝。这非常接近听觉极限和疼痛极限, 而且人们认为这就是复制音乐的理想值。

Windows 同时支援 8 位元和 16 位元的样本大小。储存 8 位元的样本时, 样本以无正负号位元组处理, 静音将储存为一个值为 0x80 的字串。16 位元的样本以带正负号整数处理, 这时静音将储存为一个值为 0 的字串。

要计算未压缩声音所需的储存空间, 可用以秒为单位的的声音持续时间乘以取样频率。如果用 16 位元样本而不是 8 位元样本, 则将其加倍, 如果是录制立体声则再加倍。例如, 1 小时的 CD 声音 (或者是在每个立体声样本占 2 位元组、每秒 44,100 个样本的速度下进行 3 600 秒) 需要 635MB, 这快要接近一张 CD-ROM 的储存量了。

在软体中产生正弦波

对於第一个关於波形声音的练习, 我们不打算将声音储存到档案中或播放录制的声音。我们将使用低阶的波形声音 API (即, 字首是 waveOut 的函式) 来建立一个称作 SINEWAVE 的声音正弦波生成器。此程式以 1 Hz 的增量来生成从

20Hz（人可感觉的最低值）到 5,000Hz（与人感觉的最高值相差两个八度音阶）的正弦波。

我们知道，标准 C 执行时期程式库包括了一个 `sin` 函式，该函式传回一个弧度角的正弦值（2 鵠《鵠褥 360 度）。`sin` 函式传回值的范围是从 -1 到 1（早在第五章，我们就在 SINEWAVE 程式中使用过这个函式）。因此，应该很容易使用 `sin` 函式生成输出到波形声音硬体的正弦波资料。基本上是用代表波形（这时是正弦波）的资料来填充缓冲区，并将此缓冲区传递给 API。（这比前面所讲的稍微有些复杂，但我将详细介绍）。波形声音硬体播放完缓冲区中的资料後，应将第二个缓冲区中的资料传递给它，并且以此类推。

第一次考虑这个问题（而且对 PCM 也一无所知）时，您大概会认为将一个周期的正弦波分成若干固定数量的样本——例如 360 个——才合理。对于 20 Hz 的正弦波，每秒输出 7,200 个样本。对于 200 Hz 的正弦波，每秒则要输出 72,000 个样本。这有可能实作，但实际上却不能这么做。对于 5,000 Hz 的正弦波，就需要每秒输出 1,800,000 个样本，这的确会增大 DAC 的负担！更重要的是，对于更高的频率，这种作法会比实际需要的精确度还高。

就脉冲编码调制而言，取样频率是个常数。假定取样频率是 SINEWAVE 程式中使用的 11,025Hz。如果要生成一个 2,756.25Hz（确切地说是四分之一的取样频率）的正弦波，则正弦波的每个周期就有 4 个样本。对于 25Hz 的正弦波，每个周期就有 441 个样本。通常，每周期的样本数等於取样频率除以要得到的正弦波频率。一旦知道了每周期的样本数，用 2π 弧度除此数，然後用 `sin` 函式来获得每周期的样本。然後再反复对一个周期进行取样，从而建立一个连续的波形。

问题是每周期的样本数可能带有小数，因此在使用时这种方法并不是很好。每个周期的尾部都会有间断。

使它正常工作的关键是保留一个静态的「相位角」变数。此角初始化为 0。第一个样本是 0 度正弦。然後，相位角增加一个值，该值等於 2π 乘以频率再除以取样频率。用此相位角作为第二个样本，并且按此方法继续。一旦相位角超过 2π 弧度，则减去 2π 弧度，而不要把相位角再初始化为 0。

例如，假定要用 11,025Hz 的取样频率来生成 1,000Hz 的正弦波。即每周期有大约 11 个样本。为便於理解，此处相位角按度数给出——大约前一个半周期的相位角是：0、32.65、65.31、97.96、130.61、163.27、195.92、228.57、261.22、293.88、326.53、359.18、31.84、64.49、97.14、129.80、162.45、195.10，以此类推。存入缓冲区的波形资料是这些角度的正弦值，并已缩放到每样本的位元数。为後来的缓冲区建立资料时，可继续增加最後的相位角，而

不要将它初始化为 0。

如程式 22-2 所示, FillBuffer 函式完成这项工作——与 SINEWAVE 程式的其余部分一起完成。

程式 22-2 SINEWAVE

```
SINEWAVE.C
/*-----
      SINEWAVE.C --      Multimedia Windows Sine Wave Generator
                              (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <math.h>
#include "resource.h"

#define          SAMPLE_RATE          11025
#define          FREQ_MIN              20
#define          FREQ_MAX             5000
#define          FREQ_INIT             440
#define          OUT_BUFFER_SIZE      4096
#define          PI                    3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("SineWave") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
    }
    return 0 ;
}

VOID FillBuffer (PBYTE pBuffer, int iFreq)
{
    static double      fAngle ;
    int                i ;

    for (i = 0 ; i < OUT_BUFFER_SIZE ; i++)
    {
        pBuffer [i] = (BYTE) (127 + 127 * sin (fAngle)) ;
        fAngle += 2 * PI * iFreq / SAMPLE_RATE ;
        if ( fAngle > 2 * PI)
            fAngle -= 2 * PI ;
    }
}
```

```

}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      BOOL                bShutOff, bClosing ;
    static      HWAVEOUT            hWaveOut ;
    static      HWND                hwndScroll ;
    static      int                 iFreq = FREQ_INIT ;
    static      PBYTE               pBuffer1, pBuffer2 ;
    static      PWAVEHDR            pWaveHdr1, pWaveHdr2 ;
    static      WAVEFORMATEX        waveformat ;
    int          iDummy ;

    switch (message)
    {
    case WM_INITDIALOG:
        hwndScroll =      GetDlgItem (hwnd, IDC_SCROLL) ;
        SetScrollRange    (hwndScroll, SB_CTL, FREQ_MIN, FREQ_MAX, FALSE) ;
        SetScrollPos      (hwndScroll, SB_CTL, FREQ_INIT, TRUE) ;
        SetDlgItemInt     (hwnd, IDC_TEXT, FREQ_INIT, FALSE) ;

        return TRUE ;

    case WM_HSCROLL:
        switch (LOWORD (wParam))
        {
            case SB_LINELEFT:      iFreq -= 1 ; break ;
            case SB_LINERIGHT:     iFreq += 1 ; break ;
            case SB_PAGELEFT:      iFreq /= 2 ; break ;
            case SB_PAGERIGHT:     iFreq *= 2 ; break ;

            case SB_THUMBTRACK:
                iFreq = HIWORD (wParam) ;
                break ;

            case SB_TOP:
                GetScrollRange (hwndScroll, SB_CTL, &iFreq, &iDummy) ;
                break ;

            case SB_BOTTOM:
                GetScrollRange (hwndScroll, SB_CTL, &iDummy, &iFreq) ;
                break ;
        }

        iFreq = max (FREQ_MIN, min (FREQ_MAX, iFreq)) ;
        SetScrollPos (hwndScroll, SB_CTL, iFreq, TRUE) ;
        SetDlgItemInt (hwnd, IDC_TEXT, iFreq, FALSE) ;
    }
}

```



```

        return TRUE ;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_ONOFF:
        // If turning on waveform, hWaveOut is NULL

        if (hWaveOut == NULL)
        {
            // Allocate memory for 2 headers and 2 buffers

            pWaveHdr1  = malloc (sizeof (WAVEHDR)) ;
            pWaveHdr2  = malloc (sizeof (WAVEHDR)) ;
            pBuffer1    = malloc (OUT_BUFFER_SIZE) ;
            pBuffer2    = malloc (OUT_BUFFER_SIZE) ;

            if (!pWaveHdr1 || !pWaveHdr2 || !pBuffer1 || !pBuffer2)
            {
                if (!pWaveHdr1) free (pWaveHdr1) ;
                if (!pWaveHdr2) free (pWaveHdr2) ;
                if (!pBuffer1) free (pBuffer1) ;
                if (!pBuffer2) free (pBuffer2) ;

                MessageBeep (MB_ICONEXCLAMATION) ;
                MessageBox (hwnd, TEXT ("Error allocating memory!"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                return TRUE ;
            }

            // Variable to indicate Off button pressed

            bShutOff = FALSE ;

            // Open waveform audio for output

            waveformat.wFormatTag          = WAVE_FORMAT_PCM ;
            waveformat.nChannels            = 1 ;
            waveformat.nSamplesPerSec       = SAMPLE_RATE ;
            waveformat.nAvgBytesPerSec      = SAMPLE_RATE ;
            waveformat.nBlockAlign          = 1 ;
            waveformat.wBitsPerSample       = 8 ;
            waveformat.cbSize                = 0 ;

            if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveformat,
                DWORD) hwnd, 0, CALLBACK_WINDOW) != MMSYSERR_NOERROR)
            {
                free (pWaveHdr1) ;

```

```

    free (pWaveHdr2) ;
    free (pBuffer1) ;
    free (pBuffer2) ;

    hWaveOut = NULL ;
    MessageBeep (MB_ICONEXCLAMATION) ;
    MessageBox (hwnd, TEXT ("Error opening waveform audio device!"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return TRUE ;
}

// Set up headers and prepare them

pWaveHdr1->lpData                = pBuffer1 ;
pWaveHdr1->dwBufferLength         = OUT_BUFFER_SIZE ;
pWaveHdr1->dwBytesRecorded        = 0 ;
pWaveHdr1->dwUser                 = 0 ;
pWaveHdr1->dwFlags               = 0 ;
pWaveHdr1->dwLoops               = 1 ;
pWaveHdr1->lpNext               = NULL ;
pWaveHdr1->reserved              = 0 ;

waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

pWaveHdr2->lpData                = pBuffer2 ;
pWaveHdr2->dwBufferLength         = OUT_BUFFER_SIZE ;
pWaveHdr2->dwBytesRecorded        = 0 ;
pWaveHdr2->dwUser                 = 0 ;
pWaveHdr2->dwFlags               = 0 ;
pWaveHdr2->dwLoops               = 1 ;
pWaveHdr2->lpNext               = NULL ;
pWaveHdr2->reserved              = 0 ;

waveOutPrepareHeader (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
}
// If turning off waveform, reset waveform audio
else
{
    bShutOff = TRUE ;
    waveOutReset (hWaveOut) ;
}
return TRUE ;
}
break ;

// Message generated from waveOutOpen call

case MM_WOM_OPEN:

```

```

        SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn Off")) ;

                                // Send two buffers to waveform
output device

        FillBuffer (pBuffer1, iFreq) ;
        waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

        FillBuffer (pBuffer2, iFreq) ;
        waveOutWrite (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
        return TRUE ;

                                // Message generated when a buffer
is finished

        case MM_WOM_DONE:
            if (bShutOff)
            {
                waveOutClose (hWaveOut) ;
                return TRUE ;
            }

                                // Fill and send out a new buffer

            FillBuffer (((PWAVEHDR) lParam)->lpData, iFreq) ;
            waveOutWrite (hWaveOut, (PWAVEHDR) lParam, sizeof
(WAVEHDR)) ;

            return TRUE ;

        case MM_WOM_CLOSE:
            waveOutUnprepareHeader (hWaveOut, pWaveHdr1, sizeof
(WAVEHDR)) ;

            waveOutUnprepareHeader (hWaveOut, pWaveHdr2, sizeof
(WAVEHDR)) ;

            free (pWaveHdr1) ;
            free (pWaveHdr2) ;
            free (pBuffer1) ;
            free (pBuffer2) ;

            hWaveOut = NULL ;
            SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn On")) ;

            if (bClosing)
                EndDialog (hwnd, 0) ;

            return TRUE ;

```

```

        case WM_SYSCOMMAND:
            switch (wParam)
            {
                case SC_CLOSE:
                    if (hWaveOut != NULL)
                    {
                        bShutOff = TRUE ;
                        bClosing = TRUE ;

                        waveOutReset (hWaveOut) ;
                    }
                    else
                        EndDialog (hwnd, 0) ;

                    return TRUE ;
            }
            break ;
    }
    return FALSE ;
}

SINEWAVE.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
SINEWAVE          DIALOG DISCARDABLE 100, 100, 200, 50
STYLE              WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION            "Sine Wave Generator"
FONT 8,            "MS Sans Serif"
BEGIN
    SCROLLBAR                IDC_SCROLL,8,8,150,12
    RTEXT                    "440",IDC_TEXT,160,10,20,8
    LTEXT                    "Hz",IDC_STATIC,182,10,12,8
    PUSHBUTTON               "Turn On",IDC_ONOFF,80,28,40,14
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by SineWave.rc

#define IDC_STATIC                                -1
#define IDC_SCROLL                                1000
#define IDC_TEXT                                  1001
#define IDC_ONOFF                                 1002

```

注意, FillBuffer 常式中用到的 OUT_BUFFER_SIZE、SAMPLE_RATE 和 PI 识

别字在程式的顶部定义。FillBuffer 的 iFreq 参数是需要的频率，单位是 Hz。还要注意，sin 函数的结果调整到了 0 到 254 的范围之间。对于每个样本，sin 函数的 fAngle 参数都增加一个值，该值的大小是 2π 弧度乘以需要的频率再除以取样频率。

SINEWAVE 的视窗包含三个控制项：一个用于选择频率的水平卷动列，一个用于显示目前所选频率的静态文字区域，以及一个标记为「Turn On」的按钮。按下此按钮后，您将从连结音效卡的扩音器中听到正弦波的声音，同时按钮上的文字将变成「Turn Off」。用键盘或者滑鼠移动卷动列可以改变频率。要关闭声音，可以再次按下按钮。

SINEWAVE 程式码初始化卷动列，以便频率在 WM_INITDIALOG 讯息处理期间最低是 20Hz，最高是 5000Hz。初始化时，卷动列设定为 440 Hz。用音乐术语来说就是中音上面的 A，它在管弦乐队演奏时用来调音。DlgProc 在接收 WM_HSCROLL 讯息处理期间改变静态变数 iFreq。注意，Page Left 和 Page Right 将导致 DlgProc 增加或者减少一个八度音阶。

当 DlgProc 从按钮收到一个 WM_COMMAND 讯息时，它首先配置 4 个记忆体块——2 个用于 WAVEHDR 结构，我们马上讨论。另两个用于缓冲区储存波形资料，我们将这两个缓冲区称为 pBuffer1 和 pBuffer2。

通过呼叫 waveOutOpen 函数，SINEWAVE 打开波形声音设备以便输出，waveOutOpen 函数使用下面的参数：

```
waveOutOpen (&hWaveOut, wDeviceID, &waveformat, dwCallback,
             dwCallbackData, dwFlags) ;
```

将第一个参数设定为指向 HWAVEOUT (handle to waveform audio output: 波形声音输出代号) 型态的变数。从函数传回时，此变数将设定为一个代号，后面的波形输出呼叫中将使用该代号。

waveOutOpen 的第二个参数是设备 ID。它允许函数可以在安装多个音效卡的机器上使用。参数的范围在 0 到系统所安装的波形输出设备数之间。呼叫 waveOutGetNumDevs 可以获得波形输出设备数，而呼叫 waveOutGetDevCaps 可以找出每个波形输出设备。如果想消除设备问号，那么您可以用常数 WAVE_MAPPER (定义为 -1) 来选择设备，该设备在「控制台」的「多媒体」中「音效」页面标签里的「喜欢使用的装置」中指定。另外，如果首选设备不能满足您的需要，而其他设备可以，那么系统将选择其他设备。

第三个参数是指向 WAVEFORMATEX 结构的指标 (后面将详细介绍)。第四个参数是视窗代号或指向动态连结程式库中 callback 函数的指标，用来表示接收波形输出讯息的视窗或者 callback 函数。使用 callback 函数时，可在第五个参数中指定程式定义的资料。dwFlags 参数可设为 CALLBACK_WINDOW 或

CALLBACK_FUNCTION, 以表示第四个参数的型态。您也可用 WAVE_FORMAT_QUERY 标记来检查能否打开设备（实际上并不打开它）。还有其他几个标记可用。

waveOutOpen 的第三个参数定义为指向 WAVEFORMATEX 型态结构的指标，此结构在 MMSYSTEM.H 中定义如下：

```
typedef struct waveformat_tag
{
    WORD    wFormatTag ;           // waveform format = WAVE_FORMAT_PCM
    WORD    nChannels ;           // number of channels = 1 or 2
    DWORD    nSamplesPerSec ;      // sample rate
    DWORD    nAvgBytesPerSec ;     // bytes per second
    WORD    nBlockAlign ;         // block alignment
    WORD    wBitsPerSample ;      // bits per samples = 8 or 16
    WORD    cbSize ;              // 0 for PCM
}
WAVEFORMATEX, * PWAVEFORMATEX ;
```

您可用此结构指定取样频率（nSamplesPerSec）和取样精确度（nBitsPerSample），以及选择单声道或立体声（nChannels）。结构中有些资讯看起来是多余的，但该结构也可用於非 PCM 的取样方式。在非 PCM 取样方式下，此结构的最後一个栏位设定为非 0 值，并带有其他资讯。

對於 PCM 取样方式，nBlockAlign 栏位设定为 nChannels 乘以 wBitsPerSample 再除以 8 所得到的数值，它表示每次取样的总位元组数。nAvgBytesPerSec 栏位设定为 nSamplesPerSec 和 nBlockAlign 的乘积。

SINEWAVE 初始化 WAVEFORMATEX 结构的栏位，并呼叫 waveOutOpen 函式：

```
waveOutOpen (    &hWaveOut, WAVE_MAPPER, &waveformat,
                (DWORD) hwnd, 0, CALLBACK_WINDOW)
```

如果呼叫成功，则 waveOutOpen 函式传回 MMSYSERR_NOERROR（定义为 0），否则传回非 0 的错误代码。如果 waveOutOpen 的传回值非 0，则 SINEWAVE 清除视窗，并显示一个标识错误的讯息方块。

现在设备打开了，SINEWAVE 继续初始化两个 WAVEHDR 结构的栏位，这两个结构用於在 API 中传递缓冲。WAVEHDR 定义如下：

```
typedef struct wavehdr_tag
{
    LPSTR lpData;                  // pointer to data
buffer
    DWORD dwBufferLength;          // length of
data buffer
    DWORD dwBytesRecorded;         // used for recorded
    DWORD dwUser;                 // for program use
    DWORD dwFlags;                 // flags
    DWORD dwLoops;                 // number of
repetitions
```

```

    struct wavehdr_tag FAR *lpNext;                // reserved
    DWORD reserved;                                // reserved
}
WAVEHDR, *PWAVEHDR ;

```

SINEWAVE 将 lpData 栏位设定为包含资料的缓冲区位址, dwBufferLength 栏位设定为此缓冲区的大小, dwLoops 栏位设定为 1, 其他栏位都设定为 0 或 NULL。如果要重复回圈播放声音, 可设定 dwFlags 和 dwLoops 栏位。

SINEWAVE 下一步为两个资讯表头呼叫 waveOutPrepareHeader 函式, 以防止结构和缓冲区与磁碟发生资料交换。

到此为止, 所有的这些准备都是回应单击开启声音的按钮。但在程式的讯息伫列里已经有一个讯息在等待回应。因为我们已经在函式 waveOutOpen 中指定要用一个视窗讯息处理程式来接收波形输出讯息, 所以 waveOutOpen 函式向程式的讯息伫列发送了 MM_WOM_OPEN 讯息, wParam 讯息参数设定为波形输出代号。要处理 MM_WOM_OPEN 讯息, SINEWAVE 呼叫 FillBuffer 函式两次, 并用正弦波形资料填充 pBuffer 缓冲区。然後 SINEWAVE 把两个 WAVEHDR 结构传送给 waveOutWrite, 此函式将资料传送到波形输出硬体, 才真正开始播放声音。

当波形硬体播放完 waveOutWrite 函式传送来的资料後, 就向视窗发送 MM_WOM_DONE 讯息, 其中 wParam 参数是波形输出代号, lParam 是指向 WAVEHDR 结构的指标。SINEWAVE 在处理此讯息时, 将计算缓冲区的新资料, 并呼叫 waveOutWrite 来重新提交缓冲区。

编写 SINEWAVE 程式时也可以只用一个 WAVEHDR 结构和一个缓冲区。不过, 这样在播放完资料後将会有很短暂的停顿, 以等待程式处理 MM_WOM_DONE 讯息来提交新的缓冲区。SINEWAVE 使用的「双缓冲」技术避免了声音的不连续。

当使用者单击「Turn Off」按钮关闭声音时, DlgProc 接收到另一个 WM_COMMAND 讯息。对此讯息, DlgProc 把 bShutOff 变数设定为 TRUE, 并呼叫 waveOutReset 函式。此函式停止处理声音并发送一条 MM_WOM_DONE 讯息。bShutOff 为 TRUE 时, SINEWAVE 透过呼叫 waveOutClose 来处理 MM_WOM_DONE, 从而产生一条 MM_WOM_CLOSE 讯息。处理 MM_WOM_CLOSE 通常包括清除程序。SINEWAVE 为两个 WAVEHDR 结构而呼叫 waveOutUnprepareHeader、释放所有的记忆体块并把按钮上的文字改回「Turn On」。

如果硬体继续播放缓冲区的声音资料, 那么它自己呼叫 waveOutClose 就没有作用。您必须先呼叫 waveOutReset 来停止播放并产生 MM_WOM_DONE 讯息。当 wParam 是 SC_CLOSE 时, DlgProc 也处理 WM_SYSCOMMAND 讯息, 这是因为使用者从系统功能表中选择了「Close」。如果波形声音继续播放, DlgProc 则呼叫 waveOutReset。无论如何, 最後总要呼叫 EndDialog 来结束程式。

数位录音机

Windows 提供了一个称为「录音程式」来录制和播放数位声音。程式 22-3 所示的程式 (RECORD1) 不如「录音程式」完善, 因为它不含有任何档案 I/O, 也不允许声音编辑。然而, 这个程式显示了使用低阶波形声音 API 来录制和重播声音的基本方法。

程式 22-3 RECORD1

```
RECORD1.C
/*-----
-
      RECORD1.C -- Waveform Audio Recorder
                                     (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include "resource.h"

#define INP_BUFFER_SIZE 16384
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows
NT!"),
                                szAppName,
MB_ICONERROR) ;
    }
    return 0 ;
}

void ReverseMemory (BYTE * pBuffer, int iLength)
{
    BYTE  b ;
    int   i ;

    for (i = 0 ; i < iLength / 2 ; i++)
    {
        b = pBuffer [i] ;
        pBuffer [i] = pBuffer [iLength - i - 1] ;
        pBuffer [iLength - i - 1] = b ;
    }
}
```



```

}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      BOOL                      bRecording, bPlaying, bReverse,
bPaused,
                                                bEnding,
bTerminating ;
    static      DWORD                    dwDataLength, dwRepetitions = 1 ;
    static      HWAVEIN                  hWaveIn ;
    static      HWAVEOUT                  hWaveOut ;
    static      PBYTE                    pBuffer1, pBuffer2, pSaveBuffer,
pNewBuffer ;
    static      PWAVEHDR                  pWaveHdr1, pWaveHdr2 ;
    static      TCHAR                    szOpenError[] = TEXT ("Error
opening waveform audio!");
    static      TCHAR                    szMemError [] = TEXT ("Error
allocating memory!"); ;
    static      WAVEFORMATEX waveform ;

    switch (message)
    {
    case WM_INITDIALOG:
        // Allocate memory for wave header

        pWaveHdr1 = malloc (sizeof (WAVEHDR)) ;
        pWaveHdr2 = malloc (sizeof (WAVEHDR)) ;

        // Allocate memory for save buffer

        pSaveBuffer = malloc (1) ;
        return TRUE ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDC_RECORD_BEG:
            // Allocate buffer memory

            pBuffer1 = malloc (INP_BUFFER_SIZE) ;
            pBuffer2 = malloc (INP_BUFFER_SIZE) ;

            if (!pBuffer1 || !pBuffer2)
            {
                if (pBuffer1) free (pBuffer1) ;
                if (pBuffer2) free (pBuffer2) ;
            }
        }
    }
}

```

```

szAppName,
        MessageBeep (MB_ICONEXCLAMATION) ;
        MessageBox (hwnd, szMemError,

        MB_ICONEXCLAMATION | MB_OK) ;
        return TRUE ;
    }

    // Open waveform audio for input

    waveform.wFormatTag      = WAVE_FORMAT_PCM ;
    waveform.nChannels       = 1 ;
    waveform.nSamplesPerSec   = 11025 ;
    waveform.nAvgBytesPerSec  = 11025 ;
    waveform.nBlockAlign     = 1 ;
    waveform.wBitsPerSample   = 8 ;
    waveform.cbSize          = 0 ;

    if (waveInOpen (&hWaveIn, WAVE_MAPPER, &waveform,
        (DWORD) hwnd, 0, CALLBACK_WINDOW))
    {
        free (pBuffer1) ;
        free (pBuffer2) ;
        MessageBeep (MB_ICONEXCLAMATION) ;
        MessageBox (hwnd, szOpenError, szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;
    }

    // Set up headers and prepare them

    pWaveHdr1->lpData      = pBuffer1 ;
    pWaveHdr1->dwBufferLength = INP_BUFFER_SIZE ;
    pWaveHdr1->dwBytesRecorded = 0 ;
    pWaveHdr1->dwUser       = 0 ;
    pWaveHdr1->dwFlags      = 0 ;
    pWaveHdr1->dwLoops      = 1 ;
    pWaveHdr1->lpNext       = NULL ;
    pWaveHdr1->reserved     = 0 ;
    waveInPrepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;

    pWaveHdr2->lpData      = pBuffer2 ;
    pWaveHdr2->dwBufferLength = INP_BUFFER_SIZE ;
    pWaveHdr2->dwBytesRecorded = 0 ;
    pWaveHdr2->dwUser       = 0 ;
    pWaveHdr2->dwFlags      = 0 ;
    pWaveHdr2->dwLoops      = 1 ;
    pWaveHdr2->lpNext       = NULL ;
    pWaveHdr2->reserved     = 0 ;

    waveInPrepareHeader (hWaveIn, pWaveHdr2,

```

```
sizeof (WAVEHDR)) ;

        return TRUE ;

    case IDC_RECORD_END:

        // Reset input to return last buffer

        bEnding = TRUE ;
        waveInReset (hWaveIn) ;
        return TRUE ;

    case IDC_PLAY_BEG:

        // Open waveform audio for output

        waveform.wFormatTag          = WAVE_FORMAT_PCM ;
        waveform.nChannels             = 1 ;
        waveform.nSamplesPerSec        = 11025 ;
        waveform.nAvgBytesPerSec       = 11025 ;
        waveform.nBlockAlign           = 1 ;
        waveform.wBitsPerSample        = 8 ;
        waveform.cbSize                = 0 ;

        if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveform,
            (DWORD) hwnd, 0, CALLBACK_WINDOW))
        {
            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (hwnd, szOpenError, szAppName,
                MB_ICONEXCLAMATION | MB_OK) ;
        }
        return TRUE ;

    case IDC_PLAY_PAUSE:

        // Pause or restart output

        if (!bPaused)
        {
            waveOutPause (hWaveOut) ;
            SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
            bPaused = TRUE ;
        }
        else
        {
            waveOutRestart (hWaveOut) ;
            SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
            bPaused = FALSE ;
        }
        return TRUE ;

    case IDC_PLAY_END:
```

```
// Reset output for close preparation

        bEnding = TRUE ;
        waveOutReset (hWaveOut) ;
        return TRUE ;

case IDC_PLAY_REV:
        // Reverse save buffer and play

        bReverse = TRUE ;
        ReverseMemory (pSaveBuffer, dwDataLength) ;

        SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
        return TRUE ;

case IDC_PLAY_REP:
        // Set infinite repetitions and play

        dwRepetitions = -1 ;
        SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
        return TRUE ;

case IDC_PLAY_SPEED:
        // Open waveform audio for fast output

        waveform.wFormatTag      = WAVE_FORMAT_PCM ;
        waveform.nChannels        = 1 ;
        waveform.nSamplesPerSec   = 22050 ;
        waveform.nAvgBytesPerSec  = 22050 ;
        waveform.nBlockAlign      = 1 ;
        waveform.wBitsPerSample   = 8 ;
        waveform.cbSize           = 0 ;

        if (waveOutOpen (&hWaveOut, 0, &waveform, (DWORD) hwnd, 0,
CALLBACK_WINDOW))
        {
                essageBeep (MB_ICONEXCLAMATION) ;
                MessageBox (hwnd, szOpenError, szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        }
        return TRUE ;
    }
    break ;

case MM_WIM_OPEN:
        // Shrink down the save buffer

        pSaveBuffer = realloc (pSaveBuffer, 1) ;

        // Enable and disable buttons
```

```

        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), FALSE) ;
        SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

        // Add the buffers

        waveInAddBuffer (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
        waveInAddBuffer (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

        // Begin sampling

        bRecording = TRUE ;
        bEnding = FALSE ;
        dwDataLength = 0 ;
        waveInStart (hWaveIn) ;
        return TRUE ;

case MM_WIM_DATA:

        // Reallocate save buffer memory

        pNewBuffer = realloc ( pSaveBuffer, dwDataLength +
((PWAVEHDR) lParam)->dwBytesRecorded) ;

        if (pNewBuffer == NULL)
        {
                waveInClose (hWaveIn) ;
                MessageBeep
(MB_ICONEXCLAMATION) ;
                MessageBox (hwnd, szMemError, szAppName,
MB_ICONEXCLAMATION | MB_OK) ;
                return TRUE ;
        }

        pSaveBuffer = pNewBuffer ;
        CopyMemory (pSaveBuffer + dwDataLength, ((PWAVEHDR) lParam)->lpData,
((PWAVEHDR) lParam)->dwBytesRecorded) ;

        dwDataLength += ((PWAVEHDR) lParam)->dwBytesRecorded ;

        if (bEnding)

```

```

        {
            waveInClose (hWaveIn) ;
            return TRUE ;
        }

        // Send out a new buffer

waveInAddBuffer (hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR)) ;
return TRUE ;

case MM_WIM_CLOSE:

        // Free the buffer memory

waveInUnprepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
waveInUnprepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

        free (pBuffer1) ;
        free (pBuffer2) ;

        // Enable and disable buttons

        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE) ;
        SetFocus (GetDlgItem (hwnd, IDC_RECORD_BEG)) ;

        if (dwDataLength > 0)
        {
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), TRUE) ;
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), TRUE) ;
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), TRUE) ;
            SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;
        }
        bRecording = FALSE ;

        if (bTerminating)
            SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

        return TRUE ;

case MM_WOM_OPEN:

        // Enable and disable buttons

        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE) ;
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE) ;

```

```

EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), TRUE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), TRUE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), FALSE);
SetFocus (GetDlgItem (hwnd, IDC_PLAY_END));

// Set up header

pWaveHdr1->lpData = pSaveBuffer;
pWaveHdr1->dwBufferLength = dwDataLength;
pWaveHdr1->dwBytesRecorded = 0;
pWaveHdr1->dwUser = 0;
pWaveHdr1->dwFlags = WHDR_BEGINLOOP | WHDR_ENDLOOP;
pWaveHdr1->dwLoops = dwRepetitions;
pWaveHdr1->lpNext = NULL;
pWaveHdr1->reserved = 0;

// Prepare and write

waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR));
waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR));

bEnding = FALSE;
bPlaying = TRUE;
return TRUE;

case MM_WOM_DONE:
    waveOutUnprepareHeader (hWaveOut, pWaveHdr1, sizeof
(WAVEHDR));

    waveOutClose (hWaveOut);
    return TRUE;

case MM_WOM_CLOSE:

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), TRUE);
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG));

    SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause"));
    bPaused = FALSE;

```

```
        dwRepetitions = 1 ;
        bPlaying = FALSE ;

        if (bReverse)
        {
            ReverseMemory (pSaveBuffer, dwDataLength) ;
            bReverse = FALSE ;
        }

        if (bTerminating)
            SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

        return TRUE ;

case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
    case SC_CLOSE:
        if (bRecording)
        {
            bTerminating = TRUE ;
            bEnding = TRUE ;
            waveInReset (hWaveIn) ;
            return TRUE ;
        }
        if (bPlaying)
        {
            bTerminating = TRUE ;
            bEnding = TRUE ;
            waveOutReset (hWaveOut) ;
            return TRUE ;
        }

        free (pWaveHdr1) ;
        free (pWaveHdr2) ;
        free (pSaveBuffer) ;
        EndDialog (hwnd, 0) ;
        return TRUE ;
    }
    break ;
}

return FALSE ;
}

RECORD.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
```



```

////////////////////////////////////
/
// Dialog
RECORD          DIALOG DISCARDABLE 100, 100, 152, 74
STYLE            WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION          "Waveform Audio Recorder"
FONT 8,          "MS Sans Serif"
BEGIN
    PUSHBUTTON    "Record", IDC_RECORD_BEG, 28, 8, 40, 14
    PUSHBUTTON    "End", IDC_RECORD_END, 76, 8, 40, 14, WS_DISABLED
    PUSHBUTTON    "Play", IDC_PLAY_BEG, 8, 30, 40, 14, WS_DISABLED
    PUSHBUTTON    "Pause", IDC_PLAY_PAUSE, 56, 30, 40, 14, WS_DISABLED
    PUSHBUTTON    "End", IDC_PLAY_END, 104, 30, 40, 14, WS_DISABLED
    PUSHBUTTON    "Reverse", IDC_PLAY_REV, 8, 52, 40, 14, WS_DISABLED
    PUSHBUTTON    "Repeat", IDC_PLAY_REP, 56, 52, 40, 14, WS_DISABLED
    PUSHBUTTON    "Speedup", IDC_PLAY_SPEED, 104, 52, 40, 14, WS_DISABLED
END
RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by Record.rc

#define IDC_RECORD_BEG 1000
#define IDC_RECORD_END 1001
#define IDC_PLAY_BEG 1002
#define IDC_PLAY_PAUSE 1003
#define IDC_PLAY_END 1004
#define IDC_PLAY_REV 1005
#define IDC_PLAY_REP 1006
#define IDC_PLAY_SPEED 1007

```

RECORD.RC 和 RESOURCE.H 档案也在 RECORD2 和 RECORD3 程式中使用。

RECORD1 视窗有 8 个按钮。第一次执行 RECORD1 时，只有「Record」按钮有效。按下「Record」後，就开始录音，这时「Record」按钮无效，而「End」按钮有效。按下「End」可停止录音。这时，「Play」、「Reverse」、「Repeat」和「Speedup」也都有效，选择任一个按钮都可重放声音：「Play」表示正常播放；「Reverse」表示反向播放；「Repeat」表示无限的重复播放（好像回圈录音带）；「Speedup」以正常速度的两倍来播放。要停止播放，您可以选择「End」按钮，而按下「Pause」按钮可停止播放。按下後，「Pause」按钮将变为「Resume」按钮，用於继续播放声音。如果要录制另一段声音，新录制的声音将替换记忆体里现有的声音。

任何时候，有效按钮都是可以执行有效操作的按钮。这需要在 RECORD1 原始码中包括对 EnableWindow 的多次呼叫，但是程式并不检查具体的按钮操作是否有效。显然，这使得程式操作更为直观。

RECORD1 用了许多快捷方式来简化程式码。首先，如果安装了多个波形声音硬件设备，则 RECORD1 只使用内定设备。其次，程式按标准的 11.025 kHz 的取样频率和 8 位元的取样精确度来录音和放音，而不管设备能否提供更高的取样频率和取样精确度。唯一的例外是加速功能，加速时 RECORD1 按 22.050kHz 的取样频率播放声音，这样不仅播放速度提高了一倍，而且频率也提高了一个音阶。

录制声音既包括为输入而打开波形声音硬体，还包括将缓冲区传递给 API，以便接收声音资料。

RECORD1 设有几个记忆体块。其中三个很小，至少在初始化时很小，并且在 DlgProc 的 WM_INITDIALOG 讯息处理期间进行配置。程式配置两个 WAVEHDR 结构，分别由指标 pWaveHdr1 和 pWaveHdr2 指向。这两个结构用於将缓冲区传递给波形 API。pSaveBuffer 指标指向储存整个录音的缓冲区，最初配置时只有一个位元组。然後，随著录音的进行，该缓冲区不断增大，以适应所有的声音资料（如果录音时间过长，则 RECORD1 能够在录制程序中及时发现记忆体溢出，并允许您重放成功储存的声音）。由於这个缓冲区用来储存堆积的声音资料，所以我将其称为「储存缓冲区 (save buffer)」。指标 pBuffer1 和 pBuffer2 指向的另外两个记忆体块，大小是 16K，它们在记录接收的声音资料时配置。录音结束後释放这些记忆体块。

8 个按钮中的每一个都向 REPORT1 视窗的对话程序 DlgProc 产生 WM_COMMAND 讯息。最初只有「Record」按钮有效。按下此按钮将产生 WM_COMMAND 讯息，其中 wParam 参数等於 IDC_RECORD_BEG。为处理这个讯息，RECORD1 配置两个 16K 的缓冲区来接收声音资料，初始化 WAVEFORMATEX 结构的栏位，并将此结构传递给 waveInOpen 函式，然後设定两个 WAVEHDR 结构。

waveInOpen 函式产生一条 MM_WIM_OPEN 讯息。在此讯息处理期间，RECORD1 把储存缓冲区的大小缩减到 1 个位元组，以准备接收资料（当然，第一次录音时，储存缓冲区的大小就是 1 个位元组，但以後录制时，就可能大多了）。在 MM_WIM_OPEN 讯息处理期间，RECORD1 也将适当的按钮设定为有效和无效。然後，程式用 waveInAddBuffer 把两个 WAVEHDR 结构和缓冲区传送给 API。这时会设定某些标记，然後呼叫 waveInStart 开始录音。

采用 11.025kHz 的取样频率和 8 位元的取样精确度时，16K 的缓冲区可储存大约 1.5 秒的声音。这时，RECORD1 接收 MM_WIM_DATA 讯息。在回应此讯息处理期间，程式将根据变数 dwDataLength 和 WAVEHDR 结构中的栏位 dwBytesRecorded 对缓冲区重新配置。如果配置失败，RECORD1 呼叫 waveInClose 来停止录音。

如果重新配置成功，则 RECORD1 把 16K 缓冲区里的资料复制到储存缓冲区，

然後再次呼叫 `waveInAddBuffer`。此程序将持续到 `RECORD1` 用完储存缓冲区的记忆体，或使用者按下「End」按钮为止。

「End」按钮产生 `WM_COMMAND` 讯息，其中 `wParam` 等於 `IDC_RECORD_END`。处理这个讯息很简单，`RECORD1` 把 `bEnding` 标记设定为 `TRUE` 并呼叫 `waveInReset`。`waveInReset` 函式使录音停止，并产生 `MM_WIM_DATA` 讯息，该讯息含有部分填充的缓冲区。除了呼叫 `waveInClose` 来关闭波形输入设备外，`RECORD1` 对这个讯息正常回应。

`waveInClose` 产生 `MM_WIM_CLOSE` 讯息。`RECORD1` 回应此讯息时，释放 16K 输入缓冲区，并使相应的按钮有效或无效。尤其是，当储存缓冲区里存有资料（除非第一次配置就失败，否则一般都含有资料）时，播放按钮将有效。

录音以後，储存缓冲区里将含有这些声音资料。当使用者选择「Play」按钮时，`DlgProc` 就接收一个 `WM_COMMAND` 讯息，其中 `wParam` 等於 `IDC_PLAY_BEG`。回应时，程式将初始化 `WAVEFORMATEX` 结构的栏位，并呼叫 `waveOutOpen`。

`waveOutOpen` 呼叫再次产生 `MM_WOM_OPEN` 讯息，在此讯息处理期间，`RECORD1` 把相应的按钮设为有效或无效（只允许使用「Pause」和「End」），用储存缓冲区来初始化 `WAVEHDR` 结构的栏位，呼叫 `waveOutPrepareHeader` 来准备要播放的声音，然後呼叫 `waveOutWrite` 开始播放。

一般情况下，直到播放完储存缓冲区里的所有资料才停止。这时产生 `MM_WOM_DONE` 讯息。如果还有缓冲区要播放，则程式会在这时将它们传递给 API。由於 `RECORD1` 只播放一个大缓冲区，因此程式不再简单地准备标题，而是呼叫 `waveOutClose`。`waveOutClose` 函式产生 `MM_WOM_CLOSE` 讯息。在此讯息处理期间，`RECORD1` 使相应的按钮有效或无效，并允许声音再次播放或者录制新声音。

程式中还有一个「End」按钮，利用此按钮，使用者可以在播放完储存缓冲区之前的任何时刻停止播放。「End」按钮产生一个 `WM_COMMAND` 讯息，其中 `wParam` 等於 `IDC_PLAY_END`，回应时，程式呼叫 `waveOutReset`，此函式产生一条正常处理的 `MM_WOM_DONE` 讯息。

`RECORD1` 的视窗中还包括一个「Pause」按钮。处理此按钮很简单：第一次按时下，`RECORD1` 呼叫 `waveOutPause` 来暂停播放，并将按钮上的文字改为「Resume」。按下「Resume」按钮时，通过呼叫 `waveOutRestart` 来继续播放。

为了使程式更有趣，视窗中还包括另外三个按钮：「Reverse」、「Repeat」和「Speedup」。这些按钮都产生 `WM_COMMAND` 讯息，其中 `wParam` 的值分别等於 `IDC_PLAY_REV`、`IDC_PLAY_REP` 和 `IDC_PLAY_SPEED`。

倒放声音就是把储存缓冲区里的资料按位元组顺序反向，然後再正常播放。`RECORD1` 中有一个称为 `ReverseMemory` 的小函式使位元组反向。在 `WM_COMMAND`

讯息处理期间，程式在播放块之前呼叫此函式，并在 MM_WOM_CLOSE 讯息的後期再次呼叫此函式，以便将其恢复到正常状态。

「Repeat」按钮将往复不停地播放声音。由於 API 支援重复播放声音，所以这并不复杂。只要将 WAVEHDR 结构的 dwLoops 栏位设为重复次数，将 dwFlags 栏位设为 WHDR_BEGINLOOP 和 WHDR_ENDLOOP，分别表示回圈时缓冲区的开始部分和结束部分。因为 RECORD1 只使用一个缓冲区来播放声音，所以这两个标记组合到了 dwFlags 栏位。

要实作两倍速播放也很容易。在准备为输出而打开波形声音期间，初始化 WAVEFORMATEX 结构的栏位时，只需将 nSamplesPerSec 和 nAvgBytesPerSec 栏位设定为 22050，而不是 11025。

另一种 MCI 介面

您可能已经发现，RECORD1 很复杂。特别是在处理波形声音函式呼叫和它们产生的讯息间的交互时，更复杂。处理可能出现的记忆体不足的情况也是如此。但这也许正是它称为低阶介面的原因。我在本章的前面提到过，Windows 也提供高阶媒体控制介面 (Media Control Interface)。

对波形声音来说，低阶介面与 MCI 之间的主要区别在於 MCI 用波形档案记录声音资料，并通过读取档案来播放声音。由於在播放声音之前要读取档案、处理档案然後再写入档案，所以让 RECODE1 来实作「特殊效果」很困难。这是典型的折衷选择问题：功能齐全或是使用方便？低阶介面很灵活，但 MCI（其中的大部分）更方便。

MCI 有两种不同但又相关的实作形式。一种形式用讯息和资料结构将命令发送给多媒体设备，然後再从那里接收资讯。另一种形式使用 ASCII 文字字符串。建立文字命令的介面最初是为了让多媒体设备接受简单的描述命令语言的控制。但它也提供非常容易的交谈式控制，请参见本章前面，TESTMCI 程式的展示。

RECORD2 程式，如程式 22-4 所示，使用 MCI 形式的讯息和资料结构来实作另一个数位声音录音机和播放器。虽然它使用的对话方块模板与 RECORD1 一样，但并没有实作三个特殊效果的按钮。

程式 22-4 RECORD2

```
RECORD2.C
/*-----
-
-
- RECORD2.C -- Waveform Audio Recorder
-
- (c) Charles Petzold, 1998
-
-----*/
```

```

#include <windows.h>
#include "..\record1\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int
iCmdShow)
{
    if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
    }
    return 0 ;
}

void ShowError (HWND hwnd, DWORD dwError)
{
    TCHAR szErrorStr [1024] ;
    mciGetErrorString (dwError, szErrorStr, sizeof (szErrorStr) / sizeof
(TCHAR)) ;
    MessageBeep (MB_ICONEXCLAMATION) ;
    MessageBox (hwnd, szErrorStr, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

BOOL CALLBACK DlgProc (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      BOOL      bRecording, bPlaying, bPaused ;
    static      TCHAR      szFileName[] = TEXT ("record2.wav") ;
    static      WORD      wDeviceID ;
    DWORD
dwError ;
    MCI_GENERIC_PARMS      mciGeneric ;
    MCI_OPEN_PARMS          mciOpen ;
    MCI_PLAY_PARMS          mciPlay ;
    MCI_RECORD_PARMS        mciRecord ;
    MCI_SAVE_PARMS          mciSave ;

    switch (message)
    {
    case WM_COMMAND:
        switch (wParam)
        {
        case IDC_RECORD_BEG:
            // Delete existing waveform file

            DeleteFile (szFileName) ;

```

```

// Open waveform audio

mciOpen.dwCallback          = 0 ;
mciOpen.wDeviceID           = 0 ;
mciOpen.lpstrDeviceType=TEXT ("waveaudio") ;
mciOpen.lpstrElementName    = TEXT ("") ;
mciOpen.lpstrAlias           = NULL ;
dwError = mciSendCommand (0, MCI_OPEN,
MCI_WAIT | MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
(DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;
    if (dwError != 0)
    {
        ShowError (hwnd, dwError) ;
        return TRUE ;
    }

// Save the Device ID

wDeviceID = mciOpen.wDeviceID ;

// Begin recording

mciRecord.dwCallback  = (DWORD) hwnd ;
mciRecord.dwFrom = 0 ;
mciRecord.dwTo       = 0 ;

mciSendCommand (wDeviceID, MCI_RECORD, MCI_NOTIFY,
(DWORD) (LPMCI_RECORD_PARMS) &mciRecord) ;

// Enable and disable buttons

EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE) ;
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE) ;
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

bRecording = TRUE ;
return TRUE ;

case IDC_RECORD_END:

// Stop recording

mciGeneric.dwCallback = 0 ;

mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
(DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

```

```

// Save the file

mciSave.dwCallback = 0 ;
mciSave.lpfilename = szFileName ;

mciSendCommand (wDeviceID, MCI_SAVE, MCI_WAIT |
MCI_SAVE_FILE,
(DWORD) (LPMCI_SAVE_PARMS) &mciSave) ;

//Close the waveform device

mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
(DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

// Enable and disable buttons

EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE);
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

bRecording = FALSE ;
return TRUE ;

case IDC_PLAY_BEG:
// Open waveform audio

mciOpen.dwCallback = 0 ;
mciOpen.wDeviceID = 0 ;
mciOpen.lpstrDeviceType = NULL ;
mciOpen.lpstrElementName = szFileName ;
mciOpen.lpstrAlias = NULL ;

dwError = mciSendCommand ( 0, MCI_OPEN,
MCI_WAIT | MCI_OPEN_ELEMENT,
(DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;

if (dwError != 0)
{
ShowError (hwnd, dwError) ;
return TRUE ;
}

// Save the Device ID

wDeviceID = mciOpen.wDeviceID ;

```

```

                                                // Begin playing

        mciPlay.dwCallback      = (DWORD) hwnd ;
        mciPlay.dwFrom          = 0 ;
        mciPlay.dwTo            = 0 ;

        mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
            (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

                                                // Enable and disable buttons

        EnableWindow      (GetDlgItem      (hwnd, IDC_RECORD_BEG), FALSE);
        EnableWindow (GetDlgItem      (hwnd, IDC_RECORD_END), FALSE);
        EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_BEG),  FALSE);
        EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_PAUSE), TRUE) ;
        EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_END),  TRUE) ;
        SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

        bPlaying = TRUE ;
        return TRUE ;

        case IDC_PLAY_PAUSE:
            if (!bPaused)
                // Pause the play
            {
                mciGeneric.dwCallback = 0 ;

                mciSendCommand (wDeviceID, MCI_PAUSE, MCI_WAIT,
                    (DWORD) (LPMCI_GENERIC_PARMS) & mciGeneric);

                SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
                Paused = TRUE ;
            }
            else
                // Begin playing again
            {
                mciPlay.dwCallback      = (DWORD) hwnd ;
                mciPlay.dwFrom          = 0 ;
                mciPlay.dwTo            = 0 ;

                mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
                    (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

                SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
                bPaused = FALSE ;
            }
    }
```



```
        return TRUE ;

    case IDC_PLAY_END:

        // Stop and close

        mciGeneric.dwCallback = 0 ;

        mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
            (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

        mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
            (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

        // Enable and disable buttons

        EnableWindow (GetDlgItem(hwnd, IDC_RECORD_BEG), TRUE) ;
        EnableWindow (GetDlgItem(hwnd, IDC_RECORD_END), FALSE) ;
        EnableWindow (GetDlgItem(hwnd, IDC_PLAY_BEG), TRUE) ;
        EnableWindow (GetDlgItem(hwnd, IDC_PLAY_PAUSE), FALSE) ;
        EnableWindow (GetDlgItem(hwnd, IDC_PLAY_END), FALSE) ;
        SetFocus (GetDlgItem(hwnd, IDC_PLAY_BEG)) ;

        bPlaying = FALSE ;
        bPaused = FALSE ;
        return TRUE ;
    }
    break ;

case MM_MCINOTIFY:
    switch (wParam)
    {
        case MCI_NOTIFY_SUCCESSFUL:
            if (bPlaying)
                SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0) ;

            if (bRecording)
                SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0) ;

            return TRUE ;
    }
    break ;

case WM_SYSCOMMAND:
    switch (wParam)
    {
        case SC_CLOSE:
            if (bRecording)
                SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L) ;
```

```

        if (bPlaying)
            SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

            EndDialog (hwnd, 0) ;
            return TRUE ;
        }
        break ;
    }
    return FALSE ;
}

```

RECORD2 只使用两个 MCI 函式呼叫，其中最重要的呼叫如下所示：

```
error = mciSendCommand (wDeviceID, message, dwFlags, dwParam)
```

第一个参数是设备的识别数字 (ID)，您可以按代号来使用 ID。打开设备时就可以获得 ID，并在随后的 mciSendCommand 呼叫中使用。第二个参数是字首为 MCI 的常数。这些称为 MCI 命令讯息，RECORD2 展示了其中的七个：MCI_OPEN、MCI_RECORD、MCI_STOP、MCI_SAVE、MCI_PLAY、MCI_PAUSE 和 MCI_CLOSE。

dwFlags 参数通常由 0 或者多个位元旗标常数 (由 C 的位元 OR 运算符合成) 组成。这些通常用来表示不同的选项。一些选项是某个命令讯息所特有的，而另一些对所有的讯息都是通用的。dwParam 参数通常是指向一个资料结构的长指标，该结构表示选项以及由设备获得的资讯。许多 MCI 讯息都与资料结构有关，而且这些资料结构对于讯息来说都是唯一的。

如果 mciSendCommand 函式呼叫成功，则传回 0 值，否则传回错误代码。要向使用者报告此错误，可用下面的函式获得描述错误的文字字串：

```
mciGetErrorString (error, szBuffer, dwLength)
```

此函式在程式 TESTMCI 中也用到过。

按下「Record」按钮后，RECORD2 的视窗讯息处理程式就收到一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_BEG。RECORD2 从打开设备开始，包括设定 MCI_OPEN_PARMS 结构的栏位，并用 MCI_OPEN 命令讯息呼叫 mciSendCommand。录音时，lpstrDeviceType 栏位设定为字串「waveaudio」以说明设备型态，lpstrElementName 栏位设定为长度为 0 的字串。MCI 驱动程序使用内定的取样频率和取样精确度，但是您可以用 MCI_SET 命令进行修改。录音程序中，声音资料先储存在硬碟上的暂存档案中，最后再转化成标准的波形档案。本章的后面将介绍波形档案的格式。播放录制的声音时，MCI 使用波形档案中定义的取样频率和取样精确度。

如果 RECORD2 不能打开设备，则用 mciGetErrorString 和 MessageBox 提示错误资讯。否则从 mciSendCommand 呼叫传回，MCI_OPEN_PARMS 结构的 wDeviceID 栏位包含有设备 ID，以供后面的呼叫使用。

要开始录音，RECORD2 就呼叫 mciSendCommand，以 MCI_RECORD 命令讯息和

MCI_WAVE_RECORD_PARMS 资料结构为参数。当然，您也可以将此结构（并使用表示这些栏位已设定的位元旗标）的 dwFromz 和 dwTo 栏位进行设定，以便将声音插入现有的波形档案，其档案名在 MCI_OPEN_PARMS 结构的 lpstrElementName 栏位指定。内定状态下，任何新的声音都插入在现有档案的开始位置。

RECORD2 将 MCI_WAVE_RECORD_PARMS 结构的 dwCallback 栏位设定为程式的视窗代号，并在 mciSendCommand 呼叫中包含 MCI_NOTIFY 标记。这导致录音结束後向视窗讯息处理程式发送一条通知讯息。我将简要讨论一下这条通知讯息。

录音结束後，按下前一个「End」按钮来停止录音，这时产生一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_END。回应时，视窗讯息处理程式将呼叫 mciSendCommand 三次：MCI_STOP 命令讯息用於停止录音；MCI_SAVE 命令讯息用於把暂存档案中的声音资料传递到 MCI_SAVE_PARMS 结构中指定的档案（「record2.wav」）；MCI_CLOSE 命令讯息用於删除所有的暂存档案、释放已经建立的记忆体块并关闭设备。

播放时，MCI_OPEN_PARMS 结构的 lpstrElementName 栏位设定为档案名「record2.wav」。mciSendCommand 第三个参数中所包含的 MCI_OPEN_ELEMENT 标记表示 lpstrElementName 栏位是一个有效的档案名。通过档案的副档名称.WAV，MCI 知道使用者要打开一个波形声音设备。如果存在多个波形硬体，则打开第一个（设定 MCI_OPEN_PARMS 结构的 lpstrDeviceType 栏位，也可以打开其他波形设备）。

播放将包括带有 MCI_PLAY 命令讯息和 MCI_PLAY_PARMS 结构的 mciSendCommand 呼叫。虽然波形档案的任意部分都可以播放，但 RECORD2 只播放整个档案。

RECORD2 还包括一个「Pause」按钮来暂停播放音效档案。这个按钮产生一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_PAUSE。回应时，程式将呼叫 mciSendCommand，并以 MCI_PAUSE 命令讯息和 MCI_GENERIC_PARMS 结构作为参数。MCI_GENERIC_PARMS 结构用於这样一些讯息：它们除了需要用於通知的可选视窗代号外，不需要任何资讯。如果播放已经暂停，则通过再次使用 MCI_PLAY 命令讯息呼叫 mciSendCommand 继续播放。

按下第二个「End」按钮也可以停止播放。这时产生 wParam 等於 IDC_PLAY_END 的 WM_COMMAND 讯息。回应时，视窗讯息处理程式将呼叫 mciSendCommand 两次：第一次使用 MCI_STOP 命令讯息；第二次使用 MCI_CLOSE 命令讯息。

现在有一个问题：虽然可以通过按下「End」按钮来手工终止播放，但您可能需要播放整个档案。程式如何知道档案播放完的时间呢？这是 MCI 通知讯息的任务。

当带有 MCI_RECORD 和 MCI_PLAY 讯息来呼叫 mciSendCommand 时，RECORD2 将包括 MCI_NOTIFY 标记，并将资料结构的 dwCallback 栏位设定为程式视窗代号。这样就产生一个通知讯息，称为 MM_MCINOTIFY，并在某些环境下传递给视窗讯息处理程式。讯息参数 wParam 是一个状态代码，而 lParam 是设备 ID。

带有 MCI_STOP 或者 MCI_PAUSE 命令讯息来呼叫 mciSendCommand 时，您将接收到一个 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_ABORTED。当您按下「Pause」按钮或者两个「End」按钮中的一个时，就会出现这种情况。由於对这些按钮已进行过适当的处理，所以 RECORD2 可以忽略这种情况。播放时，您会在音效档案结束後接收到 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_SUCCESSFUL。这种情况下，视窗讯息处理程式给自己发送一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_END，来模拟使用者按下「End」按钮。然後视窗讯息处理程式作出正常回应：停止播放，关闭设备。

录音时，如果用於储存暂存档案的硬碟空间不够，您就会接收一个 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_SUCCESSFUL（虽然现在还不能说它很完美，但其功能已经很齐全了）。回应时，视窗讯息处理程式给自己发送一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_END，然後与正常情况下一样：停止录音、储存档案并关闭设备。

MCI 命令字串的方法

Windows 的多媒体介面曾经包含函式 mciExecute，其语法如下：

```
bSuccess = mciExecute (szCommand) ;
```

其中唯一的参数是 MCI 命令字串。函式传回布林值——如果呼叫成功，则传回非 0 值，否则传回 0。在功能上，mciExecute 函式相同於呼叫後三个参数为 NULL 或 0 的 mciSendString（TESTMCI 中使用的依据字串的 MCI 函式），然後在发生错误时呼叫 mciGetErrorString 和 MessageBox。

虽然 mciExecute 不再是 API 的一部分，但我还是在 RECORD3 版的数位录音机中使用了这个函式。和 RECORD2 一样，RECORD3 程式也使用 RECORD1 中的资源描述档 RECORD.RC 和 RESOURCE.H，如程式 22-5 所示。

程式 22-5 RECORD3

```
RECORD3.C
/*-----
-
-
- RECORD3.C -- Waveform Audio Recorder
-
- (c) Charles Petzold, 1998
-
*/
```

```

#include <windows.h>
#include "..\\record1\\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
    }
    return 0 ;
}

BOOL mciExecute (LPCTSTR szCommand)
{
    MCIERROR error ;
    TCHAR      szErrorStr [1024] ;

    if (error = mciSendString (szCommand, NULL, 0, NULL))
    {
        mciGetErrorString (error, szErrorStr, sizeof (szErrorStr) /
sizeof (TCHAR)) ;
        MessageBeep (MB_ICONEXCLAMATION) ;
        MessageBox (      NULL, szErrorStr, TEXT ("MCI Error"),
                        MB_OK
MB_ICONEXCLAMATION) ;
    }
    return error == 0 ;
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static BOOL bRecording, bPlaying, bPaused ;
    switch (message)
    {
    case WM_COMMAND:
        switch (wParam)
        {
        case IDC_RECORD_BEG:
            // Delete existing waveform file

            DeleteFile (TEXT ("record3.wav")) ;

```

```
                                // Open waveform audio and record

if (!mciExecute (TEXT ("open newtype waveaudio alias mysound")))
    return TRUE ;

    mciExecute (TEXT ("record mysound")) ;

                                // Enable and disable buttons

EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

    bRecording = TRUE ;
    return TRUE ;

case IDC_RECORD_END:
                                // Stop, save, and close recording

    mciExecute (TEXT ("stop mysound")) ;
    mciExecute (TEXT ("save mysound record3.wav")) ;
    mciExecute (TEXT ("close mysound")) ;

                                // Enable and disable buttons

EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

    bRecording = FALSE ;
    return TRUE ;

case IDC_PLAY_BEG:
                                // Open waveform audio and play

if (!mciExecute (TEXT ("open record3.wav alias mysound")))
    return TRUE ;

    mciExecute (TEXT ("play mysound")) ;

                                // Enable and disable buttons
```

```
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), TRUE);
EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), TRUE);
SetFocus (GetDlgItem (hwnd, IDC_PLAY_END));

    bPlaying = TRUE;
    return TRUE;

    case IDC_PLAY_PAUSE:
        if (!bPaused)
            // Pause the play
            {
                mciExecute (TEXT ("pause mysound"));
                SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume"));
                bPaused = TRUE;
            }
        else
            // Begin playing again
            {
                mciExecute (TEXT ("play mysound"));
                SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause"));
                bPaused = FALSE;
            }

        return TRUE;

    case IDC_PLAY_END:
        // Stop and close

        mciExecute (TEXT ("stop mysound"));
        mciExecute (TEXT ("close mysound"));

        // Enable and disable buttons
        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE);
        EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE);
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
        EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
        SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG));

        bPlaying = FALSE;
        bPaused = FALSE;
        return TRUE;
    }
    break;
```

```
case WM_SYSCOMMAND:
    switch (wParam)
    {
        case SC_CLOSE:
            if (bRecording)
                SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L);

            if (bPlaying)
                SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

            EndDialog (hwnd, 0) ;
            return TRUE ;
        }
    break ;
}
return FALSE ;
}
```

在研究讯息导向和文字导向的 MCI 介面时，您会发现它们非常相近。很容易就可以猜测出 MCI 将命令字符串转换为相应的命令讯息和资料结构。RECORD3 可以使用像 RECORD2 一样使用 MM_MCINOTIFY 讯息，但是它没有选择 mciExecute 函式的好处，它的缺点是程式不知道什么时候播放完波形档案。因此，这些按钮不能自动改变状态。您必须人工按下「End」按钮，以便让程式知道它已经准备再次录音或播放。

注意 MCI 的 open 命令中 alias 关键字的用法。它允许所有后来的 MCI 命令使用别名来引用设备。

波形声音档案格式

如果在十六进位转储程式下研究未压缩的 WAV 档案（即 PCM），您会发现它们具有表 22-1 所示的格式。

表 22-1 .WAV 档案格式

偏移量	位元组	资料
0000	4	「RIFF」
0004	4	波形块的大小（档案大小减 8）
0008	4	「WAVE」
000C	4	「fmt 」
0010	4	格式块的大小（16 位元组）
0014	2	wf.wFormatTag = WAVE_FORMAT_PCM = 1
0016	2	wf.nChannels

0018	4	wf.nSamplesPerSec
001C	4	wf.nAvgBytesPerSec
0020	2	wf.nBlockAlign
0022	2	wf.wBitsPerSample
0024	4	[data]
0028	4	波形资料的大小
002C		波形资料

这是一种扩充自 RIFF (Resource Interchange File Format: 资源交换档案格式) 的格式。RIFF 是用于多媒体资料档案的万用格式, 它是一种标记档案格式。在这种格式下, 档案由资料「块」组成, 而这些资料块则由前面 4 个字元的 ASCII 名称和 4 位元组 (32 位元) 的资料块大小来确认。资料块大小值不包括名称和大小所需要的 8 位元组。

波形声音档案以文字字串「RIFF」开始, 用来标识这是一个 RIFF 档案。字串後面是一个 32 位元的资料块大小, 表示档案其余部分的大小, 或者是小于 8 位元组的档案大小。

资料块以文字字串「WAVE」开始, 用来标识这是一个波形声音块, 後面是文字字串「fmt」——注意用空白使之成为 4 字元的字串——用来标识包含波形声音资料格式的子资料块。「fmt」字串的后部是格式资讯大小, 这里是 16 位元组。格式资讯是 WAVEFORMATEX 结构的前 16 个位元组, 或者, 像最初定义时一样, 是包含 WAVEFORMAT 结构的 PCMWAVEFORMAT 结构。

nChannels 栏位的值是 1 或 2, 分别对应于单声道和立体声。nSamplesPerSec 栏位是每秒的样本数; 标准值是每秒 11,025、22,050 和 44,100 个样本。nAvgBytesPerSec 栏位是取样速率, 单位是每秒样本数乘以通道数, 再乘以以位元为单位的每个样本的大小, 然后除以 8 并往上取整数。标准样本大小是 8 位元和 16 位元。nBlockAlign 栏位是通道数乘以以位元为单位的样本大小, 然后除以 8 并往上取整数。最后, 该格式以 wBitsPerSample 栏位结束, 该栏位是通道数乘以以位元为单位的样本大小。

格式资讯的后部是文字字串「data」, 然后是 32 位元的资料大小, 最后是波形资料本身。这些资料是按相同格式进行简单连结的样本, 这与低阶波形声音设备上所使用的格式相同。如果样本大小是 8 位元, 或者更少, 那么每个样本有 1 位元组用于单声道, 或者有 2 位元组用于立体声。如果样本大小在 9 到 16 位元之间, 则每个样本就有 2 位元组用于单声道, 或者 4 位元组用于立体声。对于立体声波形资料, 每个样本都由左值及其後面的右值组成。

对于 8 位元或不到 8 位元的样本大小, 样本位元组被解释为无正负号值。

例如，對於 8 位元的样本大小，静音等於 0x80 位元组的字串。對於 9 位元或更多的样本大小，样本被解释为有正负号值，这时静音的字串等於值 0。

用於读取标记档案的一个重要规则是忽略不准备处理的资料块。尽管波形声音档案需要「fmt」和「data」子资料块（按照此顺序），但它还包含其他子资料块。尤其是，波形声音档案可能包含一个标记为「INFO」的子资料块，和提供波形声音档案资讯的子资料块的子资料块。

叠加合成实验

许多年来——至少从毕达哥拉斯的年代起——人们就已经试图分析音调。起初好像非常简单，但隨後就变得复杂了。抱歉，我将重复一些已经说过的有关声音的问题。

音调，除了一些撞击声以外，都有特殊的音调或频率。这个频率可以在人类能够感受到的频谱范围内，也就是从 20Hz 到 20,000Hz 以内。例如，钢琴的频率范围在 27.5Hz 到 4186Hz 之间。音调的另一个特徵是音量或响度。这与产生音调的波形的所有振幅相对应。响度的变化用分贝度量。迄今为止，一切都很好。

然後有一件难办的事称做「音质」。非常简单，音质就是声音的性质，利用它，我们可以区分按相同音调相同音量演奏的钢琴、小提琴和喇叭。

法国数学家 Fourier 发现一些周期性的波形——不论多么复杂——它们都可以表示为许多频率是基础频率整数倍的正弦波形。这个基础频率，也称作第一个谐波，是波形周期的频率。第一个泛音，也称作二级谐波，是基本频率的两倍；第二个泛音，或者三级谐波的频率是基本频率的三倍，依次类推。谐波振幅的相互关系形成了波形的形状。

例如，方波可以表示为许多的正弦波，其中偶数谐波（即 2、4、6 等等）的振幅都是 0，而奇数谐波（即 1、3、5 等等）的振幅都按 1、1/3、1/5 比例依次类推。在锯齿波中，所有的泛音都出现，而振幅都按 1、1/2、1/3、1/4 比例依此类推。

對於德国科学家 Hermann Helmholtz (1821-1894)，这是了解音质的关键。在他的名著《On the Sensations of Tone》（1885 年，1954 年由 Dover Press 再版）中，Helmholtz 假定耳朵和大脑将复杂的声音分解为正弦波，而这些正弦波相关的强度就是我们所感受的音质。不幸的是，事情还没有这么简单。

随著 1968 年 Wendy Carlos 的唱片《Switched on Bach》的发布，电子音乐合成器引起了公众的广泛注意。那时使用的合成器（例如 Moog）是类比合成器。这些合成器使用类比电路来产生各种声音波形，例如方波、三角波形和锯

齿波形。要使这些波形听起来更像真实的乐器，它们取决于单个音符的变化程序。波形的所有振幅以「包络 (envelope)」形成。当音符开始时，振幅由 0 开始增加，通常增加非常快。这就是所谓的起奏。然后当音符持续时，振幅保持为常数，这时称为持续。音符结束时，振幅降为 0，这时称为释放。

波形通过滤波器，滤波器将削弱一些谐波，并将简单波形转换得更复杂、更有乐感。这些滤波器的切断频率由包络控制，以便声音的谐波内容在音符的程序中改变。

因为这些合成器以丰富的波形格式调和开始，而且一些谐波通过滤波器进行了削弱，这种形式的合成称为「负合成」。

即使在负合成期间，许多人也还会在电子音乐中发现叠加合成是下一个大问题。

在叠加合成中，您可以从许多整数倍正弦波生成器开始，选择整数倍以便每个正弦波都对应一个谐波。每个谐波的振幅都由一个包络单独控制。使用类比电路的叠加合成不实用，因为对单个音符就需要 8 和 24 之间数目的正弦波生成器，而与这些正弦波生成器相关的频率必须精确的互相对齐。类比波形生成器稳定性很差，而且容易发生频率漂移。

不过，由数位合成器（可以数位化地使用对照表产生波形）和电脑产生的波形，频率漂移并不是个问题，因而叠加合成也就切实可行了。因此总的来说：在录制真实的乐曲时，可以用 Fourier 分解法将其分解成多个谐波。然后就可以确定每个谐波的相对强度，再用多个正弦波数位化地产生声音。

如果开始实验时用 Fourier 分析法分析实际的音调，并从多个正弦波来产生这些音调，那么人们将发现音质并不像 Helmholtz 所认为的那样简单。

最大的问题是真实音调的谐波之间并没有精确的整数关系。事实上，「谐波」一词对于实际的音调来说并不十分适当。各种正弦波组成都不和谐，或者更准确地说是「泛音」。

人们发现，实际音调泛音之间的不和谐在创造「真实的」声音时很重要。静态和谐会产生「电流」声。每个泛音都在单个音符上改变振幅和频率。泛音中，相对频率和振幅的关系对于不同的泛音以及来自相同乐器的不同强度是不同的。实际音调中最复杂的部分发生在音符的起奏部分，这时比较不和谐。人们发现音符的这个复杂的起奏位置对于人类感受音质很重要。

简而言之，实际乐器的声音比任何想像的都更复杂。分析音调的观点，以及后面用于控制泛音的振幅和频率的相对简单的包络观点显然都不实用。

实际乐曲的一些分析法发表于早期 (1977 到 1978 年间) 的《Computer Music Journal》（当时由 People's Computer Company 发行，现在由 MIT Press 发行）

由 James A. Moorer、John Grey 和 John Strawn Some 编写了第三部分丛书《Lexicon of Analyzed Tones》，该书显示了在小提琴、双簧管、单簧管和喇叭上演奏一个音符（小于半秒种）的泛音的振幅和频率图形。所用的音符是中音 C 上的降 E。小提琴用 20 个泛音，双簧管和单簧管用 21 个，而喇叭用 12 个。实际上，《Computer Music Journal》的 Volume II、Number 2（1978 年 9 月）包含了用线段来近似双簧管、单簧管和喇叭的不同频率和振幅的包络。

因此，利用 Windows 上支援的声音波形功能，下面的程序很简单：将这些数字键入程式、为每个泛音都产生多个正弦波样本、添加这些样本并将其发送给波形声音音效卡，因此把 20 年前原始录制的声音重新制造出来也很容易。ADDSYNTH（「叠加合成」）如程式 22-6 所示。

程式 22-6 ADDSYNTH

```
ADDSYNTH.C
/*-----
    ADDSYNTH.C -- Additive Synthesis Sound Generation
                                     (c) Charles Petzold, 1998
-----*/
/

#include <windows.h>
#include <math.h>
#include "addsynth.h"
#include "resource.h"

#define ID_TIMER 1
#define SAMPLE_RATE 22050
#define MAX_PARTIALS 21
#define PI 3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("AddSynth") ;
// Sine wave generator
// -----

double SineGenerator (double dFreq, double * pdAngle)
{
    double dAmp ;
    dAmp = sin (* pdAngle) ;
    * pdAngle += 2 * PI * dFreq / SAMPLE_RATE ;

    if (* pdAngle >= 2 * PI)
        * pdAngle -= 2 * PI ;

    return dAmp ;
}
```

```

// Fill a buffer with composite waveform
// -----

VOID FillBuffer (INS ins, PBYTE pBuffer, int iNumSamples)
{
    static double          dAngle [MAX_PARTIALS] ;
    double                dAmp, dFrq, dComp, dFrac ;
    int                    i, iPrt, iMsecTime, iCompMaxAmp, iMaxAmp, iSmp ;
                          // Calculate the composite maximum amplitude

    iCompMaxAmp = 0 ;
    for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
    {
        iMaxAmp = 0 ;
        for (i = 0 ; i < ins.ppprt[iPrt].iNumAmp ; i++)
            iMaxAmp=max(iMaxAmp, ins.ppprt[iPrt].pEnvAmp[i].iValue) ;
        iCompMaxAmp += iMaxAmp ;
    }

    // Loop through each sample
    for (iSmp = 0 ; iSmp < iNumSamples ; iSmp++)
    {
        dComp = 0 ;
        iMsecTime = (int) (1000 * iSmp / SAMPLE_RATE) ;

        // Loop through each partial
        for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
        {
            dAmp = 0 ;
            dFrq = 0 ;

            for (i = 0 ; i < ins.ppprt[iPrt].iNumAmp - 1 ; i++)
            {
                if (iMsecTime >= ins.ppprt[iPrt].pEnvAmp[i ].iTime &&
                    iMsecTime <= ins.ppprt[iPrt].pEnvAmp[i+1].iTime)
                {
                    dFrac = (double) (iMsecTime -
                        ins.ppprt[iPrt].pEnvAmp[i ].iTime) /
                        (ins.ppprt[iPrt].pEnvAmp[i+1].iTime -
                        ins.ppprt[iPrt].pEnvAmp[i ].iTime) ;

                    dAmp = dFrac * ins.ppprt[iPrt].pEnvAmp[i+1].iValue +
                        (1-dFrac) * ins.ppprt[iPrt].pEnvAmp[i ].iValue ;
                    break ;
                }
            }
        }
    }
}

```

```

        for (i = 0 ; i < ins.pprt[iPrt].iNumFrq - 1 ; i++)
        {
            if (iMsecTime >= ins.pprt[iPrt].pEnvFrq[i ].iTime &&
                iMsecTime <= ins.pprt[iPrt].pEnvFrq[i+1].iTime)
            {
                dFrac = (double) (iMsecTime -ins.pprt[iPrt].pEnvFrq[i ].iTime) /

                (ins.pprt[iPrt].pEnvFrq[i+1].iTime -

                ins.pprt[iPrt].pEnvFrq[i ].iTime) ;
                dFrq = dFrac * ins.pprt[iPrt].pEnvFrq[i+1].iValue + (1-dFrac) *
ins.pprt[iPrt].pEnvFrq[i ].iValue ;
                break ;
            }
        }
        dComp += dAmp * SineGenerator (dFrq, dAngle + iPrt) ;
    }
    pBuffer[iSmp] = (BYTE) (127 + 127 * dComp / iCompMaxAmp) ;
}

}

// Make a waveform file
// -----

BOOL MakeWaveFile (INS ins, TCHAR * szFileName)
{
    DWORD                dwWritten ;
    HANDLE                hFile ;
    int                   iChunkSize, iPcmSize, iNumSamples ;
    PBYTE                 pBuffer ;
    WAVEFORMATEX          waveform ;

    hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;
    if (hFile == NULL)
        return FALSE ;
    iNumSamples = ((long) ins.iMsecTime * SAMPLE_RATE / 1000 + 1) / 2 * 2 ;
    iPcmSize    = sizeof (PCMWAVEFORMAT) ;
    iChunkSize = 12 + iPcmSize + 8 + iNumSamples ;

    if (NULL == (pBuffer = malloc (iNumSamples)))
    {
        CloseHandle (hFile) ;
        return FALSE ;
    }

    FillBuffer (ins, pBuffer, iNumSamples) ;
    waveform.wFormatTag                = WAVE_FORMAT_PCM ;

```

```

    waveform.nChannels          = 1 ;
    waveform.nSamplesPerSec     = SAMPLE_RATE ;
    waveform.nAvgBytesPerSec    = SAMPLE_RATE ;
    waveform.nBlockAlign       = 1 ;
    waveform.wBitsPerSample    = 8 ;
    waveform.cbSize             = 0 ;

    WriteFile (hFile, "RIFF", 4, &dwWritten, NULL) ;
    WriteFile (hFile, &iChunkSize, 4, &dwWritten, NULL) ;
    WriteFile (hFile, "WAVEfmt ", 8, &dwWritten, NULL) ;
    WriteFile (hFile, &iPcmSize, 4, &dwWritten, NULL) ;
    WriteFile (hFile, &waveform, sizeof (WAVEFORMATEX) - 2, &dwWritten,
NULL) ;
    WriteFile (hFile, "data", 4, &dwWritten, NULL) ;
    WriteFile (hFile, &iNumSamples, 4, &dwWritten, NULL) ;
    WriteFile (hFile, pBuffer, iNumSamples, &dwWritten,
NULL) ;

    CloseHandle (hFile) ;
    free (pBuffer) ;

    if ((int) dwWritten != iNumSamples)
    {
        DeleteFile (szFileName) ;
        return FALSE ;
    }
    return TRUE ;
}

void TestAndCreateFile (    HWND hwnd, INS ins, TCHAR * szFileName,
                           int idButton)
{
    TCHAR szMessage [64] ;
    if (-1 != GetFileAttributes (szFileName))
        EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
    else
    {
        if (MakeWaveFile (ins, szFileName))
            EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
        else
        {
            wsprintf (szMessage, TEXT ("Couldnot create %x."), szFileName) ;
            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (hwnd, szMessage, szAppName,
MB_OK | MB_ICONEXCLAMATION) ;
        }
    }
}

```

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
    {
        MessageBox (      NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
    }
    return 0 ;
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static TCHAR * szTrum = TEXT ("Trumpet.wav") ;
    static TCHAR * szOboe = TEXT ("Oboe.wav") ;
    static TCHAR * szClar = TEXT ("Clarinet.wav") ;

    switch (message)
    {
    case WM_INITDIALOG:
        SetTimer (hwnd, ID_TIMER, 1, NULL) ;
        return TRUE ;

    case WM_TIMER:
        KillTimer (hwnd, ID_TIMER) ;
        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        TestAndCreateFile (hwnd, insTrum, szTrum, IDC_TRUMPET) ;
        TestAndCreateFile (hwnd, insOboe, szOboe, IDC_OBOE) ;
        TestAndCreateFile (hwnd, insClar, szClar, IDC_CLARINET) ;

        SetDlgItemText (hwnd, IDC_TEXT, TEXT (" ")) ;
        SetFocus (GetDlgItem (hwnd, IDC_TRUMPET)) ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
        return TRUE ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDC_TRUMPET:
            PlaySound (szTrum, NULL, SND_FILENAME | SND_SYNC) ;
            return TRUE ;
        }
```



```

        case IDC_OBOE:
            PlaySound (szOboe, NULL, SND_FILENAME | SND_SYNC) ;
            return TRUE ;

        case IDC_CLARINET:
            PlaySound (szClar, NULL, SND_FILENAME | SND_SYNC) ;
            return TRUE ;

    }
    break ;

case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
        case SC_CLOSE:
            EndDialog (hwnd, 0) ;
            return TRUE ;

    }
    break ;

}
return FALSE ;
}

```

ADDSYNTH.RC (摘录)

//Microsoft Developer Studio generated resource script.

#include "resource.h"

#include "afxres.h"

////////////////////////////////////
/

// Dialog

ADDSYNTH DIALOG DISCARDABLE 100, 100, 176, 49

STYLE WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU

CAPTION "Additive Synthesis"

FONT 8, "MS Sans Serif"

BEGIN

PUSHBUTTON "Trumpet", IDC_TRUMPET, 8, 8, 48, 16

PUSHBUTTON "Oboe", IDC_OBOE, 64, 8, 48, 16

PUSHBUTTON "Clarinet", IDC_CLARINET, 120, 8, 48, 16

LTEXT "Preparing Data...", IDC_TEXT, 8, 32, 100, 8

END

RESOURCE.H (摘录)

// Microsoft Developer Studio generated include file.

// Used by AddSynth.rc

```

#define IDC_TRUMPET 1000
#define IDC_OBOE 1001
#define IDC_CLARINET 1002
#define IDC_TEXT 1003

```

这里没有给出附加档案 ADDSYNTH.H, 因为它包含几百行令人讨厌的叙述, 您将在本书附上的光碟上找到它。在 ADDSYNTH.H 的开始位置, 我定义了三个结构, 用於储存包络资料。每个振幅和频率分别储存到型态 ENV 的结构阵列中。这些数字对由时间 (毫秒) 和振幅值 (按任意度量单位) 或频率 (以周期/秒为单位) 组成。这些阵列的长度可变, 其变化范围从 6 到 14。假定振幅和频率值之间直接相关。

每种乐器都包括一个泛音集 (喇叭用 12 个, 双簧管和单簧管分别使用 21 个), 这些泛音集储存在型态 PRT 的结构阵列中。PRT 结构储存振幅和频率包络的点数, 以及指向 ENV 阵列的指标。INS 结构包括音调的总时间 (以毫秒为单位)、泛音数以及指向储存泛音的 PRT 阵列的指标。

ADDSYNTH 有三个标记为「Trumpet」、「Oboe」和「Clarinet」的按钮。PC 的速度还没有快到足以即时计算所有的叠加合成, 因此第一次执行 ADDSYNTH 时, 这些按钮将失效, 直到程式计算完样本并建立了 TRUMPET.WAV、OBOE.WAV 和 CLARINET.WAV 音效档案後, 按钮才启动, 而且可以使用 PlaySound 函式播放这三种声音。下次执行时, 程式将检查波形档案是否存在, 而不需重新建立。

ADDSYNTH 中的 FillBuffer 函式完成了大多数工作。FillBuffer 从计算合成最大振幅的总数开始。为此, 它在乐器的泛音中回圈, 以找出每个泛音的最大振幅, 然後将所有的最大振幅加起来。此值後來用於将样本缩放到 8 位元的样本大小。

然後 FillBuffer 计算每个样本的值。每个样本都对应於一段以毫秒为单位的时间, 该时间取决於取样频率 (实际上, 在 22.05 kHz 的取样频率下, 每 22 个样本对应於相同的毫秒时间值)。然後, FillBuffer 在泛音中回圈。對於频率和振幅, 它找出与毫秒时间值对应的包络线段, 并执行线性插补。

频率值与相位角值一起传递给 SineGenerator 函式。本章前面讨论过, 产生数位化的正弦波形需要保持相位角值, 并依据频率值增加。从 SineGenerator 函式传回时, 正弦值将乘以泛音的振幅并累加。样本的所有泛音都加在起来之後, 样本就缩放到位元组大小。

起床号波形声音

WAKEUP, 如程式 22-7 所示, 是原始码档案看起来不是很完整的程式之一。程式视窗看起来像对话方块, 但是没有资源描述档 (我们已经知道如何编写), 并且程式使用一个波形档案, 但在光碟上却没有这样的档案。不过, 程式非常有趣: 它播放的声音很大, 并且非常令人讨厌。WAKEUP 是我的闹钟, 能够唤醒我继续工作。

程式 22-7 WAKEUP

```

WAKEUP.C
/*-----
-
      WAKEUP.C --      Alarm Clock Program
                                  (c) Charles Petzold, 1998
-----*/

#include <windows.h>
#include <commctrl.h>

      // ID values for 3 child windows
#define      ID_TIMEPICK      0
#define      ID_CHECKBOX      1
#define      ID_PUSHBTN      2

      // Timer ID
#define      ID_TIMER      1

      // Number of 100-nanosecond increments (ie FILETIME ticks) in an hour
#define FTTICKSPERHOUR (60 * 60 * (LONGLONG) 100000000)
      // Defines and structure for waveform "file"
#define      SAMPRATE      11025
#define      NUMSAMPS      (3 * SAMPRATE)
#define      HALFSAMPS      (NUMSAMPS / 2)

typedef struct
{
    char      chRiff[4] ;
    DWORD      dwRiffSize ;
    char      chWave[4] ;
    char      chFmt [4] ;
    DWORD      dwFmtSize ;
    PCMWAVEFORMAT pwf ;
    char      chData[4] ;
    DWORD      dwDataSize ;
    BYTE      byData[0] ;
}

WAVEFORM ;

      // The window proc and the subclass proc
LRESULT      CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT      CALLBACK SubProc (HWND, UINT, WPARAM, LPARAM) ;

      // Original window procedure addresses for the subclassed windows
WNDPROC SubbedProc [3] ;

```

```

    // The current child window with the input focus

HWND hwndFocus ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInst,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    static TCHAR      szAppName [] = TEXT ("WakeUp") ;
    HWND              hwnd ;
    MSG               msg ;
    WNDCLASS           wndclass ;

    wndclass.style
        = 0 ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = (HBRUSH) (1 + COLOR_BTNFACE) ;
    wndclass.lpszMenuName
        = NULL ;
    wndclass.lpszClassName
        = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, szAppName,
                           WS_OVERLAPPED | WS_CAPTION |
                           WS_SYSMENU | WS_MINIMIZEBOX,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

```

```

}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static HWND                hwndDTP, hwndCheck, hwndPush ;
    static WAVEFORM            waveform = { "RIFF", NUMSAMPS + 0x24,
"WAVE", "fmt ",
                                sizeof (PCMWAVEFORMAT), 1, 1, SAMPRATE,
                                SAMPRATE, 1, 8, "data", NUMSAMPS } ;
    static WAVEFORM            * pwaveform ;
    FILETIME                    ft ;
    HINSTANCE                    hInstance ;
    INITCOMMONCONTROLSEX        icex ;
    int                          i, cxChar, cyChar ;
    LARGE_INTEGER                li ;
    SYSTEMTIME                    st ;

    switch (message)
    {
    case WM_CREATE:
                                                // Some initialization stuff

        hInstance      =      (HINSTANCE)      GetWindowLong      (hwnd,
GWL_HINSTANCE) ;

        icex.dwSize = sizeof (icex) ;
        icex.dwICC = ICC_DATE_CLASSES ;
        InitCommonControlsEx (&icex) ;

        // Create the waveform file with alternating square waves

        pwaveform = malloc (sizeof (WAVEFORM) + NUMSAMPS) ;
        * pwaveform = waveform ;

        for (i = 0 ; i < HALFSAMPS ; i++)
            if (i % 600 < 300)
                if (i % 16 < 8)
                    pwaveform->byData[i] = 25 ;
                else
                    pwaveform->byData[i] = 230 ;
            else
                if (i % 8 < 4)
                    pwaveform->byData[i] = 25 ;
                else
                    pwaveform->byData[i] = 230 ;
        // Get character size and set a fixed window size.
        cxChar = LOWORD (GetDialogBaseUnits ()) ;

```

```

        cyChar = HIWORD (GetDialogBaseUnits ()) ;

        SetWindowPos (    hwnd, NULL, 0, 0, 42 * cxChar, 10 * cyChar /
3 + 2 *
GetSystemMetrics (SM_CYBORDER) +GetSystemMetrics (SM_CYCAPTION)
,SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE) ;

        // Create the three child windows

hwndDTP = CreateWindow (DATETIMEPICK_CLASS, TEXT (""),
        WS_BORDER | WS_CHILD | WS_VISIBLE | DTS_TIMEFORMAT,
        2 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
        hwnd, (HMENU) ID_TIMEPICK, hInstance, NULL) ;
hwndCheck = CreateWindow (TEXT ("Button"), TEXT ("Set Alarm"),
        WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
        16 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
        hwnd, (HMENU) ID_CHECKBOX, hInstance, NULL) ;

hwndPush = CreateWindow (TEXT ("Button"), TEXT ("Turn Off"),
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON | WS_DISABLED,
        28 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
        hwnd, (HMENU) ID_PUSHBTN, hInstance, NULL) ;

        hwndFocus = hwndDTP ;

        // Subclass the three child windows

        SubbedProc [ID_TIMEPICK] = (WNDPROC)
SetWindowLong (hwndDTP, GWL_WNDPROC, (LONG) SubProc) ;
        SubbedProc [ID_CHECKBOX] = (WNDPROC)
SetWindowLong (hwndCheck, GWL_WNDPROC, (LONG) SubProc);
        SubbedProc [ID_PUSHBTN] = (WNDPROC)
SetWindowLong (hwndPush, GWL_WNDPROC, (LONG) SubProc) ;

        // Set the date and time picker control to the current time
        // plus 9 hours, rounded down to next lowest hour

        GetLocalTime (&st) ;
        SystemTimeToFileTime (&st, &ft) ;
        li = * (LARGE_INTEGER *) &ft ;
        li.QuadPart += 9 * FTTICKSPERHOUR ;
        ft = * (FILETIME *) &li ;
        FileTimeToSystemTime (&ft, &st) ;
        st.wMinute = st.wSecond = st.wMilliseconds = 0 ;
        SendMessage (hwndDTP, DTM_SETSYSTEMTIME, 0, (LPARAM) &st) ;
        return 0 ;

case WM_SETFOCUS:

```

```

        SetFocus (hwndFocus) ;
        return 0 ;

case WM_COMMAND:
    switch (LOWORD (wParam))        // control ID
    {
    case ID_CHECKBOX:

        // When the user checks the "Set Alarm" button, get the
        // time in the date and time control and subtract from
        // it the current PC time.

        if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
        {
            SendMessage (hwndDTP, DTM_GETSYSTEMTIME, 0, (LPARAM) &st) ;
            SystemTimeToFileTime (&st, &ft) ;
            li = * (LARGE_INTEGER *) &ft ;

            GetLocalTime (&st) ;
            SystemTimeToFileTime (&st, &ft) ;
            li.QuadPart -= ((LARGE_INTEGER *) &ft)->QuadPart ;

            // Make sure the time is between 0 and 24 hours!
            // These little adjustments let us completely ignore
            // the date part of the SYSTEMTIME structures.

            while (    li.QuadPart < 0)
                li.QuadPart += 24 * FTTICKSPERHOUR ;

            li.QuadPart %= 24 * FTTICKSPERHOUR ;

            // Set a one-shot timer! (See you in the morning.)

            SetTimer (hwnd, ID_TIMER, (int) (li.QuadPart / 10000), 0) ;
        }
        // If button is being unchecked, kill the timer.

        else
            KillTimer (hwnd, ID_TIMER) ;

        return 0 ;

    // The "Turn Off" button turns off the ringing alarm, and also
    // unchecks the "Set Alarm" button and disables itself.

    case ID_PUSHBTN:
        PlaySound (NULL, NULL, 0) ;
        SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;

```

```

        EnableWindow (hwndDTP, TRUE) ;
        EnableWindow (hwndCheck, TRUE) ;
    EnableWindow (hwndPush, FALSE) ;
    SetFocus (hwndDTP) ;
    return 0 ;
}
return 0 ;

// The WM_NOTIFY message comes from the date and time picker.
// If the user has checked "Set Alarm" and then gone back to
// change the alarm time, there might be a discrepancy between
// the displayed time and the one-shot timer. So the program
// unchecks "Set Alarm" and kills any outstanding timer.

case WM_NOTIFY:
    switch (wParam)                                // control ID
    {
    case ID_TIMEPICK:
        switch (((NMHDR *) lParam)->code) // notification code
        {
            case DTN_DATETIMECHANGE:
                if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
                {
                    KillTimer (hwnd, ID_TIMER) ;
                    SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;
                }
                return 0 ;
            }
        }
        return 0 ;

// The WM_COMMAND message comes from the two buttons.

case WM_TIMER:

// When the timer message comes, kill the timer (because we only
// want a one-shot) and start the annoying alarm noise going.

        KillTimer (hwnd, ID_TIMER) ;
        PlaySound ( (PTSTR) pwaveform, NULL,
                                SND_MEMORY |
SND_LOOP | SND_ASYNC);

// Let the sleepy user turn off the timer by slapping the
// space bar. If the window is minimized, it's restored; then
// it's brought to the forefront; then the pushbutton is enabled
// and given the input focus.

```



```

        EnableWindow (hwndDTP, FALSE) ;
        EnableWindow (hwndCheck, FALSE) ;
        EnableWindow (hwndPush, TRUE) ;

        hwndFocus = hwndPush ;
        ShowWindow (hwnd, SW_RESTORE) ;
        SetForegroundWindow (hwnd) ;
        return 0 ;

// Clean up if the alarm is ringing or the timer is still set.

case WM_DESTROY:
    free (pwaveform) ;

    if (IsWindowEnabled (hwndPush))
        PlaySound (NULL, NULL, 0) ;

    if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
        KillTimer (hwnd, ID_TIMER) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK SubProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    int idNext, id = GetWindowLong (hwnd, GWL_ID) ;
    switch (message)
    {
    case WM_CHAR:
        if (wParam == '\\t')
        {
            idNext = id ;

            do
            idNext = (idNext +
                (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3 ;
            while (!IsWindowEnabled (GetDlgItem (GetParent
(hwnd), idNext)));

            SetFocus (GetDlgItem (GetParent (hwnd), idNext)) ;
            return 0 ;
        }
        break ;
    }
}

```

```

        case WM_SETFOCUS:
            hwndFocus = hwnd ;
            break ;
    }
    return CallWindowProc ( SubbedProc [id], hwnd, message,
wParam, lParam) ;
}

```

WAKEUP 使用的波形只有两个方波，但是变化迅速。实际的波形在 WndProc 的 WM_CREATE 讯息处理期间计算。所有的波形档案都储存在记忆体中。指向这个记忆体块的指标传递给 PlaySound 函式，该函式使用 SND_MEMORY、SND_LOOP 和 SND_ASYNC 参数。

WAKEUP 使用称为「Date-Time Picker」的通用控制项。这个控制项用来让使用者选择指定的日期和时间（WAKEUP 只使用时间挑选功能）。程式可以使用 SYSTEMTIME 结构来获得和设定时间，在获得和设定 PC 自身时钟时也使用该结构。要多方面了解 Date-Time Picker，请试著建立不带有任意 DTS 样式旗标的视窗。

注意 WM_CREATE 讯息结束时的处理方式：程式假定您在睡觉之前执行它，并希望它在 8 小时之后来唤醒您。

现在很明显，可以从 GetLocalTime 函式在 SYSTEMTIME 结构获得目前时间，而且可以「手工」增加时间。但在一般情况下，此计算将涉及检查大於 24 小时的结果时间，这意味著您必须增加天数栏位，然後可能涉及增加月（因此还必须有用於每月天数和闰年检查的逻辑），最後您可能还要增加年。

事实上，推荐的方法（来自/Platform SDK/Windows Base Services/General Library/Time/Time Reference/Time Structures/SYSTEMTIME）是将 SYSTEMTIME 转换为 FILETIME 结构（使用 SystemTimeToFileTime），将 FILETIME 结构强制转换为 LARGE_INTEGER 结构，在大整数上执行计算，再强制转换回 FILETIME 结构，然後转换回 SYSTEMTIME 结构（使用 FileTimeToSystemTime）。

顾名思义，FILETIME 结构用於获得和设定档案最後一次更新的时间。此结构如下：

```

type struct _FILETIME // ft
{
    DWORD dwLowDateTime ;
    DWORD dwHighDateTime ;
}
FILETIME ;

```

这两个栏位一起表示了从 1601 年 1 月 1 日起每隔 1000 亿分之一秒所显示的 64 位元值。

Microsoft C/C++编译器支援 64 位元整数作为 ANSI C 的非标准延伸语法。

资料型态是__int64。您可以对__int64 型态执行所有的常规算术运算，并且有一些执行时期程式库函数也支援它们。Windows 的 WINNT.H 表头档案定义如下：

```
typedef __int64 LONGLONG ;  
typedef unsigned __int64 DWORDLONG ;
```

在 Windows 中，这有时称为「四字组」，或者更普遍地称为「大整数」。也有一个 union 定义如下：

```
typedef union _LARGE_INTEGER  
{  
    struct  
    {  
        DWORD LowPart ;  
        LONG HighPart ;  
    } ;  
    LONGLONG QuadPart ;  
}  
LARGE_INTEGER ;
```

这是 /Platform SDK/Windows Base Services/General Library/Large Integer Operations 中的全部文件。此 union 允许您使用 32 位元或者 64 位元的大整数。

MIDI 和音乐

由电子音乐合成器制造者协会在 19 世纪 80 年代早期开发了「乐器数位化介面」(MIDI: Musical Instrument Digital Interface)。MIDI 是用於将它们中的电子乐器与电脑连结起来的协定，也是电子音乐领域中相当重要的标准。MIDI 规范由 MIDI Manufacturers Association (MMA) 维护，它的网站是 <http://www.midi.org>。

使用 MIDI

MIDI 为透过电缆来传递数位命令定义了传输协定。MIDI 电缆使用 5 针 DIN 接头，但是只使用了三个接头。一个是遮罩，一个是回路，而第三个传输资料。MIDI 协定在每秒 31,250 位元的速度下是单向的。资料的每个位元组都由一个开始位元开始，以一个停止位元结束，用於每秒 3,125 位元组的有效传输速率。

重要的是要了解真实的声音——不论是类比格式还是数位格式——不是经由 MIDI 电缆传输的。通过电缆传输的通常都是简单的命令讯息，长度一般是 1、2 或 3 位元组。

简单的 MIDI 设定可以包括两片 MIDI 相容硬体。一个是本身不发声，但是单独产生 MIDI 讯息的 MIDI 键盘。此键盘有一个有标记有「MIDI Out」的 MIDI

埠。用 MIDI 电缆将这个埠与 MIDI 声音合成器的「MIDI In」埠连结起来。合成器看起来很像前面有几个按钮的小盒子。

按下键盘上的一个键时（假定是中音 C），键盘就将 3 个位元组发送给 MIDI Out 埠。在十六进位中，这些位元组是：

90 3C 40

第一个位元组（90）显示 Note On 讯息。第二个位元组是键号，其中 3C 是中音 C。第三个位元组是敲按键的速度，此速度范围是从 1 到 127。我们恰巧使用了一个对速度不敏感的键盘，因此它发送平均速度值。这个 3 位元组的讯息顺著 MIDI 电缆进入合成器的 Midi In 埠。通过播放中音 C 的音调来回应合成器。

释放键时，键盘会将另一个 3 位元组讯息发送给 MIDI Out 埠：

90 3C 00

这与 Note On 命令相同，但带有 0 速位元组。这个位元组值 0 表示 Note Off 命令，意味著应该关闭音符。合成器通过停止声音来回应。

如果合成器有复调音乐的能力（即，同时播放多个音符的能力），那么您就可以在键盘上演奏和弦。键盘产生多条 Note On 讯息，并且合成器将播放所有的音符。当您释放和弦时，键盘就将多条 Note Off 讯息发送给合成器。

一般来说，这种设定中的键盘称为「MIDI 控制器」，它负责产生 MIDI 讯息来控制合成器。MIDI 控制器看起来不像键盘。MIDI 控制器包括下面几种：看起来像单簧管或萨克斯管的 MID 管乐控制器、MIDI 吉他控制器、MIDI 弦乐控制器和 MIDI 鼓控制器。至少所有这些控制器都产生 3 位元组的 Note On 和 Note Off 讯息。

胜过类似的键盘或传统乐器，控制器也可以是「编曲器」，它是在记忆体中储存 Note On 和 Note Off 讯息顺序，然後再播放的硬体。现在单机编曲器已经比几年前少见多了，因为它们已经被电脑所替代。安装 MIDI 卡的电脑也可以生成 Note On 和 Note Off 讯息来控制合成器。MIDI 编辑软体，允许您在萤幕上作曲，还可以储存来自 MIDI 控制器的 MIDI 讯息，并处理这些讯息，然後将 MIDI 讯息发送给合成器。

合成器有时也称为「声音模组（sound module）」或「音源器（tone generator）」。MIDI 不指定如何真正产生这些声音的方法。合成器可以使用任何一种声音生成技术。

实际上，只有非常简单的 MIDI 控制器（例如管乐控制器）才只有 MIDI Out 电缆埠。通常键盘都有内建合成器，并且有三个 MIDI 电缆埠，分别标记为「MIDI In」、「MIDI Out」和「MIDI Thru」。MIDI In 埠接受 MIDI 讯息，从而播放键盘的内部合成器。MIDI Out 埠将 MIDI 讯息从键盘发送到外部合成器。MIDI Thru

埠是一个输出埠，它复制 MIDI In 埠的输入信号——无论从 MIDI In 埠获得什么都发送给 MIDI Thru 埠 (MIDI Thru 埠不包括从 MIDI Out 埠发送的任何资讯)。

透过电缆连结 MIDI 硬体只有两种方法：将一个硬体上的 MIDI Out 连结到另一个的 MIDI In，或者将 MIDI Thru 与 MIDI In 连结。MIDI Thru 埠允许连结一系列的 MIDI 合成器。

程式更改

合成器能制作哪种声音？是钢琴声、小提琴声、喇叭声还是飞碟声？通常合成器能够生成的各种声音都储存在 ROM 或者其他地方。它们通常称为「声音」、「乐器」或者「音色」。（「音色」一词来自类比合成器的时代，当时通过将音色和弦插入合成器前面的插孔中来设定不同的声音）。

在 MIDI 中，合成器能够生成的各种声音称为「程式」。改变这个程式需要向合成器发送 MIDI Program Change 讯息

C0 pp

其中，pp 的范围是 0 到 127。通常 MIDI 键盘的顶部是一系列有限的按钮，这些按钮将产生 Program Change 讯息。透过按下这些按钮，您可以从键盘控制合成器的声音。这些按钮号通常由 1 开始，而不是由 0 开始，因此程式代号 1 与 Program Change 位元组的 0 对应。

MIDI 规格没有说明程式代号与乐器的对应关系。例如，著名的 Yamaha DX7 合成器上的前三个程式分别称为「Warm Strings」、「Mellow Horn」和「Pick Guitar」。而在 Yamaha TX81Z 音调发生器上，它们是 Grand Piano、Upright Piano 和 Deep Grand。在 Roland MT-32 声音模组上，它们是 Acoustic Piano 1、Acoustic Piano 2 和 Acoustic Piano 3。因此，如果不希望在从键盘制作程式改变时感到吃惊，那么最好了解一下乐器声与您将使用的合成器的程式代号的对应关系。

这對於包含 Program Change 讯息的 MIDI 档案来说是一个实际问题——这些档案并不是装置无关的，因为它们的内容在不同的合成器上听起来是不一样的。然而，在最近几年，「General MIDI」（GM）标准已经把这些程式代号标准化。Windows 支援 General MIDI。如果合成器与 General MIDI 规格不一致，那么程式转换可使它模拟 General MIDI 合成器。

MIDI 通道

迄今为止，我已经讨论了两条 MIDI 讯息，第一条是 Note On:

90 kk vv

其中，kk 是键号 (0 到 127)，vv 是速度 (0 到 127)。0 速度表示 Note Off

命令。第二条是 Program Change:

C0 pp

其中, pp 的范围是从 0 到 127。这些是典型的 MIDI 讯息。第一个位元组称作「状态」位元组。根据位元组的状态,它通常後跟 0、1 或 2 位元组的「资料」(我即将说明的「系统专有」讯息除外)。从资料位元组中分辨出状态位元组很容易:高位总是 1 用於状态位元组,0 用於资料位元组。

然而,我还没有讨论过这两个讯息的普通格式。Note On 讯息的普通格式如下:

9n kk vv

而 Program Change 是:

Cn pp

在这两种情况下, n 表示状态位元组的低四位元,其变化范围是 0 到 15。这就是 MIDI「通道」。通道一般从 1 开始编号,因此,如果 n 为 0,则代表通道 1。

使用 16 个不同通道允许一条 MIDI 电缆传输 16 种不同声音的讯息。通常,您将发现 MIDI 讯息的特殊字串以 Program Change 讯息开始,为所用的不同通道设定声音,而字串的後面是多条 Note On 和 Note Off 命令。再後面可能是其他的 Program Change 命令。但任何时候,每个通道都只与一种声音联系。

让我们作一个简单范例:假定我已经讨论过的键盘控制能够同时产生用於两条不同通道——通道 1 和通道 2——的 MIDI 讯息。透过按下键盘上的按钮将两条 Program Change 讯息发送给合成器:

C0 01

C1 05

现在设定通道 1 用於程式 2,并设定通道 2 用於程式 6(回忆通道代号和程式代号都是基於 1 的,但讯息中的编码是基於 0 的)。现在按下键盘上的键时,就发送两条 Note On 讯息,一条用於一个通道:

90 kk vv

91 kk vv

这就允许您和谐地同时播放两种乐器的声音。

另一种方法是「分开」键盘。低键可以在通道 1 上产生 Note On 讯息,高键可以在通道 2 上产生 Note On 讯息。这就允许您在一个键盘上独立播放两种乐器的声音。

当您考虑 PC 上的 MIDI 编曲软体时,使用 16 个通道将更为有利。每个通道都代表不同的乐器。如果有能够独立播放 16 种不同乐器的合成器,那么您就可

以编写用於 16 个波段的管弦乐曲，而且只使用一条 MIDI 电缆将 MIDI 卡与合成器连结起来。

MIDI 讯息

尽管 Note On 和 Program Change 讯息在任何 MIDI 执行中都是最重要的讯息，但并不是所有的 MIDI 都可以执行。表 22-2 是 MIDI 规格中定义的 MIDI 通道讯息表。我在前面提到过，状态位元组的高位元总是设定著，而状态位元组後面的资料位元组的高位元都等於 0。这意味著状态位元组的范围是 0x80 到 0xFF，而资料位元组的范围是 0 到 0x7F。

表 22-2 MIDI 通道讯息 (n = 通道代号，从 0 到 15)

MIDI 讯息	资料位元组	值
Note Off	8n kk vv	kk = 键号 (0-127) vv = 速度 (0-127)
Note On	9n kk vv	kk = 键号 (0-127) vv = 速度 (1-127, 0 = note off)
Polyphonic After Touch	An kk tt	kk = 键号 (0-127) tt = 按下之後 (0-127)
Control Change	Bn cc xx	cc = 控制器 (0-121) xx = 值 (0-127)
Channel Mode Local Control	Bn 7A xx	xx = 0 (关), 127 (开)
All Notes Off	Bn 7B 00	
Omni Mode Off	Bn 7C 00	
Omni Mode On	Bn 7D 00	
Mono Mode On	Bn 7E cc	cc = 频道数
Poly Mode On	Bn 7F 00	
Program Change	Cn pp	pp = 程式 (0-127)
Channel After Touch	Dn tt	tt = 按下之後 (0-127)
Pitch Wheel Change	En ll hh	ll = 低 7 位元 (0-127) hh = 高 7 位元 (0-127)

虽然没有严格的要求，键号通常还是与西方音乐的传统音符相对应（例如，

对于打击声音，每个键号码可以是不同的打击乐器）。当键号与钢琴类的键盘对应时，键 60（十进位）是中音 C。MIDI 键号在普通的 88 键钢琴范围的基础上向下扩展了 21 个音符，向上扩展了 19 个音符。速度代号是按下某键的速度，在钢琴上它控制声音的响度与和谐特徵。特殊的声音可以依这种方式或其他方式来回应键的速度。

前面展示的例子使用带有 0 速度位元组的 Note On 讯息来表示 Note Off 命令。对于键盘（或者其他控制器）还有一个单独的 Note Off 命令，该命令实作释放键的速度，不过，非常少见。

还有两个「接触後」讯息。接触後是一些键盘的特徵，按下某个键以後，再用力按下键可以在某些方式上改变声音。一个讯息（状态位元组 0xDn）是将接触後应用於通道中目前演奏的所有音符，这是最常见的。状态位元组 0xA_n 表示独立应用每个单独键的接触後。

通常，键盘上都有一些用於进一步控制声音的刻度盘或开关。这些装置称为「控制器」，所有变化都由状态位元组 0xB_n 表示。通过从 0 到 121 的号码确认控制器。0xB_n 状态位元组也用於 Channel Mode 讯息，这些讯息显示了合成器如何在通道中回应同时发生的音符。

一个非常重要的控制器是上下转换音调的轮，它有一个单独的 MIDI 讯息，其状态位元组是 0xE_n。

表 22-2 中所缺少的是状态位元组以从 F0 到 FF 开始的讯息。这些讯息称为系统讯息，因为它们适用於整个 MIDI 系统，而不是部分通道。系统讯息通常用於同步的目的、触发编曲器、重新设定硬体以及获得资讯。

许多 MIDI 控制器连续发送状态位元组 0xFE，该位元组称为 Active Sensing 讯息。这简单地表示了 MIDI 控制器仍依附於系统。

一条重要的系统讯息是以状态位元组 0xF0 开始的「系统专用」讯息。此讯息用於将资料块按厂商与合成器所依靠的格式传递给合成器（例如，用这种方法可以将新的声音定义从电脑传递给合成器）。系统专用讯息只是可以包含多於 2 个资料位元组的唯一讯息。实际上，资料位元组数是变化的，而每个资料位元组的高位都设定为 0。状态位元组 0xF7 表示系统专用讯息的结尾。

系统专用讯息也用於从合成器转储资料（例如，声音定义）。这些资料都是通过 MIDI Out 埠来自合成器。如果要用装置无关的方式对 MIDI 编写程式，则应该尽可能避免使用系统专用讯息。但是它们对于定义新的合成器声音是非常有用的。

MIDI 档案（副档名是 .MDI）是带有定时资讯的 MIDI 资讯集，可以用 MCI 播放 MIDI 档案。不过，我将在本章的後面讨论低阶 midiOut 函式。

MIDI 编曲简介

低阶 MIDI 的 API 包括字首为 midiIn 和 midiOut 的函式，它们分别用於读取来自外部控制器的 MIDI 序列和在内部或外部的合成器上播放音乐。尽管其名称为「低阶」，但使用这些函式时并不需要了解 MIDI 卡上的硬体介面。

要在播放音乐的准备期间打开一个 MIDI 输出设备，可以呼叫 midiOutOpen 函式：

```
error = midiOutOpen (&hMidiOut, wDeviceID, dwCallBack,
                    dwCallBackData, dwFlags) ;
```

如果呼叫成功，则函式传回 0，否则传回错误代码。如果参数设定正确，则常见的一种错误就是 MIDI 设备已被其他程式使用。

该函式的第一个参数是指向 HMIDIOUT 型态变数的指标，它接收後面用於 MIDI 输出函式的 MIDI 输出代号。第二个参数是设备 ID。要使用真实的 MIDI 设备，这个参数范围可以从 0 到小於由 midiOutGetNumDevs 传回的数值。您还可以使用 MIDIMAPPER，它在 MMSYSTEM.H 中定义为-1。大多数情况下，函式的後三个参数设定为 NULL 或 0。

一旦打开一个 MIDI 输出设备并获得了其代号，您就可以向该设备发送 MIDI 讯息。此时可以呼叫：

```
error = midiOutShortMsg (hMidiOut, dwMessage) ;
```

第一个参数是从 midiOutOpen 函式获得的代号。第二个参数是包装在 32 位元 DWORD 中的 1 位元组、2 位元组或者 3 位元组的讯息。我在前面讨论过，MIDI 讯息以状态位元组开始，後面是 0、1 或 2 位元组的资料。在 dwMessage 中，状态位元组是最不重要的，第一个资料位元组次之，第二个资料位元组再次之，最重要的位元组是 0。

例如，要在 MIDI 通道 5 上以 0x7F 的速度演奏中音 C（音符是 0x3C），则需要 3 位元组的 Note On 讯息：

0x95 0x3C 0x7F

midiOutShortMsg 的参数 dwMessage 等於 0x007F3C95。

三个基础的 MIDI 讯息是 Program Change（可为某一特定通道而改变乐器声音）、Note On 和 Note Off。打开一个 MIDI 输出设备後，应该从一条 Program Change 讯息开始，然後发送相同数量的 Note On 和 Note Off 讯息。

当您一直演奏您想演奏的音乐时，您可以重置 MIDI 输出设备以确保关闭所有的音符：

```
midiOutReset (hMidiOut) ;
```

然後关闭设备：

```
midiOutClose (hMidiOut) ;
```

使用低阶的 MIDI 输出 API 时, midiOutOpen、midiOutShortMsg、midiOutReset 和 midiOutClose 是您需要的四个基础函式。

现在让我们演奏一段音乐。BACHTOCC, 如程式 22-8 所示, 演奏了 J. S. Bach 著名的风琴演奏的 D 小调《Toccata and Fugue》中托卡塔部分的第一小节。

程式 22-8 BACHTOCC

```
BACHTOCC.C
/*-----
---
      BACHTOCC.C --          Bach Toccata in D Minor (First Bar)
                                   (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#define ID_TIMER    1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("BachTocc") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS       wndclass ;

    wndclass.style
        = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc
        = WndProc ;
    wndclass.cbClsExtra
        = 0 ;
    wndclass.cbWndExtra
        = 0 ;
    wndclass.hInstance
        = hInstance ;
    wndclass.hIcon
        = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor
        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground
        = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName
        = NULL ;
    wndclass.lpszClassName
        = szAppName ;
    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;
        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Bach Toccata in D Minor (First
Bar)"),
                        WS_OVERLAPPEDWINDOW,
```

```

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

if (!hwnd)
    return 0 ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}
return msg.wParam ;
}

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                        int iData1, int iData2)
{
    DWORD dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
    return midiOutShortMsg (hMidi, dwMessage) ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static struct
    {
        int iDur ;
        int iNote [2] ;
    }
    noteseq [] = { 110, 69, 81, 110, 67, 79, 990, 69, 81, 220, -1, -1,
110, 67, 79, 110, 65, 77, 110, 64, 76, 110, 62, 74,
220, 61, 73, 440, 62, 74, 1980, -1, -1, 110, 57, 69,
110, 55, 67, 990, 57, 69, 220, -1, -1, 220, 52, 64,
220, 53, 65, 220, 49, 61, 440, 50, 62, 1980, -1, -1 } ;

    static HMIDIOUT hMidiOut ;
    static int iIndex ;
    int i ;

    switch (message)
    {
    case WM_CREATE:
        // Open MIDIMAPPER device

        if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))

```

```

        {
            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (    hwnd, TEXT ("Cannot open
MIDI output device!"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return -1 ;
        }

        // Send Program Change messages for "Church Organ"

        MidiOutMessage (hMidiOut, 0xC0, 0, 19, 0) ;
        MidiOutMessage (hMidiOut, 0xC0, 12, 19, 0) ;

        SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
        return 0 ;

case WM_TIMER:

        // Loop for 2-note polyphony

        for (i = 0 ; i < 2 ; i++)
        {
            // Note Off messages for previous note

            if (iIndex != 0 && noteseq[iIndex - 1].iNote[i] != -1)
            {
                MidiOutMessage (hMidiOut, 0x80, 0, noteseq[iIndex - 1].iNote[i], 0) ;

                MidiOutMessage (hMidiOut, 0x80, 12, noteseq[iIndex - 1].iNote[i], 0) ;
            }
            // Note On messages for new note

            if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]) &&
                noteseq[iIndex].iNote[i] != -1)
            {
                MidiOutMessage (hMidiOut, 0x90, 0, noteseq[iIndex].iNote[i], 127) ;

                MidiOutMessage (hMidiOut, 0x90, 12, noteseq[iIndex].iNote[i], 127) ;
            }
        }

        if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]))
        {
            SetTimer (hwnd, ID_TIMER, noteseq[iIndex++].iDur - 1, NULL) ;
        }
        else
        {
            KillTimer (hwnd, ID_TIMER) ;
            DestroyWindow (hwnd) ;
        }
    }
}

```

```

        return 0 ;

case WM_DESTROY:
    midiOutReset (hMidiOut) ;
    midiOutClose (hMidiOut) ;
    PostQuitMessage (0) ;
    return 0 ;
}

return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

图 22-1 显示了 Bach 的 D 小调 Toccata 的第一小节。



图 22-1 Bach 的 D 小调 Toccata and Fugue 的第一小节

在这里要做的就是将音乐转换成一系列的数值——基本键号和定时资讯，其中定时资讯表示发送 Note On（对应於风琴键按下）和 Note Off（释放键）讯息的时间。由於风琴键盘对速度不敏感，所以我们用相同的速度来演奏所有的音符。另外一个简化是忽略断奏（即，在连续的音符之间留下一个很短的停顿，以达到尖硬的效果）和连奏（在连续的音符之间有更圆润的重叠）之间的区别。我们假定一个音符结束後面紧接著下一个音符开始。

如果看得懂乐谱，那么您就会注意到托卡塔曲以两个平行的八度音阶开始。因此 BACHTOCC 建立了一个资料结构 noteseq 来储存一系列的音符持续时间以及两个键号。不幸的是，音乐持续进入第二小节就需要更特殊的方法来储存此资讯。我将四分音符的持续时间定义为 1760 毫秒，也就是说，八分音符（在音符或者休止符上有一个符尾）的持续时间是 880 毫秒，十六分音符（两个符尾）是 440 毫秒，三十二分音符（三个符尾）是 220 毫秒，六十四分音符（四个符尾）是 110 毫秒。

这第一小节中有两个波音——一个在第一个音符处，另一个在小节的中间。

这在乐谱上用带一条短竖线的曲线表示。在结构复杂的乐曲中，波音符号表示此音符实际应演奏为三个音符——标出的音符、比它低一个全音的音符，然後还是标出的音符。前两个音符演奏得要快，第三个音符要持续剩余的时间。例如，第一个音符是带波音的 A，则应演奏为 A、G、A。我将波音的前两个音符定义为六十四分音符，所以每个音符都持续 110 毫秒。

在第一小节还有四个延长符号。乐谱上表示为中间带点的半圆形。延长符号表示该音符在演奏时所持续的时间比标记的时间要长，通常由演奏者决定具体的时间。我对于延长符号延长了 50% 的时间。

可以看到，即使是转换一小段看来简单直接的乐曲，例如 D 小调《Toccata》的开头，也并不是件容易的事！

noteseq 结构阵列包含了这一小节中平行的音符和休止符的三个数位。音符持续时间的後面是用於平行八度音阶的两个 MIDI 键号。例如，第一个音符是 A，持续时间是 110 毫秒。因为中音 C 的 MIDI 键号是 60，所以中音 C 上面的 A 的键号是 69，比 A 高一个八度音阶的键号是 81。因此，noteseq 阵列的前三个数是 110、69 和 81。我用音符值-1 表示休止符。

WM_CREATE 讯息处理期间，BACHTOCC 设定一个 Windows 计时器用於定时 1000 毫秒——表示乐曲从第 1 秒开始演奏——然後用 MIDIMAPPER 设备 ID 呼叫 midiOutOpen。

BACHTOCC 只需要一种乐器（风琴）的声音，所以只需要一个通道。为了简化 MIDI 讯息的发送，BACHTOCC 中还定义了一个小函式 MidiOutMessage。此函式接收 MIDI 输出代号、状态位元组、通道代号和两个位元组资料。其功能是把些数字打包到一条 32 位元的讯息并呼叫 midiOutShortMsg。

在 WM_CREATE 讯息处理程序的後期，BACHTOCC 发送一条 Program Change 讯息来选择「教堂风琴」的声音。在 General MIDI 声音配置中，教堂风琴声音在 Program Change 讯息中用数位位元组 19 表示。实际演奏的音符出现在 WM_TIMER 讯息处理期间。用回圈来处理两个音符的多音。如果前一个音符还在演奏，BACHTOCC 就为该音符发送 Note Off 讯息。然後，如果下一个音符不是休止符，则向通道 0 和 12 发送 Note On 讯息。随後，重置 Windows 计时器，使其与 noteseq 结构中音符的持续时间一致。

音乐演奏完後，BACHTOCC 删除视窗。在 WM_DESTROY 讯息处理期间，程式呼叫 midiOutReset 和 midiOutClose，然後终止程式。

尽管 BACHTOCC 合理地处理和计算声音（即使还不完全像真人演奏风琴），但一般情况下用 Windows 计时器按这种方式来演奏音乐并不管用。问题在於 Windows 计时器是依据 PC 的系统时钟，其解析度不能满足音乐的要求。而且，

Windows 计时器不是同步的。这样，如果其他程式正忙於执行，则获得 WM_TIMER 讯息就会有轻微的延迟。如果程式不能立即处理这些讯息，就会放弃 WM_TIMER 讯息，这时的声音听起来一团糟。

因此，当 BACHTOCC 显示了如何呼叫低阶 MIDI 输出函式时，使用 Windows 计时器显然不适合精确的音乐创作。所以，Windows 还提供了一系列附加的计时器函式，使用低阶的 MIDI 输出函式时可以利用这些函式。这些函式的字首为 time，您可以利用它们将计时器的解析度设定到最小 1 毫秒。我将在本章结尾的 DRUM 程式向您展示使用这些函式的方法。

通过键盘演奏 MIDI 合成器

因为大多数 PC 使用者可能都没有连结在机器上的 MIDI 键盘，所以可以用每个人都有的键盘（上面全部的字母键和资料键）来代替。程式 22-9 所示的程式 KBMIDI 允许您用 PC 键盘来演奏电子音乐合成器——不管是连结在音效卡上的，还是挂接在 MIDI Out 埠的外部合成器。KBMIDI 让您完全控制 MIDI 输出设备（即内部或外部的合成器）、MIDI 通道和乐器声音。除了演奏时的趣味性以外，我还发现此程式对於开发 Windows 如何实作 MIDI 支援很有用。

程式 22-9 KBMIDI

```
KBMIDI.C
/*-----
    KBMIDI.C -- Keyboard MIDI Player
                                     (c) Charles Petzold, 1998
-----*/

#include <windows.h>
// Defines for Menu IDs
// -----

#define IDM_OPEN          0x100
#define IDM_CLOSE         0x101
#define IDM_DEVICE        0x200
#define IDM_CHANNEL       0x300
#define IDM_VOICE         0x400

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
TCHAR          szAppName [] = TEXT ("KBMidi") ;
HMIDIOUT       hMidiOut ;
int             iDevice = MIDIMAPPER, iChannel = 0, iVoice = 0, iVelocity = 64 ;
int             cxCaps, cyChar, xOffset, yOffset ;

// Structures and data for showing families and instruments on menu
```

```

// -----

typedef struct
{
    TCHAR * szInst ;
    int    iVoice ;
}
INSTRUMENT ;
typedef struct
{
    TCHAR * szFam ;
    INSTRUMENT inst [8] ;
}
FAMILY ;
FAMILY fam [16] = {

    TEXT ("Piano"),

        TEXT ("Acoustic Grand Piano"), 0,
        TEXT ("Bright Acoustic Piano"), 1,
        TEXT ("Electric Grand Piano"), 2,
        TEXT ("Honky-tonk Piano"), 3,
        TEXT ("Rhodes Piano"), 4,
        TEXT ("Chorused Piano"), 5,
        TEXT ("Harpsichord"), 6,
        TEXT ("Clavinet"), 7,
    TEXT ("Chromatic Percussion"),
        TEXT ("Celesta"), 8,
        TEXT ("Glockenspiel"), 9,
        TEXT ("Music Box"), 10,
        TEXT ("Vibraphone"), 11,
        TEXT ("Marimba"), 12,
        TEXT ("Xylophone"), 13,
        TEXT ("Tubular Bells"), 14,
        TEXT ("Dulcimer"), 15,
    TEXT ("Organ"),
        TEXT ("Hammond Organ"), 16,
        TEXT ("Percussive Organ"), 17,
        TEXT ("Rock Organ"), 18,
        TEXT ("Church Organ"), 19,
        TEXT ("Reed Organ"), 20,
        TEXT ("Accordion"), 21,
        TEXT ("Harmonica"), 22,
        TEXT ("Tango Accordion"), 23,
    TEXT ("Guitar"),
        TEXT ("Acoustic Guitar (nylon)"), 24,
        TEXT ("Acoustic Guitar (steel)"), 25,
        TEXT ("Electric Guitar (jazz)"), 26,

```



```

TEXT ("Electric Guitar (clean)"), 27,
TEXT ("Electric Guitar (muted)"), 28,
TEXT ("Overdriven Guitar"), 29,
TEXT ("Distortion Guitar"), 30,
TEXT ("Guitar Harmonics"), 31,
TEXT ("Bass"),
TEXT ("Acoustic Bass"), 32,
TEXT ("Electric Bass (finger)"), 33,
TEXT ("Electric Bass (pick)"), 34,
TEXT ("Fretless Bass"), 35,
TEXT ("Slap Bass 1"), 36,
TEXT ("Slap Bass 2"), 37,
TEXT ("Synth Bass 1"), 38,
TEXT ("Synth Bass 2"), 39,
TEXT ("Strings"),
TEXT ("Violin"), 40,
TEXT ("Viola"), 41,
TEXT ("Cello"), 42,
TEXT ("Contrabass"), 43,
TEXT ("Tremolo Strings"), 44,
TEXT ("Pizzicato Strings"), 45,
TEXT ("Orchestral Harp"), 46,
TEXT ("Timpani"), 47,
TEXT ("Ensemble"),
TEXT ("String Ensemble 1"), 48,
TEXT ("String Ensemble 2"), 49,
TEXT ("Synth Strings 1"), 50,
TEXT ("Synth Strings 2"), 51,
TEXT ("Choir Aahs"), 52,
TEXT ("Voice Oohs"), 53,
TEXT ("Synth Voice"), 54,
TEXT ("Orchestra Hit"), 55,
TEXT ("Brass"),
TEXT ("Trumpet"), 56,
TEXT ("Trombone"), 57,
TEXT ("Tuba"), 58,
TEXT ("Muted Trumpet"), 59,
TEXT ("French Horn"), 60,
TEXT ("Brass Section"), 61,
TEXT ("Synth Brass 1"), 62,
TEXT ("Synth Brass 2"), 63,
TEXT ("Reed"),
TEXT ("Soprano Sax"), 64,
TEXT ("Alto Sax"), 65,
TEXT ("Tenor Sax"), 66,
TEXT ("Baritone Sax"), 67,
TEXT ("Oboe"), 68,
TEXT ("English Horn"), 69,

```

```

        TEXT ("Bassoon"),          70,
        TEXT ("Clarinet"),         71,
    TEXT ("Pipe"),
        TEXT ("Piccolo"),          72,
        TEXT ("Flute "),           73,
        TEXT ("Recorder"),         74,
        TEXT ("Pan Flute"),        75,
        TEXT ("Bottle Blow"),      76,
        TEXT ("Shakuhachi"),       77,
        TEXT ("Whistle"),          78,
        TEXT ("Ocarina"),          79,
    TEXT ("Synth Lead"),
        TEXT ("Lead 1 (square)"),   80,
        TEXT ("Lead 2 (sawtooth)"), 81,
        TEXT ("Lead 3 (caliope lead)"), 82,
        TEXT ("Lead 4 (chiff lead)"), 83,
        TEXT ("Lead 5 (charang)"),  84,
        TEXT ("Lead 6 (voice)"),    85,
        TEXT ("Lead 7 (fifths)"),   86,
        TEXT ("Lead 8 (brass + lead)"), 87,
    TEXT ("Synth Pad"),
        TEXT ("Pad 1 (new age)"),    88,
        TEXT ("Pad 2 (warm)"),       89,
        TEXT ("Pad 3 (polysynth)"), 90,
        TEXT ("Pad 4 (choir)"),      91,
        TEXT ("Pad 5 (bowed)"),      92,
        TEXT ("Pad 6 (metallic)"),   93,
        TEXT ("Pad 7 (halo)"),       94,
        TEXT ("Pad 8 (sweep)"),      95,
    TEXT ("Synth Effects"),
        TEXT ("FX 1 (rain)"),        96,
        TEXT ("FX 2 (soundtrack)"), 97,
        TEXT ("FX 3 (crystal)"),     98,
        TEXT ("FX 4 (atmosphere)"),  99,
        TEXT ("FX 5 (brightness)"), 100,
        TEXT ("FX 6 (goblins)"),    101,
        TEXT ("FX 7 (echoes)"),     102,
        TEXT ("FX 8 (sci-fi)"),     103,
    TEXT ("Ethnic"),
        TEXT ("Sitar"),             104,
        TEXT ("Banjo"),             105,
        TEXT ("Shamisen"),          106,
        TEXT ("Koto"),              107,
        TEXT ("Kalimba"),           108,
        TEXT ("Bagpipe"),           109,
        TEXT ("Fiddle"),            110,
        TEXT ("Shanai"),            111,
    TEXT ("Percussive"),

```

```

        TEXT ("Tinkle Bell"),          112,
        TEXT ("Agogo"),                113,
        TEXT ("Steel Drums"),          114,
        TEXT ("Woodblock"),            115,
        TEXT ("Taiko Drum"),            116,
        TEXT ("Melodic Tom"),           117,
        TEXT ("Synth Drum"),            118,
        TEXT ("Reverse Cymbal"),        119,
    TEXT ("Sound Effects"),
        TEXT ("Guitar Fret Noise"),     120,
        TEXT ("Breath Noise"),           121,
        TEXT ("Seashore"),              122,
        TEXT ("Bird Tweet"),            123,
        TEXT ("Telephone Ring"),        124,
        TEXT ("Helicopter"),            125,
        TEXT ("Applause"),              126,
        TEXT ("Gunshot"),               127
} ;

// Data for translating scan codes to octaves and notes
// -----

#define NUMSCANS    (sizeof key / sizeof key[0])
struct
{
    int          iOctave ;
    int          iNote ;
    int          yPos ;
    int          xPos ;
    TCHAR *      szKey ;
}
key [] =
{
    // Scan Char Oct Note
    // ---- ---- --- ----
    -1,  -1,  1,  -1,  NULL, // 0      None
    -1,  -1, -1,  -1,  NULL, // 1      Esc
    -1,  -1,  0,  0,   TEXT (""), //      2      1
    5,   1,  0,  2,   TEXT ("C#"), //      3      2      5      C#
    5,   3,  0,  4,   TEXT ("D#"), //      4      3      5      D#
    -1,  -1,  0,  6,   TEXT (""), //      5      4
    5,   6,  0,  8,   TEXT ("F#"), //      6      5      5      F#
    5,   8,  0, 10,   TEXT ("G#"), //      7      6      5      G#
    5,  10,  0, 12,   TEXT ("A#"), //      8      7      5      A#
    -1,  -1,  0, 14,   TEXT (""), //      9      8
    6,   1,  0, 16,   TEXT ("C#"), //     10      9      6      C#
    6,   3,  0, 18,   TEXT ("D#"), //     11      0      6      D#
    -1,  -1,  0, 20,   TEXT (""), //     12      -

```

6,	6,	0,	22,	TEXT ("F#"),	//	13	=	6	F#	
-1,	-1,	-1,	-1,	NULL, //	14	Back				
-1,	-1,	-1,	-1,	NULL, //	15	Tab				
5,	0,	1,	1,	TEXT ("C"), //	16		q	5	C	
5,	2,	1,	3,	TEXT ("D"), //	17		w	5	D	
	5,	4,	1,	5, TEXT ("E"),	//	18			e	5
E										
	5,	5,	1,	7, TEXT ("F"),	//	19			r	5
F										
	5,	7,	1,	9, TEXT ("G"),	//	20			t	5
G										
	5,	9,	1,	11, TEXT ("A"),	//	21			y	5
A										
	5,	11,	1,	13, TEXT ("B"),	//	22			u	5
B										
	6,	0,	1,	15, TEXT ("C"),	//	23			i	6
C										
	6,	2,	1,	17, TEXT ("D"),	//	24			o	6
D										
	6,	4,	1,	19, TEXT ("E"),	//	25			p	6
E										
	6,	5,	1,	21, TEXT ("F"),	//	26			[6
F										
	6,	7,	1,	23, TEXT ("G"),	//	27]	6
G										
	-1,	-1,	-1,	-1, NULL,	//	28		Ent		
	-1,	-1,	-1,	-1, NULL,	//	29		Ctrl		
	3,	8,	2,	2, TEXT ("G#"),	//	30			a	3
G#										
	3,	10,	2,	4, TEXT ("A#"),	//	31			s	3
A#										
	-1,	-1,	2,	6, TEXT (""),	//	32			d	
	4,	1,	2,	8, TEXT ("C#"),	//	33			f	4
C#										
	4,	3,	2,	10, TEXT ("D#"),	//	34			g	4
D#										
	-1,	-1,	2,	12, TEXT (""),	//	35			h	
	4,	6,	2,	14, TEXT ("F#"),	//	36			j	4
F#										
	4,	8,	2,	16, TEXT ("G#"),	//	37			k	4
G#										
	4,	10,	2,	18, TEXT ("A#"),	//	38			l	4
A#										
	-1,	-1,	2,	20, TEXT (""),	//	39			;	
	5,	1,	2,	22, TEXT ("C#"),	//	40			'	5
C#										
	-1,	-1,	-1,	-1, NULL,	//	41			`	

```

        -1,    -1,    -1,    -1,    NULL,           // 42    Shift
        -1,    -1,    -1,    -1,    NULL,           // 43        \
(not line continuation)
        3,     9,     3,     3,     TEXT ("A"),       // 44        z     3
A
        3,    11,     3,     5,     TEXT ("B"),       // 45        x     3
B
        4,     0,     3,     7,     TEXT ("C"),       // 46        c     4
C
        4,     2,     3,     9,     TEXT ("D"),       // 47        v     4    D
        4,     4,     3,    11,     TEXT ("E"),       // 48        b     4
E
        4,     5,     3,    13,     TEXT ("F"),       // 49        n     4
F
        4,     7,     3,    15,     TEXT ("G"),       // 50        m     4
G
        4,     9,     3,    17,     TEXT ("A"),       // 51        ,     4    A
        4,    11,     3,    19,     TEXT ("B"),       // 52        .     4
B
        5,     0,     3,    21,     TEXT ("C")        // 53        /     5
C
} ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    MSG                msg;
    HWND               hwnd ;
    WNDCLASS           wndclass ;

    wndclass.style      = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc = WndProc ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon      = LoadIcon (NULL,
IDI_APPLICATION) ;
    wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName,
MB_ICONERROR) ;

```

```

        return 0 ;
    }

    hwnd = CreateWindow ( szAppName, TEXT ("Keyboard MIDI Player"),
        WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    if (!hwnd)
        return 0 ;
    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd);

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

// Create the program's menu (called from WndProc, WM_CREATE)
// -----

HMENU CreateTheMenu (int iNumDevs)
{
    TCHAR                szBuffer [32] ;
    HMENU                hMenu, hMenuPopup, hMenuSubPopup ;
    int                  i, iFam, iIns ;
    MIDIOUTCAPS          moc ;

    hMenu = CreateMenu () ;
        // Create "On/Off" popup menu
    hMenuPopup = CreateMenu () ;
    AppendMenu (hMenuPopup, MF_STRING, IDM_OPEN, TEXT ("%Open")) ;
    AppendMenu (hMenuPopup, MF_STRING | MF_CHECKED, IDM_CLOSE,
        TEXT ("%Closed")) ;
    AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
        TEXT ("%Status")) ;

        // Create "Device" popup menu

    hMenuPopup = CreateMenu () ;
        // Put MIDI Mapper on menu if it's installed
    if (!midiOutGetDevCaps (MIDIMAPPER, &moc, sizeof (moc)))
        AppendMenu (hMenuPopup, MF_STRING, IDM_DEVICE + (int)
MIDIMAPPER,

```

```

                                                                    moc.szPname) ;

else
    iDevice = 0 ;
    // Add the rest of the MIDI devices
    for (i = 0 ; i < iNumDevs ; i++)
    {
        midiOutGetDevCaps (i, &moc, sizeof (moc)) ;
        AppendMenu (hMenuPopup, MF_STRING, IDM_DEVICE + i,
moc.szPname) ;
    }

    CheckMenuItem (hMenuPopup, 0, MF_BYPOSITION | MF_CHECKED) ;
    AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
        TEXT ("&Device")) ;
        // Create "Channel" popup menu
    hMenuPopup = CreateMenu () ;
    for (i = 0 ; i < 16 ; i++)
    {
        wsprintf (szBuffer, TEXT ("%d"), i + 1) ;
        AppendMenu (hMenuPopup, MF_STRING | (i ? MF_UNCHECKED :
MF_CHECKED),
            IDM_CHANNEL + i, szBuffer) ;
    }

    AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
        TEXT ("&Channel")) ;
        // Create "Voice" popup menu
    hMenuPopup = CreateMenu () ;
    for (iFam = 0 ; iFam < 16 ; iFam++)
    {
        hMenuSubPopup = CreateMenu () ;
        for (iIns = 0 ; iIns < 8 ; iIns++)
        {
            wsprintf (szBuffer, TEXT ("%d.\t%s"), iIns + 1,

fam[iFam].inst[iIns].szInst) ;
            AppendMenu (hMenuSubPopup,
MF_STRING | (fam[iFam].inst[iIns].iVoice ?
MF_UNCHECKED : MF_CHECKED),
fam[iFam].inst[iIns].iVoice + IDM_VOICE,
szBuffer) ;
        }

        wsprintf (szBuffer, TEXT ("%c.\t%s"), 'A' + iFam,
fam[iFam].szFam) ;
        AppendMenu (hMenuPopup, MF_STRING | MF_POPUP, (UINT) hMenuSubPopup,
szBuffer) ;
    }

```

```
AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
TEXT("&Voice"));
return hMenu ;
}

// Routines for simplifying MIDI output
// -----

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
int iData1,
int iData2)
{
    DWORD dwMessage ;
    dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
    return midiOutShortMsg (hMidi, dwMessage) ;
}

DWORD MidiNoteOff ( HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int
iVel)
{
    return MidiOutMessage (hMidi, 0x080, iChannel, 12 * iOct + iNote, iVel) ;
}

DWORD MidiNoteOn ( HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int
iVel)
{
    return MidiOutMessage ( hMidi, 0x090, iChannel, 12 * iOct + iNote,
iVel) ;
}

DWORD MidiSetPatch (HMIDIOUT hMidi, int iChannel, int iVoice)
{
    return MidiOutMessage (hMidi, 0x0C0, iChannel, iVoice, 0) ;
}

DWORD MidiPitchBend (HMIDIOUT hMidi, int iChannel, int iBend)
{
    return MidiOutMessage (hMidi, 0x0E0, iChannel, iBend & 0x7F, iBend >> 7) ;
}

// Draw a single key on window
// -----

VOID DrawKey (HDC hdc, int iScanCode, BOOL fInvert)
{
    RECT rc ;
    rc.left = 3 * cxCaps * key[iScanCode].xPos / 2 + xOffset ;
    rc.top = 3 * cyChar * key[iScanCode].yPos / 2 + yOffset ;
```



```

    rc.right      = rc.left + 3 * cxCaps ;
    rc.bottom    = rc.top  + 3 * cyChar / 2 ;

    SetTextColor  (hdc, fInvert ? 0x00FFFFFFul : 0x00000000ul) ;
    SetBkColor    (hdc, fInvert ? 0x00000000ul : 0x00FFFFFFul) ;

    FillRect (hdc, &rc, GetStockObject (fInvert ? BLACK_BRUSH : WHITE_BRUSH)) ;
    DrawText (hdc, key[iScanCode].szKey, -1, &rc,
              DT_SINGLELINE | DT_CENTER |
DT_VCENTER) ;
    FrameRect (hdc, &rc, GetStockObject (BLACK_BRUSH)) ;
}

// Process a Key Up or Key Down message
// -----

VOID ProcessKey (HDC hdc, UINT message, LPARAM lParam)
{
    int iScanCode, iOctave, iNote ;
    iScanCode = 0xFF & HIWORD (lParam) ;
    if (iScanCode >= NUMSCANS)          // No scan codes over 53
        return ;

    if ((iOctave = key[iScanCode].iOctave) == -1)          // Non-music
key
        return ;

    if (GetKeyState (VK_SHIFT) < 0)
        iOctave += 0x20000000 & lParam ? 2 : 1 ;
    if (GetKeyState (VK_CONTROL) < 0)
        iOctave -= 0x20000000 & lParam ? 2 : 1 ;
    iNote = key[iScanCode].iNote ;
    if (message == WM_KEYUP)          // For key up
    {
        MidiNoteOff (hMidiOut, iChannel, iOctave, iNote, 0) ; // Note
off
                                DrawKey (hdc, iScanCode, FALSE) ;
                                return ;
    }

    if (0x40000000 & lParam)          // ignore typemantics
        return ;

    MidiNoteOn (hMidiOut, iChannel, iOctave, iNote, iVelocity) ; // Note on
    DrawKey (hdc, iScanCode, TRUE) ;          // Draw the inverted key
}

// Window Procedure

```

```
// -----

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static BOOL bOpened = FALSE ;
    HDC          hdc ;
    HMENU         hMenu ;
    int           i, iNumDevs, iPitchBend, cxClient, cyClient ;
    MIDIOUTCAPS   moc ;
    PAINTSTRUCT    ps ;
    SIZE          size ;
    TCHAR         szBuffer [16] ;

    switch (message)
    {
    case WM_CREATE:

        // Get size of capital letters in system font

        hdc = GetDC (hwnd) ;

        GetTextExtentPoint (hdc, TEXT ("M"), 1, &size) ;
        cxCaps = size.cx ;
        cyChar = size.cy ;

        ReleaseDC (hwnd, hdc) ;

        // Initialize "Volume" scroll bar

        SetScrollRange          (hwnd, SB_HORZ, 1, 127, FALSE) ;
        SetScrollPos             (hwnd, SB_HORZ, iVelocity, TRUE) ;

        // Initialize "Pitch Bend" scroll bar

        SetScrollRange          (hwnd, SB_VERT, 0, 16383, FALSE) ;
        SetScrollPos             (hwnd, SB_VERT, 8192, TRUE) ;

        // Get number of MIDI output devices and set up menu

        if (0 == (iNumDevs = midiOutGetNumDevs ()))
        {
            MessageBeep (MB_ICONSTOP) ;
            MessageBox (    hwnd, TEXT ("No MIDI output devices!"),
                szAppName, MB_OK | MB_ICONSTOP) ;
            return -1 ;
        }
        SetMenu (hwnd, CreateTheMenu (iNumDevs)) ;
        return 0 ;
    }
}
```

```
case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    xOffset = (cxClient - 25 * 3 * cxCaps / 2) / 2 ;
    yOffset = (cyClient - 11 * cyChar) / 2 + 5 * cyChar ;
    return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    // "Open" menu command

    if (LOWORD (wParam) == IDM_OPEN && !bOpened)
    {
        if (midiOutOpen (&hMidiOut, iDevice, 0, 0, 0))
        {
            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (hwnd, TEXT ("Cannot open MIDI device"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
        }
        else
        {
            CheckMenuItem (hMenu, IDM_OPEN, MF_CHECKED) ;
            CheckMenuItem (hMenu, IDM_CLOSE, MF_UNCHECKED) ;

            MidiSetPatch (hMidiOut, iChannel, iVoice) ;
            bOpened = TRUE ;
        }
    }

    // "Close" menu command
    else if (LOWORD (wParam) == IDM_CLOSE && bOpened)
    {
        CheckMenuItem (hMenu, IDM_OPEN, MF_UNCHECKED) ;
        CheckMenuItem (hMenu, IDM_CLOSE, MF_CHECKED) ;

        // Turn all keys off and close device
        for (i = 0 ; i < 16 ; i++)
            MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;
        midiOutClose (hMidiOut) ;
        bOpened = FALSE ;
    }

    // Change MIDI "Device" menu command
    else if ( LOWORD (wParam) >= IDM_DEVICE - 1 &&
        LOWORD (wParam) < IDM_CHANNEL)
```

```

        {
            CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_UNCHECKED) ;
            iDevice = LOWORD (wParam) - IDM_DEVICE ;
            CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_CHECKED) ;

            // Close and reopen MIDI device

            if (bOpened)
            {
                SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
                SendMessage (hwnd, WM_COMMAND, IDM_OPEN, 0L) ;
            }

            // Change MIDI "Channel" menu command

            else if ( LOWORD (wParam) >= IDM_CHANNEL &&
                     LOWORD (wParam) <  IDM_VOICE)
            {
                CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_UNCHECKED);
                iChannel = LOWORD (wParam) - IDM_CHANNEL ;
                CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_CHECKED) ;

                if (bOpened)
                    MidiSetPatch (hMidiOut, iChannel, iVoice) ;
            }

            // Change MIDI "Voice" menu command

            else if (LOWORD (wParam) >= IDM_VOICE)
            {
                CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_UNCHECKED) ;
                iVoice = LOWORD (wParam) - IDM_VOICE ;
                CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_CHECKED) ;

                if (bOpened)
                    MidiSetPatch (hMidiOut, iChannel, iVoice) ;
            }

            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;

            // Process a Key Up or Key Down message

case WM_KEYUP:
case WM_KEYDOWN:
        hdc = GetDC (hwnd) ;

```

```

        if (bOpened)
            ProcessKey (hdc, message, lParam) ;

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

        // For Escape, turn off all notes and repaint

case WM_CHAR:
    if (bOpened && wParam == 27)
    {
        for (i = 0 ; i < 16 ; i++)
            MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;

        InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

    // Horizontal scroll: Velocity

case WM_HSCROLL:
    switch (LOWORD (wParam))
    {
        case SB_LINEUP:            iVelocity -= 1 ; break ;
        case SB_LINEDOWN:          iVelocity += 1 ; break ;
        case SB_PAGEUP:            iVelocity -= 8 ; break ;
        case SB_PAGEDOWN:          iVelocity += 8 ; break ;
        case SB_THUMBPOSITION:      iVelocity = HIWORD (wParam) ; break ;
        default:
            return 0 ;
    }

    iVelocity = max (1, min (iVelocity, 127)) ;
    SetScrollPos (hwnd, SB_HORZ, iVelocity, TRUE) ;
    return 0 ;

    // Vertical scroll: Pitch Bend

case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
        case SB_THUMBTRACK:        iPitchBend = 16383 - HIWORD (wParam) ; break ;
        case SB_THUMBPOSITION:      iPitchBend = 8191 ; break ;
        default:                    return 0 ;
    }

    iPitchBend = max (0, min (iPitchBend, 16383)) ;
    SetScrollPos (hwnd, SB_VERT, 16383 - iPitchBend, TRUE) ;

    if (bOpened)
        MidiPitchBend (hMidiOut, iChannel, iPitchBend) ;

```

```

        return 0 ;

case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    for (i = 0 ; i < NUMSCANS ; i++)
        if (key[i].xPos != -1)
            DrawKey (hdc, i, FALSE) ;

    midiOutGetDevCaps (iDevice, &moc, sizeof (MIDIOUTCAPS)) ;
    wsprintf (szBuffer, TEXT ("Channel %i"), iChannel + 1) ;

    TextOut (  hdc, cxCaps, 1 * cyChar,
Opened ? TEXT ("Open") : TEXT ("Closed"),
bOpened ? 4 : 6) ;
    TextOut (  hdc, cxCaps, 2 * cyChar, moc.szPname,
lstrlen (moc.szPname)) ;
    TextOut    (hdc, cxCaps, 3 * cyChar, szBuffer, lstrlen
(szBuffer)) ;

    TextOut    (hdc, cxCaps, 4 * cyChar,
fam[iVoice / 8].inst[iVoice % 8].szInst,
lstrlen (fam[iVoice / 8].inst[iVoice % 8].szInst)) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

case WM_DESTROY :
    SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

```

执行 KBMIDI 时,视窗显示了键盘上的键与传统钢琴或风琴按键的对应方式。左下角的 Z 键以 110 Hz 的频率演奏 A。键盘的最下行,右边是中音 C,倒数第二行为其升音或降音。上面两行键继续按此规律变化,从中音 C 到 G#。这样,整个范围是三个八度音阶。另外,分别按 Shift 键和 Ctrl 键可使整个音域上升或下降 1 个八度音阶,这样有效的音域就是 5 个八度音阶。

不过,如果立即开始演奏,那么您将听不到任何声音。您必须先从「Status」功能表中选择「Open」,打开一个 MIDI 输出设备。如果埠打开成功,则按下一个键就向合成器发送一条 MIDI Note On 讯息,释放键则产生一条 Note Off 讯息。取决於键盘的按键特性,您可以同时演奏几个音符。

从「Status」功能表里选择「Close」来关闭 MIDI 设备。这對於需要在不终止 KBMIDI 程式的情况下执行 Windows 下的其他 MIDI 软体来说是很方便的。

「Device」功能表列出了已安装的 MIDI 输出设备，这些设备通过呼叫 `midiOutGetDevCaps` 函式获得。其中有些设备可能是 MIDI Out 埠连结的实际存在或不存在的外部合成器。列表还包括 MIDI Mapper 设备。这是从「控制台」的「多媒体」中选择的 MIDI 合成器。

「Channel」功能表用来选择从 1 到 16 的 MIDI 通道，内定状态下选择通道 1。KBMIDI 程式产生的所有 MIDI 讯息都发送到所选的通道。

KBMIDI 最後一个功能表项是「Voice」，它是一个双层功能表，用於选择 128 种乐器声音，这些声音在 General MIDI 规范中定义并在 Windows 中实作。这 128 种乐器声音分为 16 乐器组，每个乐器组有 8 种乐器。由於不同的 MIDI 键号对应於不同的泛音，所以这 128 种乐器声音也称为有旋律的声音。

General MIDI 中还定义了大量无旋律的打击乐器。要演奏打击乐器，可以从「Channel」功能表选择通道 10，还可以从「Voice」功能表选择第一种乐器声音（「Acoustic Grand Piano」）。这样，按不同的键就可以得到不同打击乐器的声音。从 MIDI 键号 35（低於中音 C 两个八度音阶的 B）到 81（高於中音 C 近两个八度音阶的 A），共有 47 种不同的打击乐器声音。在下面的 DRUM 程式中就利用了打击乐器通道。

KBMIDI 程式有水平和垂直卷动列。由於 PC 键盘对按键速度不敏感，所以用水平卷动列来控制音符速度。一般来说，这与演奏音符的音量一致。设定完水平卷动列以後，所有的 Note On 讯息都将使用这个速度。

垂直卷动列将产生一条称为「Pitch Bend」的 MIDI 讯息。要使用此特性，请按下下一个或多个键，然後用滑鼠拖动卷动列。向上拖动卷动列音符频率将上升，向下拖动则频率下降。释放卷动列後将恢复正常的基音。

这两个卷动列要小心使用：因为拖动卷动列时，键盘讯息将不进入程式的讯息回圈。因此，如果按下下一个键後就开始拖动卷动列，然後在完成拖动之前就释放了该键，那么音符仍将发声。所以，拖动卷动列时不要按下或者释放任何键。对功能表也有类似的规则：按著键时不要进行功能表选择。另外，在按下与释放某个键期间，不要用 Ctrl 或 Shift 键来改变八度音阶。

如果一个或者多个音符出现「粘滯现象」，即释放後继续发声，那么请按下 Esc 键。按下此键将通过向 MIDI 合成器的 16 个通道发送 16 条 All Notes Off 讯息，来关闭声音。

KBMIDI 没有资源描述档，而是通过搜索来建立的功能表。设备名称从 `midiOutGetDevCaps` 函式获得，乐器种类和名称则储存在程式的一个大资料结构中。

KBMIDI 定义了几个小函式来简化 MIDI 讯息。除了 Pitch Bend 讯息以外，

其他讯息都在前面讨论过了。Pitch Bend 讯息用两个 7 位元值组成一个 14 位元的音调弯曲等级：0 到 0x1FFF 之间的值降低基音，0x2001 到 0x3FFF 之间的值升高基音。

从「Status」功能表选择「Open」时，KBMIDI 为选择的设备呼叫 midiOutOpen；如呼叫成功，则呼叫 MidiSetPatch 函式。设备改变时，KBMIDI 必须关闭前一个设备，必要时再打开新设备。当改变 MIDI 设备、MIDI 通道、乐器声音时，KBMIDI 也必须呼叫 MidiSetPatch。

KBMIDI 通过处理 WM_KEYUP 讯息和 WM_KEYDOWN 讯息来控制音符的发音。KBMIDI 中用一个阵列把键盘扫描码映射成八度音阶和音符。例如，美国英语键盘上 Z 键的扫描码是 44，阵列将其标记为八度音阶是 3，音符是 9（即 A）。在 KBMIDI 的 MidiNoteOn 函式里，这些组合成了 MIDI 键号 45（即 12 乘以 3 再加上 9）。此资料结构也用於在视窗中画出键——每个键都有特定的水平和垂直位置，以及显示在矩形中的文字字串。

水平卷动列的处理是很直接的：所有需要做的就是储存新的速度级并设定新的卷动列的位置。但是处理垂直卷动列以控制音调弯曲的操作稍有一点特殊，它处理的卷动列命令只有两个：用滑鼠拖动卷动列时发生的 SB_THUMBTRACK，以及释放卷动列时的 SB_THUMBPOSITION。处理 SB_THUMBPOSITION 命令时，KBMIDI 将卷动列位置设定为中间等级，并呼叫 MidiPitchBend，其中参数值是 8192。

MIDI 击鼓器

有些打击乐器，如木琴或定音鼓，是「有旋律的」或「半音阶的」，因为它们可以用不同的音阶演奏乐曲。木琴用木板来对应不同的音阶，定音鼓也可以演奏曲调。这两种乐器及其他的有旋律的打击乐器都可以在 KBMIDI 的「Voice」功能表里选择。

但是，其他许多打击乐器都没有旋律，它们不能调音，而且通常含有太多的噪音，以致不能与某个基音相联系。在「General MIDI」规范中，这些没旋律的打击乐器声在通道 10 有效。不同的键号对应 47 种不同的打击乐器。

DRUM 程式，如程式 22-10 所示，是一个电脑击鼓器。此程式让您用 47 种不同的打击乐器的声音来构造最大到 32 个音符的一个序列，然後在选择的速度和音量下反复演奏这个序列。

程式 22-10 DRUM

```
DRUM.C
/*-----
-
DRUM.C -- MIDI Drum Machine
```


(c) Charles Petzold, 1998

```

*/

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "drumtime.h"
#include "drumfile.h"
#include "resource.h"

LRESULT      CALLBACK WndProc          (HWND, UINT, WPARAM, LPARAM) ;
BOOL         CALLBACK AboutProc        (HWND, UINT, WPARAM, LPARAM) ;

void          DrawRectangle              (HDC, int, int, DWORD *, DWORD *) ;
void          ErrorMessage               (HWND, TCHAR *, TCHAR *) ;
void          DoCaption                  (HWND, TCHAR *) ;
int           AskAboutSave               (HWND, TCHAR *) ;

TCHAR * szPerc [NUM_PERC] =
{
    TEXT ("Acoustic Bass Drum"),          TEXT ("Bass Drum 1"),
    TEXT ("Side Stick"),                  TEXT ("Acoustic Snare"),
    TEXT ("Hand Clap"),                   TEXT ("Electric Snare"),
    TEXT ("Low Floor Tom"),                TEXT ("Closed High Hat"),
    TEXT ("High Floor Tom"),               TEXT ("Pedal High Hat"),
    TEXT ("Low Tom"),                     TEXT ("Open High Hat"),
    TEXT ("Low-Mid Tom"),                  TEXT ("High-Mid Tom"),
    TEXT ("Crash Cymbal 1"),               TEXT ("High Tom"),
    TEXT ("Ride Cymbal 1"),                TEXT ("Chinese Cymbal"),
    TEXT ("Ride Bell"),                   TEXT ("Tambourine"),
    TEXT ("Splash Cymbal"),                TEXT ("Cowbell"),
    TEXT ("Crash Cymbal 2"),               TEXT ("Vibraslap"),
    TEXT ("Ride Cymbal 2"),                TEXT ("High Bongo"),
    TEXT ("Low Bongo"),                   TEXT ("Mute High Conga"),
    TEXT ("Open High Conga"),              TEXT ("Low Conga"),
    TEXT ("High Timbale"),                 TEXT ("Low Timbale"),
    TEXT ("High Agogo"),                   TEXT ("Low Agogo"),
    TEXT ("Cabasa"),                      TEXT ("Maracas"),
    TEXT ("Short Whistle"),                TEXT ("Long Whistle"),
    TEXT ("Short Guiro"),                  TEXT ("Long Guiro"),
    TEXT ("Claves"),                      TEXT ("High Wood Block"),
    TEXT ("Low Wood Block"),               TEXT ("Mute Cuica"),
    TEXT ("Open Cuica"),                  TEXT ("Mute Triangle"),
    TEXT ("Open Triangle")
} ;

```

```

TCHAR          szAppName  []    = TEXT ("Drum") ;
TCHAR          szUntitled []    = TEXT ("(Untitled)") ;
TCHAR          szBuffer [80 + MAX_PATH] ;
HANDLE         hInst ;
int            cxChar, cyChar ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND          hwnd ;
    MSG           msg ;
    WNDCLASS      wndclass ;

    hInst = hInstance ;
    wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (hInstance, szAppName) ;
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground  = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = szAppName ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                      szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (szAppName, NULL,
WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
                      WS_MINIMIZEBOX | WS_HSCROLL | WS_VSCROLL,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, szCmdLine) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

```

```

}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static      BOOL      bNeedSave ;
    static      DRUM      drum ;
    static      HMENU      hMenu ;
    static      int        iTempo = 50, iIndexLast ;
    static      TCHAR      szFileName      [MAX_PATH],      szTitleName
[MAX_PATH] ;
    HDC          hdc ;
    int          i, x, y ;
    PAINTSTRUCT  ps ;
    POINT        point ;
    RECT         rect ;
    TCHAR        *      szError ;

    switch (message)
    {
    case WM_CREATE:

        // Initialize DRUM structure

        drum.iMsecPerBeat = 100 ;
        drum.iVelocity     = 64 ;
        drum.iNumBeats     = 32 ;

        DrumSetParams (&drum) ;

        // Other initialization

        cxChar = LOWORD (GetDialogBaseUnits ()) ;
        cyChar = HIWORD (GetDialogBaseUnits ()) ;

        GetWindowRect (hwnd, &rect) ;
        MoveWindow (hwnd,      rect.left, rect.top,
77 * cxChar, 29 * cyChar, FALSE) ;

        hMenu = GetMenu (hwnd) ;

        // Initialize "Volume" scroll bar

        SetScrollRange      (hwnd, SB_HORZ, 1, 127, FALSE) ;
        SetScrollPos        (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

        // Initialize "Tempo" scroll bar

        SetScrollRange      (hwnd, SB_VERT, 0, 100, FALSE) ;

```

```
        SetScrollPos          (hwnd, SB_VERT, iTempo, TRUE) ;

        DoCaption (hwnd, szTitleName) ;
        return 0 ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
            case IDM_FILE_NEW:
                if ( bNeedSave && IDCANCEL ==
AskAboutSave (hwnd, szTitleName))

                    return 0 ;

                // Clear drum pattern

                for (i = 0 ; i < NUM_PERC ; i++)
                {
                    drum.dwSeqPerc [i] = 0 ;
                    drum.dwSeqPian [i] = 0 ;
                }

                InvalidateRect (hwnd, NULL, FALSE) ;
                DrumSetParams (&drum) ;
                bNeedSave = FALSE ;
                return 0 ;

            case IDM_FILE_OPEN:

                // Save previous file

                if (bNeedSave && IDCANCEL ==
                    AskAboutSave (hwnd, szTitleName))
                    return 0 ;

                // Open the selected file

                if (DrumFileOpenDlg (hwnd, szFileName, szTitleName))
                {
                    szError = DrumFileRead (&drum, szFileName) ;

                    if (szError != NULL)
                    {
                        ErrorMessage (hwnd, szError, szTitleName) ;
                        szTitleName [0] = '\\0' ;
                    }
                    else
                    {
                        // Set new parameters

```

```

        Tempo = (int) (50 *
            (log10 (drum.iMsecPerBeat) - 1)) ;

SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;
SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

DrumSetParams (&drum) ;
InvalidateRect (hwnd, NULL, FALSE) ;
bNeedSave = FALSE ;
}

DoCaption (hwnd, szTitleName) ;
    }
    return 0 ;
case IDM_FILE_SAVE:
case IDM_FILE_SAVE_AS:
        // Save the selected file

if ((LOWORD (wParam) == IDM_FILE_SAVE && szTitleName [0]) ||
    DrumFileSaveDlg (hwnd, szFileName, szTitleName))
    {
        szError = DrumFileWrite (&drum, szFileName) ;

        if (szError != NULL)
        {
            ErrorMessage (hwnd, szError, szTitleName) ;
            szTitleName [0] = '\0' ;
        }
        else
            bNeedSave = FALSE ;

        DoCaption (hwnd, szTitleName) ;
    }
    return 0 ;

case IDM_APP_EXIT:
    SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;
    return 0 ;

case IDM_SEQUENCE_RUNNING:
    // Begin sequence

    if (!DrumBeginSequence (hwnd))
    {
        ErrorMessage (hwnd,
            TEXT ("Could not start MIDI sequence -- ")
            TEXT ("MIDI Mapper device is unavailable!"),
            szTitleName) ;
    }

```

```

        }
        else
        {
            CheckMenuItem(hMenu,
IDM_SEQUENCE_RUNNING, MF_CHECKED) ;
            CheckMenuItem(hMenu,
IDM_SEQUENCE_STOPPED, MF_UNCHECKED) ;
        }
        return 0 ;

    case IDM_SEQUENCE_STOPPED:
        // Finish at end of sequence

        DrumEndSequence (FALSE) ;
        return 0 ;

    case IDM_APP_ABOUT:
        DialogBox (hInst, TEXT ("AboutBox"), hwnd, AboutProc) ;
        return 0 ;
    }
    return 0 ;

case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:
    hdc = GetDC (hwnd) ;

    // Convert mouse coordinates to grid coordinates

    x = LOWORD (lParam) / cxChar - 40 ;
    y = 2 * HIWORD (lParam) / cyChar - 2 ;
    // Set a new number of beats of sequence

    if (x > 0 && x <= 32 && y < 0)
    {
        SetTextColor (hdc, RGB (255, 255, 255)) ;
        TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);
        SetTextColor (hdc, RGB (0, 0, 0)) ;

        if (drum.iNumBeats % 4 == 0)
            TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0,
TEXT ("."), 1) ;

        drum.iNumBeats = x ;

        TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);

        bNeedSave = TRUE ;

```

```

    }

    // Set or reset a percussion instrument beat

    if (x >= 0 && x < 32 && y >= 0 && y < NUM_PERC)
    {
        if (message == WM_LBUTTONDOWN)
            drum.dwSeqPerc[y] ^= (1 << x) ;
        else
            drum.dwSeqPian[y] ^= (1 << x) ;

        DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

        bNeedSave = TRUE ;
    }

    ReleaseDC (hwnd, hdc) ;
    DrumSetParams (&drum) ;
    return 0 ;

case WM_HSCROLL:

    // Change the note velocity

    switch (LOWORD (wParam))
    {
        case SB_LINEUP:
            drum.iVelocity -= 1 ; break ;
        case SB_LINEDOWN: drum.iVelocity += 1 ; break ;
        case SB_PAGEUP:   drum.iVelocity -= 8 ; break ;
        case SB_PAGEDOWN: drum.iVelocity += 8 ; break ;
        case SB_THUMBPOSITION:
            drum.iVelocity = HIWORD (wParam) ;
            break ;

        default:
            return 0 ;
    }

    drum.iVelocity = max (1, min (drum.iVelocity, 127)) ;
    SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;
    DrumSetParams (&drum) ;
    bNeedSave = TRUE ;
    return 0 ;

case WM_VSCROLL:

    // Change the tempo

    switch (LOWORD (wParam))

```

```

        {
        case SB_LINEUP:           iTempo -= 1 ; break ;
        case SB_LINEDOWN:         iTempo += 1 ; break ;
        case SB_PAGEUP:           iTempo -= 10 ; break ;
        case SB_PAGEDOWN:         iTempo += 10 ; break ;
        case SB_THUMBPOSITION:
                iTempo = HIWORD (wParam) ;
                break ;

        default:
                return 0 ;
        }

        iTempo = max (0, min (iTempo, 100)) ;
        SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;

        drum.iMsecPerBeat = (WORD) (10 * pow (100, iTempo / 100.0)) ;

        DrumSetParams (&drum) ;
        bNeedSave = TRUE ;
        return 0 ;

case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        SetTextAlign (hdc, TA_UPDATECP) ;
        SetBkMode (hdc, TRANSPARENT) ;

        // Draw the text strings and horizontal lines
        for (i = 0 ; i < NUM_PERC ; i++)
        {
                MoveToEx (hdc, i & 1 ? 20 * cxChar : cxChar,
                        (2 * i + 3) * cyChar / 4, NULL) ;

                TextOut (hdc, 0, 0, szPerc [i], lstrlen (szPerc [i])) ;

                GetCurrentPositionEx (hdc, &point) ;

                MoveToEx (hdc, point.x + cxChar, point.y + cyChar / 2, NULL) ;
                LineTo (hdc, 39 * cxChar, point.y + cyChar / 2) ;
        }

        SetTextAlign (hdc, 0) ;

        // Draw rectangular grid, repeat mark, and beat marks

        for (x = 0 ; x < 32 ; x++)
        {

```



```

        for (y = 0 ; y < NUM_PERC ; y++)
            DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

        SetTextColor (    hdc, x == drum.iNumBeats - 1 ?
            RGB (0, 0, 0) : RGB (255, 255, 255)) ;

        TextOut (hdc, (41 + x) * cxChar, 0, TEXT (":|"), 2) ;

        SetTextColor (hdc, RGB (0, 0, 0)) ;

        if (x % 4 == 0)
            TextOut (hdc, (40 + x) * cxChar, 0, TEXT ("."), 1) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_USER_NOTIFY:

        // Draw the "bouncing ball"

        hdc = GetDC (hwnd) ;

        SelectObject (hdc, GetStockObject (NULL_PEN)) ;
        SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;

        for (i = 0 ; i < 2 ; i++)
        {
            x = iIndexLast ;
            y = NUM_PERC + 1 ;

            Ellipse (hdc, (x + 40) * cxChar, (2 * y + 3) * cyChar / 4,
                (x + 41) * cxChar, (2 * y + 5) * cyChar / 4);

            iIndexLast = wParam ;
            SelectObject      (hdc,      GetStockObject
(BLACK_BRUSH)) ;
        }

        ReleaseDC (hwnd, hdc) ;
        return 0 ;

case WM_USER_ERROR:
        ErrorMessage (hwnd, TEXT ("Can't set timer event for tempo"),
            szTitleName) ;

        // fall through
case WM_USER_FINISHED:
        DrumEndSequence (TRUE) ;
        CheckMenuItem      (hMenu,      IDM_SEQUENCE_RUNNING,

```

```

MF_UNCHECKED) ;
        CheckMenuItem (hMenu, IDM_SEQUENCE_STOPPED,
MF_CHECKED) ;
        return 0 ;

    case WM_CLOSE:
        if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd,
szTitleName))
            DestroyWindow (hwnd) ;

        return 0 ;

    case WM_QUERYENDSESSION:
        if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd,
szTitleName))
            return 1L ;

        return 0 ;

    case WM_DESTROY:
        DrumEndSequence (TRUE) ;
        PostQuitMessage (0) ;
        return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutProc (    HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
    case WM_INITDIALOG:
        return TRUE ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDOK:
            EndDialog (hDlg, 0) ;
            return TRUE ;
        }
        break ;
    }
    return FALSE ;
}

void DrawRectangle (    HDC hdc, int x, int y, DWORD * dwSeqPerc,

```

```

                                DWORD * dwSeqPian)
{
    int iBrush ;
    if (dwSeqPerc [y] & dwSeqPian [y] & (1L << x))
        iBrush = BLACK_BRUSH ;
    else if (dwSeqPerc [y] & (1L << x))
        iBrush = DKGRAY_BRUSH ;
    else if (dwSeqPian [y] & (1L << x))
        iBrush = LTGRAY_BRUSH ;
    else
        iBrush = WHITE_BRUSH ;
    SelectObject (hdc, GetStockObject (iBrush)) ;
    Rectangle (hdc, (x + 40) * cxChar , (2 * y + 4) * cyChar / 4,
                (x + 41) * cxChar + 1, (2 * y + 6) * cyChar / 4 + 1) ;
}

void ErrorMessage (HWND hwnd, TCHAR * szError, TCHAR * szTitleName)
{
    wsprintf (szBuffer, szError,
                (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
    MessageBeep (MB_ICONEXCLAMATION) ;
    MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

void DoCaption (HWND hwnd, TCHAR * szTitleName)
{
    wsprintf (szBuffer, TEXT ("MIDI Drum Machine - %s"),
                (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
    SetWindowText (hwnd, szBuffer) ;
}

int AskAboutSave (HWND hwnd, TCHAR * szTitleName)
{
    int iReturn ;
    wsprintf (szBuffer, TEXT ("Save current changes in %s?"),
                (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
    iReturn = MessageBox ( hwnd, szBuffer, szAppName,
                MB_YESNOCANCEL | MB_ICONQUESTION) ;

    if (iReturn == IDYES)
        if (!SendMessage (hwnd, WM_COMMAND, IDM_FILE_SAVE, 0))
            iReturn = IDCANCEL ;

    return iReturn ;
}

```

DRUMTIME.H

```

/*-----
    DRUMTIME.H Header File for Time Functions for DRUM Program
-----
*/

#define NUM_PERC                                47
#define WM_USER_NOTIFY                        (WM_USER + 1)
#define WM_USER_FINISHED                    (WM_USER + 2)
#define WM_USER_ERROR                        (WM_USER + 3)

#pragma pack(push, 2)
typedef struct
{
    short iMsecPerBeat ;
    short iVelocity ;
    short iNumBeats ;
    DWORD dwSeqPerc [NUM_PERC] ;
    DWORD dwSeqPian [NUM_PERC] ;
}
DRUM, * PDRUM ;
#pragma pack(pop)
void DrumSetParams                                (PDRUM) ;
BOOL DrumBeginSequence                (HWND) ;
void DrumEndSequence                    (BOOL) ;
DRUMTIME.C
/*-----
---
    DRUMFILE.C --      Timer Routines for DRUM
                                (c) Charles Petzold, 1998
-----
-*/

#include <windows.h>
#include "drumtime.h"

#define minmax(a,x,b) (min (max (x, a), b))
#define TIMER_RES    5
void CALLBACK DrumTimerFunc (UINT, UINT, DWORD, DWORD, DWORD) ;
BOOL                                bSequenceGoing, bEndSequence ;
DRUM                                drum ;
HMIDIOUT                            hMidiOut ;
HWND                                hwndNotify ;
int                                iIndex ;
UINT                                uTimerRes, uTimerID ;

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                                int iData1, int
iData2)

```

```
{
    DWORD dwMessage ;
    dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
    return midiOutShortMsg (hMidi, dwMessage) ;
}

void DrumSetParams (PDRUM pdrum)
{
    CopyMemory (&drum, pdrum, sizeof (DRUM)) ;
}

BOOL DrumBeginSequence (HWND hwnd)
{
    TIMECAPS tc ;
    hwndNotify = hwnd ;                // Save window handle for
notification
    DrumEndSequence (TRUE) ;           // Stop current sequence if running

    // Open the MIDI Mapper output port

    if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))
        return FALSE ;
    // Send Program Change messages for channels 9 and 0
    MidiOutMessage (hMidiOut, 0xC0, 9, 0, 0) ;
    MidiOutMessage (hMidiOut, 0xC0, 0, 0, 0) ;

    // Begin sequence by setting a timer event
    timeGetDevCaps (&tc, sizeof (TIMECAPS)) ;
    uTimerRes = minmax (tc.wPeriodMin, TIMER_RES, tc.wPeriodMax) ;
    timeBeginPeriod (uTimerRes) ;

    uTimerID = timeSetEvent(max ((UINT) uTimerRes, (UINT) drum.iMsecPerBeat),
        uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;

    if (uTimerID == 0)
    {
        timeEndPeriod (uTimerRes) ;
        midiOutClose (hMidiOut) ;
        return FALSE ;
    }

    iIndex = -1 ;
    bEndSequence = FALSE ;
    bSequenceGoing = TRUE ;

    return TRUE ;
}
```

```

void DrumEndSequence (BOOL bRightAway)
{
    if (bRightAway)
    {
        if (bSequenceGoing)
        {
            // stop the timer
            if (uTimerID)
                timeKillEvent (uTimerID) ;
            timeEndPeriod (uTimerRes) ;

            // turn off all notes
            MidiOutMessage (hMidiOut, 0xB0, 9, 123, 0) ;
            MidiOutMessage (hMidiOut, 0xB0, 0, 123, 0) ;
            // close the MIDI port midiOutClose (hMidiOut) ; bSequenceGoing = FALSE ;
        }
    }
    else
        bEndSequence = TRUE ;
}

void CALLBACK DrumTimerFunc (      UINT  uID, UINT  uMsg, DWORD dwUser,
                                DWORD dw1, DWORD dw2)
{
    static DWORD      dwSeqPercLast [NUM_PERC], dwSeqPianLast [NUM_PERC] ;
    int               i ;

    // Note Off messages for channels 9 and 0

    if (iIndex != -1)
    {
        for (i = 0 ; i < NUM_PERC ; i++)
        {
            if (dwSeqPercLast[i] & 1 << iIndex)
                MidiOutMessage (hMidiOut, 0x80, 9, i + 35, 0) ;
            if (dwSeqPianLast[i] & 1 << iIndex)
                MidiOutMessage (hMidiOut, 0x80, 0, i + 35, 0) ;
        }
    }

    // Increment index and notify window to advance bouncing ball
    iIndex = (iIndex + 1) % drum.iNumBeats ;
    PostMessage (hwndNotify, WM_USER_NOTIFY, iIndex, timeGetTime ()) ;

    // Check if ending the sequence
    if (bEndSequence && iIndex == 0)
    {
        PostMessage (hwndNotify, WM_USER_FINISHED, 0, 0L) ;
    }
}

```

```

        return ;
    }

    // Note On messages for channels 9 and 0
    for (i = 0 ; i < NUM_PERC ; i++)
    {
        if (drum.dwSeqPerc[i] & 1 << iIndex)
            MidiOutMessage (hMidiOut, 0x90, 9, i + 35,
drum.iVelocity) ;
        if (drum.dwSeqPian[i] & 1 << iIndex)
            MidiOutMessage (hMidiOut, 0x90, 0, i + 35,
drum.iVelocity) ;

        dwSeqPercLast[i] = drum.dwSeqPerc[i] ;
        dwSeqPianLast[i] = drum.dwSeqPian[i] ;
    }

    // Set a new timer event
    uTimerID = timeSetEvent (max ((int) uTimerRes, drum.iMsecPerBeat),
        uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;
    if (uTimerID == 0)
    {
        PostMessage (hwndNotify, WM_USER_ERROR, 0, 0) ;
    }
}

DRUMFILE.H
/*-----
-
    DRUMFILE.H Header File for File I/O Routines for DRUM
-----
-*/

BOOL            DrumFileOpenDlg    (HWND, TCHAR *, TCHAR *) ;
BOOL            DrumFileSaveDlg    (HWND, TCHAR *, TCHAR *) ;

TCHAR *         DrumFileWrite      (DRUM *, TCHAR *) ;
TCHAR *         DrumFileRead       (DRUM *, TCHAR *) ;

DRUMFILE.C
/*-----
--
    DRUMFILE.C --            File I/O Routines for DRUM
                                (c) Charles Petzold, 1998
-----
-*/
#include <windows.h>
#include <commdlg.h>
#include "drumtime.h"
#include "drumfile.h"

OPENFILENAME ofn = { sizeof (OPENFILENAME) } ;

```

```

TCHAR * szFilter[] = { TEXT ("Drum Files (*.DRM)"),
                        TEXT (*.drm"), TEXT ("") } ;

TCHAR szDrumID      [] = TEXT ("DRUM") ;
TCHAR szListID      []  = TEXT ("LIST") ;
TCHAR szInfoID      []  = TEXT ("INFO") ;
TCHAR szSoftID      []  = TEXT ("ISFT") ;
TCHAR szDateID      []  = TEXT ("ISCD") ;
TCHAR szFmtID       []  = TEXT ("fmt ") ;
TCHAR szDataID      []  = TEXT ("data") ;
char  szSoftware     [] = "DRUM by Charles Petzold, Programming Windows" ;

TCHAR szErrorNoCreate      []  = TEXT ("File %s could not be opened for
writing.");
TCHAR szErrorCannotWrite   []  = TEXT ("File %s could not be
written to. ") ;
TCHAR szErrorNotFound      []  = TEXT ("File %s not found or cannot be
opened.");
TCHAR szErrorNotDrum       []  = TEXT ("File %s is not a standard DRUM
file.");
TCHAR szErrorUnsupported   []  = TEXT ("File %s is not a supported
DRUM file.");
TCHAR szErrorCannotRead    []  = TEXT ("File %s cannot be
read.");

BOOL DrumFileOpenDlg (HWND hwnd, TCHAR * szFileName, TCHAR * szTitleName)
{
    ofn.hwndOwner          = hwnd ;
    ofn.lpstrFilter         = szFilter [0] ;
    ofn.lpstrFile           = szFileName ;
    ofn.nMaxFile            = MAX_PATH ;
    ofn.lpstrFileTitle      = szTitleName ;
    ofn.nMaxFileTitle       = MAX_PATH ;
    ofn.Flags               =
OFN_CREATEPROMPT ;
    ofn.lpstrDefExt         = TEXT ("drm") ;

    return GetOpenFileName (&ofn) ;
}

BOOL DrumFileSaveDlg ( HWND hwnd, TCHAR * szFileName,
                        TCHAR * szTitleName)
{
    ofn.hwndOwner          = hwnd ;
    ofn.lpstrFilter         = szFilter [0] ;
    ofn.lpstrFile           = szFileName ;
    ofn.nMaxFile            = MAX_PATH ;
    ofn.lpstrFileTitle      = szTitleName ;

```



```

        ofn.nMaxFileTitle                = MAX_PATH ;
        ofn.Flags                        =
OFN_OVERWRITEPROMPT ;
        ofn.lpstrDefExt                  = TEXT ("drm") ;

        return GetSaveFileName (&ofn) ;
}

TCHAR * DrumFileWrite (DRUM * pdrum, TCHAR * szFileName)
{
    char                                szDateBuf [16] ;
    HMMIO                                hmmio ;
    int                                  iFormat = 2 ;
    MMCKINFO                            mmckinfo [3] ;
    SYSTEMTIME                          st ;
    WORD                                wError = 0 ;

    memset (mmckinfo, 0, 3 * sizeof (MMCKINFO)) ;
        // Recreate the file for writing
    if ((hmmio = mmioOpen (szFileName, NULL,
        MMIO_CREATE | MMIO_WRITE | MMIO_ALLOCBUF)) == NULL)
        return szErrorNoCreate ;
        // Create a "RIFF" chunk with a "CPDR" type
    mmckinfo[0].fccType = mmioStringToFOURCC (szDrumID, 0) ;
    wError |= mmioCreateChunk (hmmio, &mmckinfo[0], MMIO_CREATERIFF) ;
        // Create "LIST" sub-chunk with an "INFO" type
    mmckinfo[1].fccType = mmioStringToFOURCC (szInfoID, 0) ;
    wError |= mmioCreateChunk (hmmio, &mmckinfo[1], MMIO_CREATELIST) ;
        // Create "ISFT" sub-sub-chunk
    mmckinfo[2].ckid = mmioStringToFOURCC (szSoftID, 0) ;
    wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
    wError |= (mmioWrite (hmmio, szSoftware, sizeof (szSoftware)) !=
        sizeof (szSoftware)) ;
    wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
        // Create a time string
    GetLocalTime (&st) ;
    wsprintfA (szDateBuf, "%04d-%02d-%02d", st.wYear, st.wMonth, st.wDay) ;
        // Create "ISCD" sub-sub-chunk
    mmckinfo[2].ckid = mmioStringToFOURCC (szDateID, 0) ;
    wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
    wError |= (mmioWrite (hmmio, szDateBuf, (strlen (szDateBuf) + 1)) !=
        (int) (strlen
(szDateBuf) + 1)) ;
    wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
    wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;

        // Create "fmt " sub-chunk
    mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;

```

```

wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
wError |= (mmioWrite (hmmio, (PSTR) &iFormat, sizeof (int)) !=
           sizeof (int)) ;
wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
                               // Create the "data" sub-chunk
mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
wError |= (mmioWrite (hmmio, (PSTR) pdrum, sizeof (DRUM)) !=
           sizeof (DRUM)) ;
wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
wError |= mmioAscend (hmmio, &mmckinfo[0], 0) ;

                               // Clean up and return
wError |= mmioClose (hmmio, 0) ;
if (wError)
{
    mmioOpen (szFileName, NULL, MMIO_DELETE) ;
    return szErrorCannotWrite ;
}
return NULL ;
}

TCHAR * DrumFileRead (DRUM * pdrum, TCHAR * szFileName)
{
    DRUM          drum ;
    HMMIO          hmmio ;
    int            i, iFormat ;
    MMCKINFO       mmckinfo [3] ;

    ZeroMemory (mmckinfo, 2 * sizeof (MMCKINFO)) ;

                               // Open the file

    if ((hmmio = mmioOpen (szFileName, NULL, MMIO_READ)) == NULL)
        return szErrorNotFound ;
                               // Locate a "RIFF" chunk with a "DRUM" form-type
    mmckinfo[0].ckid = mmioStringToFOURCC (szDrumID, 0) ;
    if (mmioDescend (hmmio, &mmckinfo[0], NULL, MMIO_FINDRIFF))
    {
        mmioClose (hmmio, 0) ;
        return szErrorNotDrum ;
    }

                               // Locate, read, and verify the "fmt " sub-chunk
    mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;
    if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
    {
        mmioClose (hmmio, 0) ;
    }
}

```

```
        return szErrorNotDrum ;
    }

    if (mmckinfo[1].cksize != sizeof (int))
    {
        mmioClose (hmmio, 0) ;
        return szErrorUnsupported ;
    }

    if (mmioRead (hmmio, (PSTR) &iFormat, sizeof (int)) != sizeof (int))
    {
        mmioClose (hmmio, 0) ;
        return szErrorCannotRead ;
    }

    if (iFormat != 1 && iFormat != 2)
    {
        mmioClose (hmmio, 0) ;
        return szErrorUnsupported ;
    }

    // Go to end of "fmt " sub-chunk
    mmioAscend (hmmio, &mmckinfo[1], 0) ;
    // Locate, read, and verify the "data" sub-chunk
    mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
    if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
    {
        mmioClose (hmmio, 0) ;
        return szErrorNotDrum ;
    }

    if (mmckinfo[1].cksize != sizeof (DRUM))
    {
        mmioClose (hmmio, 0) ;
        return szErrorUnsupported ;
    }

    if (mmioRead (hmmio, (LPSTR) &drum, sizeof (DRUM)) != sizeof (DRUM))
    {
        mmioClose (hmmio, 0) ;
        return szErrorCannotRead ;
    }

    // Close the file
    mmioClose (hmmio, 0) ;
    // Convert format 1 to format 2 and copy the DRUM structure data
    if (iFormat == 1)
    {
```

```

        for (i = 0 ; i < NUM_PERC ; i++)
        {
            drum.dwSeqPerc [i] = drum.dwSeqPian [i] ;
            drum.dwSeqPian [i] = 0 ;
        }
    }

    memcpy (pdrum, &drum, sizeof (DRUM)) ;
    return NULL ;
}

DRUM.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Menu
DRUM MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New",      IDM_FILE_NEW
        MENUITEM "&Open...",   IDM_FILE_OPEN
        MENUITEM "&Save",      IDM_FILE_SAVE
        MENUITEM "Save &As...", IDM_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "E&xit",      IDM_APP_EXIT
    END
    POPUP "&Sequence"
    BEGIN
        MENUITEM "&Running",   IDM_SEQUENCE_RUNNING
        MENUITEM "&Stopped",   IDM_SEQUENCE_STOPPED
        , CHECKED
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About...",  IDM_APP_ABOUT
    END
END

////////////////////////////////////
/
// Icon
DRUM          ICON          DISCARDABLE          "drum.ico"

////////////////////////////////////
/

```

```
// Dialog
ABOUTBOX DIALOG DISCARDABLE 20, 20, 160, 164
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON "OK",IDOK,54,143,50,14
ICON "DRUM",IDC_STATIC,8,8,21,20
CTEXT "DRUM",IDC_STATIC,34,12,90,8
CTEXT "MIDI Drum Machine",IDC_STATIC,7,36,144,8
CONTROL "",IDC_STATIC,"Static",SS_BLACKFRAME,8,88,144,46
LTEXT "Left Button:\t\tDrum sounds",IDC_STATIC,12,92,136,8
LTEXT "Right Button:\t\tPiano sounds",IDC_STATIC,12,102,136,8
LTEXT "Horizontal Scroll:\t\tVelocity",IDC_STATIC,12,112,136,8
LTEXT "Vertical Scroll:\t\tTempo",IDC_STATIC,12,122,136,8
CTEXT "Copyright (c) Charles Petzold, 1998",IDC_STATIC,8,48, 144,8
CTEXT "" "Programming Windows," " 5th Edition",IDC_STATIC,8,60, 144,8
END
```

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
// Used by Drum.rc
```

```
#define IDM_FILE_NEW 40001
#define IDM_FILE_OPEN 40002
#define IDM_FILE_SAVE 40003
#define IDM_FILE_SAVE_AS 40004
#define IDM_APP_EXIT 40005
#define IDM_SEQUENCE_RUNNING 40006
#define IDM_SEQUENCE_STOPPED 40007
#define IDM_APP_ABOUT 40008
```

当第一次执行 DRUM 时，您将看到在视窗中有两列，左边一列按名称列出了 47 种不同的打击乐器。右边的网格是打击乐器的声音与时间的二维阵列。每一个打击器都对应网格中的一列。32 行就是 32 拍。如果能让这 32 拍出现在一个 4/4 拍的小节中（即每小节 4 个四分音符），那么每 1 拍对应一个三十二分音符。

从「Sequence」功能表选择「Running」时，程式将试图打开 MIDI Mapper 设备。如果失败，萤幕将出现一个讯息方块。否则，您将看到一个「跳动的小球」随演奏的节拍在网格底部跳过。

在网格的任何位置单击鼠标左键可以在此拍中演奏打击乐器的声音，这时区域将变成暗灰色。用鼠标右键还可以添加钢琴的拍子，这时区域将会变成亮灰色。如果按下两个键（同时或分别），此区域将变成黑色，而且可以同时听到打击乐器和钢琴的声音。再次单击其中的一个键或双键将关闭该拍中的声音。

网格上部是每 4 拍一个点。这些点使我们不用过多的计算就可以很简易地确定单击的位置。网格的右上角是一个冒号和一条竖线（:|），它们看起来像

传统音乐符号中的反复记号。这个符号表示序列的长度。您可以通过单击鼠标来将反复记号放置於网格内的任意位置。该序列最多（但不包括）只能演奏反复记号以内的拍子。如果要建立华尔兹节奏，则应将反复记号设定为 3 拍的若干倍。

水平卷动列控制 MIDI Note On 讯息中的速率位元组。这虽然能改变一些合成器的音质，但一般会影响音量。程式起初将速率卷动列设定在中间位置。竖直卷动列控制拍子。这是对数刻度，范围从每拍 1 秒（卷动列在底部）到每拍 10 毫秒（卷动列在顶部）。程式最初将拍子设定为每拍 100 毫秒（1/10 秒），这时卷动列在中间。

「File」功能表允许您储存和读取副档名为.DRM 的档案，这是我定义的一种格式。这些档案很小并采用了 RIFF 的档案格式，这是一种所有新的多媒体资料档案推荐使用的格式。「Help」功能表中的「About」选项显示一个对话方块，该对话方块用一段非常简明的摘要来说明鼠标在网格中的用法以及两个卷动列的功能。

最後，「Sequence」功能表中的「Stopped」选项用於目前序列结束後终止乐曲并关闭 MIDI Mapper 设备。

多媒体 time 函式

您可能会注意到 DRUM.C 没有呼叫任何多媒体函式。而所有的实际操作都发生在 DRUMTIME 模组中。

虽然普通的 Windows 计时器使用起来很简单，但它对即时时间应用却有灾难性的影响。就像我们在 BACHTOCC 程式中所看到的一样，演奏音乐就是这样的一种即时时间应用，对此 Windows 计时器是不合适的。为了提供在 PC 上演奏 MIDI 所需要的精确度，多媒体 API 还包括一个高解析度的计时器，此计时器通过 7 个字首是 time 的函式实作。这些函式有一个是多余的，而 DRUMTIME 展示了其余 6 个函式的用途。计时器函式将处理执行在一个单独执行绪中的 callback 函式。系统将按照程式指定的计时器延迟时间来呼叫计时器。

处理多媒体计时器时，可以用毫秒指定两种不同的时间。第一个是延迟时间，第二个称为解析度。您可以认为解析度是容错误差。如果指定一个延迟 100 毫秒，而解析度是 10 毫秒，则计时器的实际延迟范围在 90 到 110 毫秒之间。

使用计时器之前，应获得计时器的设备能力：

```
timeGetDevCaps (&timecaps, uSize) ;
```

第一个参数是 TIMECAPS 型态结构的指标，第二个参数是此结构的大小。TIMECAPS 结构只有两个栏位，wPeriodMin 和 wPeriodMax。这是计时器装置驱动程序所支援的最小和最大的解析度值。如果呼叫 timeGetDevCaps 後再查看这些

值, 会发现 wPeriodMin 是 1 而 wPeriodMax 是 65535, 所以此函式并不是很重要。不过, 得到这些解析度值并用於其他计时器函式呼叫是个好主意。

下一步呼叫

```
timeBeginPeriod (uResolution) ;
```

来指出程式所需要的计时器解析度的最低值。该值应在 TIMECAPS 结构所确定的范围之内。此呼叫允许为可能使用计时器的多个程式提供最好的计时器装置驱动程式。呼叫 timeBeginPeriod 及 timeEndPeriod 必须成对出现, 我将在後面对 timeEndPeriod 作简短的描述。

现在可以真正设定一个计时器事件:

```
idTimer = timeSetEvent ( uDelay, uResolution, CallbackFunc, dwData, uFlag) ;
```

如果发生错误, 从呼叫传回的 idTimer 将是 0。在呼叫的下面, 将从 Windows 里用 uDelay 毫秒来呼叫 CallbackFunc 函式, 其中允许的误差由 uResolution 指定。uResolution 值必须大於或等於传递给 timeBeginPeriod 的解析度。dwData 是程式定义的资料, 後来传递给 CallbackFunc。最後一个参数可以是 TIME_ONESHOT, 也可以是 TIME_PERIODIC。前者用於在 uDelay 毫秒数中获得一次 CallbackFunc 呼叫, 而後者用於每个 uDelay 毫秒都获得一次 CallbackFunc 呼叫。

要在呼叫 CallbackFunc 之前终止只发生一次的计时器事件, 或者暂停周期性的计时器事件, 请呼叫

```
timeKillEvent (idTimer) ;
```

呼叫 CallbackFunc 後不必删除只发生一次的计时器事件。在程式中用完计时器以後, 请呼叫

```
timeEndPeriod (wResolution) ;
```

其中的参数与传递给 timeBeginPeriod 的相同。

另两个函式的字首是 time。函式

```
dwSysTime = timeGetTime () ;
```

传回从 Windows 第一次启动到现在的系统时间, 单位是毫秒。函式

```
timeGetSystemTime (&mmtime, uSize) ;
```

需要一个 MMTIME 结构的指标 (与第一个参数一样), 以及此结构的大小 (与第二个参数一样)。虽然 MMTIME 结构可以在其他环境中用来得到非毫秒格式的系统时间, 但此例中它都传回毫秒时间。所以 timeGetSystemTime 是多余的。

Callback 函式只限於它所能做的 Windows 函式呼叫中。Callback 函式可以呼叫 PostMessage, PostMessage 包含有四个计时器函式 (timeSetEvent、timeKillEvent、timeGetTime 和多余的 timeGetSystemTime)、两个 MIDI 输出函式 (midiOutShortMsg 和 midiOutLongMsg) 以及调试函式 OutputDebugStr。

很明显, 设计多媒体计时器主要是用於 MIDI 序列而很少用於其他方面。当

然，可以使用 `PostMessage` 来通知计时器事件的视窗讯息处理程式，而且视窗讯息处理程式可以做任何它想做的事，只是不能回应计时器 `callback` 自身的准确性。

`Callback` 函式有五个参数，但只使用了其中两个参数：从 `timeSetEvent` 传回的计时器 ID 和最初作为参数传递给 `timeSetEvent` 的 `dwData` 值。

`DRUM.C` 模组呼叫 `DRUMTIME.C` 中的 `DrumSetParams` 函式有很多次——建立 `DRUM` 视窗时、使用者在网格上单击或者移动滚动列时、从磁片上载入 `.DRM` 档案时以及清除网格时。`DrumSetParams` 的唯一的参数是指向 `DRUM` 型态结构的指标，此结构型态在 `DRUMTIME.H` 定义。该结构以毫秒为单位储存拍子时间、速度（通常对应於音量）、序列中的拍数以及用於储存网格（为打击乐器和钢琴声设定）的两套 47 个 32 位元组的整数。这些 32 位元整数中的每一位元都对应序列的一拍。`DRUM.C` 模组将在静态记忆体中维护一个 `DRUM` 型态的结构，并在呼叫 `DrumSetParams` 时向它传递一个指标。`DrumSetParams` 只简单地复制此结构的内容。

要启动序列，`DRUM` 呼叫 `DRUMTIME` 中的 `DrumBeginSequence` 函式。唯一的参数就是视窗代号，其作用是通知。`DrumBeginSequence` 打开 `MIDI Mapper` 输出设备，如果成功，则发送 `Program Change` 讯息来为 `MIDI` 通道 0 和 9 选择乐器声音（这些通道是基於 0 的，所以 9 实际指的是 `MIDI` 通道 10，即打击乐器通道。另一个通道用於钢琴声）。`DrumBeginSequence` 透过呼叫 `timeGetDevCaps` 和 `timeBeginPeriod` 来继续工作。在 `TIMER_RES` 定义的理想计时器解析度通常是 5 毫秒，但我定义了一个称作 `minmax` 的巨集来计算从 `timeGetDevCaps` 传回的限制范围以内的解析度。

下一个呼叫是 `timeSetEvent`，用於确定拍子时间，计算解析度、`callback` 函式 `DrumTimerFunc` 以及 `TIME_ONESHOT` 常数。`DRUMTIME` 用的是只发生一次的计时器，而不是周期性计时器，所以速度可以随序列的执行而动态变化。`timeSetEvent` 呼叫之後，计时器装置驱动程式将在延迟时间结束以後呼叫 `DrumTimerFunc`。

`DrumTimerFunc` 是 `DRUMTIME.C` 中的函式，在 `DRUMTIME.C` 中有许多重要的操作。变数 `iIndex` 储存序列中目前的拍子。`Callback` 从为目前演奏的声音发送 `MIDI Note Off` 讯息开始。`iIndex` 的初始值 -1 以防止第一次启动序列时发生这种情况。

接下来，`iIndex` 递增并将其值连同使用者定义的一个 `WM_USER_NOTIFY` 讯息一起传递给 `DRUM` 中的视窗代号。`wParam` 讯息参数设定为 `iIndex`，以便在 `DRUM.C` 中，`WndProc` 能够移动网格底部的「跳动的小球」。

DrumTimerFunc 将下列事件作为结束：把 Note On 讯息发送给通道 0 和 9 的合成器上，并储存网格值以便下一次可以关闭声音，然後透过呼叫 timeSetEvent 来设定新的只发生一次的计时器事件。

要停止序列，DRUM 呼叫 DrumEndSequence，其中唯一的参数可以设定为 TRUE 或 FALSE。如果是 TRUE，则 DrumEndSequence 按下面的程序立即结束序列：删除所有待决的计时器事件，呼叫 timeEndPeriod，向两个 MIDI 通道发送「all notes off」讯息，然後关闭 MIDI 输出埠。当使用者决定终止程式时，DRUM 用 TRUE 参数呼叫 DrumEndSequence。

然而，当使用者在 DRUM 里的「Sequence」功能表中选择「Stop」时，程式将用 FALSE 作为参数呼叫 DrumEndSequence。这就允许序列在结束之前完成目前的回圈。DrumEndSequence 透过把 bEndSequence 整体变数设定为 NULL 来回应此呼叫。如果 bEndSequence 是 TRUE，并且拍子的索引值设定为 0，则 DrumTimerFunc 把使用者定义的 WM_USER_FINISHED 讯息发送给 WndProc。WndProc 必须通过用 TRUE 作为参数呼叫 DrumEndSequence 来回应该讯息，以便正确地结束计时器和 MIDI 埠的使用。

RIFF 档案 I/O

DRUM 程式也可以储存和检索储存在 DRUM 结构中资讯的档案。这些档案格式都是 RIFF (Resource Interchange File Format: 资源交换档案格式)，即一般建议使用的多媒体档案型态。当然，您可以用标准档案 I/O 函式来读写 RIFF 档案，但更简便的方法是使用字首是 mmio (对「多媒体输入/输出」) 的函式。

检查 WAV 格式时我们发现，RIFF 是标记档案格式，这意味著档案中的资料由不同长度的资料块组成。每个资料块都用一个标记来识别。一个标记就是一个 4 位元组的 ASCII 字串。这与 32 位元整数的标记名称相比要容易些。标记的後面是资料块长度及其资料。因为档案中的资讯不是位於档案开头固定的偏移量而是用标记定义，所以标记档案格式是通用的。这样，可以透过添加附加标记来增强档案格式。在读档案时，程式可以很容易地找到所需要的资料并跳过不需要的或者不理解的标记。

Windows 中的 RIFF 档案由独立的资料块组成。一个资料块可以分为资料块类型、资料块大小以及资料本身。资料块类型是 4 字元的 ASCII 码标记，标记中间不能有空格，但末尾可以有。资料块大小是一个 4 位元组 (32 位元) 的值，用於显示资料块的大小。资料本身必须占用偶数个位元组，必要时可以在结尾补 0。这样，资料块的每个部分都是从档案开头就字组对齐好了的。资料块大小不包括资料块类型和资料块大小所需要的 8 位元组，并且不反映添加的资料。

對於一些资料块类型，资料块大小与特定档案无关，是相同的。在资料块是包含资讯的固定长度的结构时，就是这种情况。其他情况下，资料块大小根据特定档案变化。

有两个特殊型态的资料块分别称为 RIFF 资料块和 LIST 资料块。其中，资料以一个 4 字元 ASCII 形式型态开始，後面是一个或多个子资料块。LIST 资料块与 RIFF 资料块类似，只是资料以 4 字元的 ASCII 列表型态开始。RIFF 资料块用於所有的 RIFF 档案，而 LIST 资料块只在档案内部用来合并相关子资料块。

一个 RIFF 档案就是一个 RIFF 资料块。因此，RIFF 档案以字串「RIFF」和一个表示档案长度减去 8 位元组的 32 位元值开始。（实际上，如果需要补充资料则档案可能会长一个位元组。）

多媒体 API 包括 16 个字首是 mmio 的函式，这些函式是专门为 RIFF 档案设计的。DRUMFILE.C 中已经用到其中几个函式来读写 DRUM 资料档案。

要用 mmio 函式打开档案，则第一步是呼叫 mmioOpen。函式传回一个档案代号。mmioCreateChunk 函式在档案中建立一个资料块，这使用 MMCKINFO 定义的资料块名称和特徵。mmioWrite 函式写入资料块。写完资料块以後，呼叫 mmioAscend。传递给 mmioAscend 的 MMCKINFO 结构必须与前面通过传递给 mmioCreateChunk 来建立资料块的 MMCKINFO 结构相同。通过从目前档案指标中减去结构的 dwDataOffset 栏位来执行 mmioAscend 函式，此档案指标现在位於资料块的结尾，并且此值储存在资料的前面。如果资料块在长度上不是 2 位元组的倍数，则 mmioAscend 函式也填补资料。

RIFF 档案由巢状组织的资料块套叠组成。为使 mmioAscend 正常工作，必须维护多个 MMCKINFO 结构，每个结构与档案中的一个曾级相联系。DRUM 资料档案共有三级。因此，在 DRUMFILE.C 中的 DrumFileWrite 函式中，我为三个 MMCKINFO 结构定义了一个阵列，可以分别标记为 mmckinfo[0]、mmckinfo[1] 和 mmckinfo[2]。在第一次 mmioCreateChunk 呼叫中，mmckinfo[0] 结构与 DRUM 形式型态一起用於建立 RIFF 型态的块。其後是第二次 mmioCreateChunk 呼叫，它用 mmckinfo[1] 与 INFO 列表型态一起建立 LIST 型态的资料块。

第三次 mmioCreateChunk 呼叫用 mmckinfo[2] 建立一个 ISFT 型态的资料块，此资料块用於识别建立资料档案的软体。下面的 mmioWrite 呼叫用於写字串 szSoftware，呼叫 mmioAscend 可用 mmckinfo[2] 来填充此资料块的资料块大小栏位。这是第一个完整的资料块。下一个资料块也在 LIST 资料块内。程式继续用另一个 mmioCreateChunk 来呼叫建立 ISCD (creation data: 建立资料) 资料块，并再次使用 mmckinfo[2]。在 mmioWrite 呼叫来写入资料块以後，使用 mmckinfo[2] 呼叫 mmioAscend 来填充资料块大小。现在写到了此资料块的结尾，

也是 LIST 块的结尾。所以，要填充 LIST 资料块的资料块大小栏位，可再次呼叫 `mmioAscend`，这次使用 `mmckinfo[1]`，它最初用於建立 LIST 资料块。

要建立「fmt」和「data」资料块，`mmioCreateChunk` 使用 `mmckinfo[1]`；`mmioWrite` 呼叫的後面也使用 `mmckinfo[1]` 的 `mmioAscend`。在这一点上，除了 RIFF 资料块本身以外，所有的资料块大小都填好了。这需要多次使用 `mmckinfo[0]` 来呼叫 `mmioAscend`。虽然有多次呼叫，但只呼叫 `mmioClose` 一次。

看起来好像 `mmioAscend` 呼叫改变了目前的档案指标，而且它的确填充了资料块大小，但在函式传回时，在资料块结束（或可能因补充资料而增加 1 位元组）以後，档案指标恢复到以前的位置。从应用的观点来看，所有的档案写入都是按从头到尾的顺序。

`mmioOpen` 呼叫成功後，除了磁碟空间耗尽之外，不会发生其他错误。使用变数 `wError` 从 `mmioCreateChunk`、`mmioWrite`、`mmioAscend` 和 `mmioClose` 呼叫累计错误代码，如果磁碟空间不足则每个呼叫都会失败。如果发生了错误，则 `mmioOpen` 以 `MMIO_DELETE` 常数为参数来删除档案，并传回错误资讯。

读 RIFF 档案与建立 RIFF 档案类似，只不过是呼叫 `mmioRead` 而不是 `mmioWrite`，呼叫 `mmioDescend` 而不是 `mmioCreateChunk`。「下降」(descend) 到一个资料块，是指找到资料块位置，并把档案指标移动到资料块大小之後（或者在 RIFF 或 LIST 资料块类型的形式型态或者列表型态的後面）。从资料块「上升」指的是把档案指标移动到资料块的结尾。`mmioDescend` 和 `mmioAscend` 函式都不能把档案指标移到档案的前一个位置。

DRUM 以前的版本在 1992 年的《PC Magazine》发表。那时，Windows 支援两个不同等级的 MIDI 合成器（称为「基本的」和「扩展的」）。那个程式写的档案有格式识别字 1。本章的 DRUM 程式将格式识别字设定为 2。不过，它可以读取并转换早期的格式。这在 `DrumFileRead` 常式中完成。

第二十三章 领略 Internet

Internet——全世界电脑透过不同协定交换资讯的大型连结体——近几年重新定义了个人计算的几个领域。虽然拨接资讯服务和电子邮件系统在 Internet 流行开来之前就已经存在，但它们通常局限於文字模式，并且根本没有连结而是各自分隔的。例如，每一种资讯服务都需要拨不同的电话号码，用不同的使用者 ID 和密码登录。每一种电子邮件系统仅允许在特定系统的缴款使用者之间发送和接收邮件。

现在，往往只需要拨单一支电话就可以连结整个 Internet，而且可以和有电子邮件位址的人进行全球通信。特别是在 World Wide Web 上，超文字、图形和多媒体（包括声音、音乐和视讯）的使用已经扩展了线上资讯的范围和功能。

如果要提供涵盖 Windows 中所有与 Internet 相关程式设计问题的彻底介绍，可能还需要再加上几本书才够。所以，本章实际上主要集中在如何让小型的 Microsoft Windows 应用程式能够有效地从 Internet 上取得资讯的两个领域。这两个领域分别是 Windows Sockets (Winsock) API 和 Windows Internet (WinInet) API 支援的档案传输协定 (FTP: File Transfer Protocol) 的部分。

Windows Sockets

Socket 是由 University of California 在 Berkeley 分校开发的概念，用於在 UNIX 作业系统上添加网路通讯支援。那里开发的 API 现在称为「Berkeley socket interface」。

Sockets 和 TCP/IP

Socket 通常（但不专用於）与主宰 Internet 通信的传输控制项协定/网际网路协定 (TCP/IP: Transmission Control Protocol/Internet Protocol) 牵连在一起。网际网路协定 (IP: Internet Protocol)，作为 TCP/IP 的组成部分之一，用来将资料打包成「资料封包 (datagram)」，该资料封包包含用於标识资料来源和目的地的表头资讯。而传输控制协定 (TCP: Transmission Control Protocol) 则提供了可靠的传输和检查 IP 资料封包正确性的方法。

在 TCP/IP 下，通讯端点由 IP 位址和埠号定义。IP 位址包括 4 个位元组，用於确定 Internet 上的伺服器。IP 位址通常按「由点连结的四个小於 255 的数字」的格式显示，例如「209.86.105.231」。埠号确定了特定的服务或伺服器提供的服务。其中一些埠号已经标准化，以提供众所周知的服务。

当 Socket 与 TCP/IP 合用时, Socket 就是 TCP/IP 的通讯端点。因此, Socket 指定了 IP 位址和埠号。

网路时间服务

下面给出的范例程式与提供时间协定 (Time Protocol) 的 Internet 伺服器相连结。此程式将获得目前准确的日期和时间, 并用此资讯设定您的 PC 时钟。

在美国, 国家标准和技术协会 (National Institute of Standards and Technology) (以前称为国家标准局 (National Bureau of Standards)) 负责维护准确时间, 该时间与世界各地的机构相联系。准确时间可用於无线电广播、电话号码、电脑拨号电话号码以及 Internet, 关于这些的所有文件都位於网站 <http://www.bldrdoc.gov/timefreq> (网域名称「bldrdoc」指的是 Boulder、Colorado、NIST Time 的位置和 Frequency Division)。

我们只对 NIST Network Time Service 感兴趣, 其详细的文件位於 <http://www.bldrdoc.gov/timefreq/service/nts.htm>。此网页列出了十个提供 NIST 时间服务的伺服器。例如, 第一个名称为 time-a.timefreq.bldrdoc.gov, 其 IP 位址为 132.163.135.130。

(我曾经编写过一个使用非 Internet NIST 电脑拨接服务的程式, 并发表於《PC Magazine》, 您也可以在 Ziff-Davis 的网站 <http://www.zdnet.com/pcmag/pctech/content/16/20/ut1620.001.html> 中找到。此程式对于想学习如何使用 Windows Telephony API 的人很有帮助。)

在 Internet 上有三个不同的时间服务, 每一个都由 Request for Comment (RFC) 描述为 Internet 标准。日期协定 (Daytime Protocol) (RFC-867) 提供了一个 ASCII 字串用於指出准确的日期和时间。该 ASCII 字串的准确格式并不标准, 但人们可以理解其中的含义。时间协定 (RFC-868) 提供了一个 32 位元的数字, 用来表示从 1900 年 1 月 1 日至今的秒数。该时间是 UTC (不考虑字母顺序, 它表示世界时间座标 (Coordinated Universal Time)), 它类似於所谓的格林威治标准时间 (Greenwich Mean Time) 或者 GMT——英国格林威治时间。第三个协定称为网路时间协定 (Network Time Protocol) (RFC-1305), 该协定很复杂。

对于我们的目的, 即包括分析 Socket 和不断更新 PC 时钟, 时间协定 RFC-868 已经够用了。RFC-868 只是一个两页的简短文件, 主要是说用 TCP 获得准确时间的程式应该有如下步骤:

1. 连结到提供此服务的伺服器埠 37。
2. 接收 32 位元的时间。

3. 关闭连结。
4. 现在我们已经知道了编写存取时间服务的 Socket 应用程式的每个细节。

NETTIME 程式

Windows Sockets API, 通常也称为 WinSock, 与 Berkeley Sockets API 相容, 因此, 可以想像 UNIX Socket 程式码可以顺利地拿到 Windows 上使用。Windows 下更进一步的支援由对 Berkeley Socket 扩充的功能提供, 其函式的形式是以 WSA(「WinSock API」)为字首。相关的概述和参考位於/Platform SDK/Networking and Distributed Services/Windows Sockets Version 2。

NETTIME, 如程式 23-1 所示, 展示了使用 WinSock API 的方法。

程式 23-1 NETTIME

```
NETTIME.C
/*-----
--
--      NETTIME.C --      Sets System Clock from Internet Services
--                               (c) Charles Petzold, 1998
--
--*/

#include <windows.h>
#include "resource.h"

#define WM_SOCKET_NOTIFY (WM_USER + 1)
#define ID_TIMER          1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK MainDlg (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK ServerDlg (HWND, UINT, WPARAM, LPARAM) ;

void ChangeSystemTime (HWND hwndEdit, ULONG ulTime) ;
void FormatUpdatedTime (HWND hwndEdit, SYSTEMTIME * pstOld,
                      SYSTEMTIME * pstNew) ;
void EditPrintf (HWND hwndEdit, TCHAR * szFormat, ...) ;
HINSTANCE hInst ;
HWND hwndModeless ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   PSTR szCmdLine, int iCmdShow)
{
    static TCHAR szAppName[] = TEXT ("NetTime") ;
    HWND hwnd ;
    MSG msg ;
```

```
RECT                                rect ;
WNDCLASS                            wndclass ;

hInst = hInstance ;
wndclass.style                      = 0 ;
wndclass.lpfnWndProc                = WndProc ;
wndclass.cbClsExtra                  = 0 ;
wndclass.cbWndExtra                  = 0 ;
wndclass.hInstance                  = hInstance ;
wndclass.hIcon                      = LoadIcon (NULL, IDI_APPLICATION) ;
wndclass.hCursor                    = NULL ;
wndclass.hbrBackground              = NULL ;
wndclass.lpszMenuName                = NULL ;
wndclass.lpszClassName              = szAppName ;

if (!RegisterClass (&wndclass))
{
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                szAppName, MB_ICONERROR) ;
    return 0 ;
}

hwnd = CreateWindow (szAppName, TEXT ("Set System Clock from Internet"),
                    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
                    WS_BORDER | WS_MINIMIZEBOX,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

    // Create the modeless dialog box to go on top of the window
hwndModeless = CreateDialog (hInstance, szAppName, hwnd, MainDlg) ;
    // Size the main parent window to the size of the dialog box.
    // Show both windows.

GetWindowRect (hwndModeless, &rect) ;
AdjustWindowRect (&rect, WS_CAPTION | WS_BORDER, FALSE) ;

SetWindowPos (    hwnd, NULL, 0, 0, rect.right - rect.left,
                  rect.bottom - rect.top, SWP_NOMOVE) ;

ShowWindow (hwndModeless, SW_SHOW) ;
ShowWindow (hwnd, iCmdShow) ;
UpdateWindow (hwnd) ;

// Normal message loop when a modeless dialog box is used.
while (GetMessage (&msg, NULL, 0, 0))
{
    if (hwndModeless == 0 || !IsDialogMessage (hwndModeless, &msg))
```

```

        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
        case WM_SETFOCUS:
            SetFocus (hwndModeless) ;
            return 0 ;

        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK MainDlg (      HWND hwnd, UINT message, WPARAM wParam,
                             LPARAM lParam)
{
    static char          szIPAddr[32] = { "132.163.135.130" } ;
    static HWND          hwndButton, hwndEdit ;
    static SOCKET         sock ;
    static struct         sockaddr_in sa ;
    static TCHAR          szOKLabel[32] ;
    int                  iError, iSize ;
    unsigned long         ulTime ;
    WORD                  wEvent, wError ;
    WSADATA               WSAData ;

    switch (message)
    {
        case WM_INITDIALOG:
            hwndButton = GetDlgItem (hwnd, IDOK) ;
            hwndEdit = GetDlgItem (hwnd, IDC_TEXTOUT) ;
            return TRUE ;

        case WM_COMMAND:
            switch (LOWORD (wParam))
            {
                case IDC_SERVER:

```



```
        DialogBoxParam (hInst, TEXT ("Servers"), hwnd, ServerDlg,
(LPARAM) szIPAddr) ;

        return TRUE ;

        case IDOK:
        // Call "WSAStartup" and display description text

            if (iError = WSAStartup (MAKEDWORD(2,0), &WSAData))
            {
                EditPrintf (hwndEdit, TEXT ("Startup error #i.\r\n"), iError) ;
                return TRUE ;
            }

            EditPrintf (hwndEdit, TEXT ("Started up %hs\r\n"),
                WSAData.szDescription);

        // Call "socket"

            sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;

            if (sock == INVALID_SOCKET)
            {
                EditPrintf (hwndEdit, TEXT ("Socket creation error #i.\r\n"),
WSAGetLastError ()) ;
                WSACleanup () ;
                return TRUE ;
            }

            EditPrintf (hwndEdit, TEXT ("Socket %i created.\r\n"), sock) ;

        // Call "WSAAsyncSelect"

            if (SOCKET_ERROR == WSAAsyncSelect (sock, hwnd, WM_SOCKET_NOTIFY,
FD_CONNECT | FD_READ))
            {
                EditPrintf (        hwndEdit, TEXT ("WSAAsyncSelect error #i.\r\n"),
WSAGetLastError ()) ;
                closesocket (sock) ;
                WSACleanup () ;
                return TRUE ;
            }

        // Call "connect" with IP address and time-server port

            sa.sin_family          = AF_INET ;
            sa.sin_port             = htons (IPPORT_TIMESERVER) ;
            sa.sin_addr.S_un.S_addr = inet_addr (szIPAddr) ;

            connect(sock, (SOCKADDR *) &sa, sizeof (sa)) ;
```

```

// "connect" will return SOCKET_ERROR because even if it
// succeeds, it will require blocking. The following only
// reports unexpected errors.

if (WSAEWOULDBLOCK != (iError = WSAGetLastError ()))
{
    EditPrintf (hwndEdit, TEXT ("Connect error #%i.\r\n"), iError) ;
    closesocket (sock) ;
    WSACleanup () ;
    return TRUE ;
}
EditPrintf (hwndEdit, TEXT ("Connecting to %hs..."), szIPAddr) ;

// The result of the "connect" call will be reported
// through the WM_SOCKET_NOTIFY message.
// Set timer and change the button to "Cancel"

SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
GetWindowText (hwndButton, szOKLabel, sizeof (szOKLabel) /sizeof
(TCHAR)) ;
SetWindowText (hwndButton, TEXT ("Cancel")) ;
SetWindowLong (hwndButton, GWL_ID, IDCANCEL) ;
return TRUE ;

case IDCANCEL:
    closesocket (sock) ;
    sock = 0 ;
    WSACleanup () ;
    SetWindowText (hwndButton, szOKLabel) ;
    SetWindowLong (hwndButton, GWL_ID, IDOK) ;

    KillTimer (hwnd, ID_TIMER) ;
    EditPrintf (hwndEdit, TEXT
("\r\nSocket closed.\r\n")) ;

    return TRUE ;

case IDC_CLOSE:
    if (sock)
        SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;

    DestroyWindow (GetParent (hwnd)) ;
    return TRUE ;
}
return FALSE ;

case WM_TIMER:
    EditPrintf (hwndEdit, TEXT (".")) ;
    return TRUE ;

```

```
case WM_SOCKET_NOTIFY:
    wEvent = WSAGETSELECTEVENT (lParam) ;    // ie, LOWORD
    wError = WSAGETSELECTERROR (lParam) ;    // ie, HIWORD

    // Process two events specified in WSAAsyncSelect

    switch (wEvent)
    {
// This event occurs as a result of the "connect" call

        case FD_CONNECT:
            EditPrintf (hwndEdit, TEXT ("\r\n")) ;

            if (wError)
            {
                EditPrintf (    hwndEdit, TEXT ("Connect error #%i."), wError) ;
                SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
                return TRUE ;
            }
            EditPrintf (hwndEdit, TEXT ("Connected to %hs.\r\n"), szIPAddr) ;

            // Try to receive data. The call will generate an error
            // of WSAEWOULDBLOCK and an event of FD_READ

            recv (sock, (char *) &ulTime, 4, MSG_PEEK) ;
            EditPrintf (hwndEdit, TEXT ("Waiting to receive...")) ;
            return TRUE ;

            // This even occurs when the "recv" call can be made

            case FD_READ:
                KillTimer (hwnd, ID_TIMER) ;
                EditPrintf (hwndEdit, TEXT ("\r\n")) ;

                if (wError)
                {
                    EditPrintf (hwndEdit, TEXT ("FD_READ error #%i."), wError) ;
                    SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
                    return TRUE ;
                }

                // Get the time and swap the bytes

                iSize = recv (sock, (char *) &ulTime, 4, 0) ;
                ulTime = ntohl (ulTime) ;
                EditPrintf (hwndEdit,
                    TEXT ("Received current time of %u seconds ")
                    TEXT ("since Jan. 1 1900.\r\n"), ulTime) ;
```

```

        // Change the system time

        ChangeSystemTime (hwndEdit, ulTime) ;
        SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
        return TRUE ;
    }
    return FALSE ;
}

}
return FALSE ;
}

BOOL CALLBACK ServerDlg (    HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static char *    szServer ;
    static WORD        wServer = IDC_SERVER1 ;
    char                szLabel [64] ;

    switch (message)
    {
    case WM_INITDIALOG:
        szServer = (char *) lParam ;
        CheckRadioButton (hwnd, IDC_SERVER1, IDC_SERVER10, wServer) ;
        return TRUE ;

    case WM_COMMAND:
        switch (LOWORD (wParam))
        {
        case IDC_SERVER1:
        case IDC_SERVER2:
        case IDC_SERVER3:
        case IDC_SERVER4:
        case IDC_SERVER5:
        case IDC_SERVER6:
        case IDC_SERVER7:
        case IDC_SERVER8:
        case IDC_SERVER9:
        case IDC_SERVER10:
            wServer = LOWORD (wParam) ;
            return TRUE ;

        case IDOK:
            GetDlgItemTextA    (hwnd,    wServer,
szLabel, sizeof (szLabel)) ;

            strtok (szLabel, "(") ;
            strcpy (szServer, strtok (NULL, "(")) ;
            EndDialog (hwnd, TRUE) ;

```

```

        return TRUE ;

        case IDCANCEL:
            EndDialog (hwnd, FALSE) ;
            return TRUE ;
    }
    break ;
}
return FALSE ;
}

void ChangeSystemTime (HWND hwndEdit, ULONG ulTime)
{
    FILETIME                ftNew ;
    LARGE_INTEGER            li ;
    SYSTEMTIME                stOld, stNew ;

    GetLocalTime (&stOld) ;
    stNew.wYear                = 1900 ;
    stNew.wMonth                = 1 ;
    stNew.wDay                = 1 ;
    stNew.wHour                = 0 ;
    stNew.wMinute                = 0 ;
    stNew.wSecond                = 0 ;
    stNew.wMilliseconds        = 0 ;

    SystemTimeToFileTime (&stNew, &ftNew) ;
    li = * (LARGE_INTEGER *) &ftNew ;
    li.QuadPart += (LONGLONG) 10000000 * ulTime ;
    ftNew = * (FILETIME *) &li ;
    FileTimeToSystemTime (&ftNew, &stNew) ;

    if (SetSystemTime (&stNew))
    {
        GetLocalTime (&stNew) ;
        FormatUpdatedTime (hwndEdit, &stOld, &stNew) ;
    }
    else
        EditPrintf (hwndEdit, TEXT ("Could NOT set new date and time. ")) ;
}

void FormatUpdatedTime (    HWND hwndEdit, SYSTEMTIME * pstOld, SYSTEMTIME *
pstNew)
{
    TCHAR szDateOld [64], szTimeOld [64], szDateNew [64], szTimeNew [64] ;
    GetDateFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
DATE_SHORTDATE,
                                pstOld, NULL, szDateOld, sizeof

```

```

(szDateOld)) ;
    GetTimeFormat ( LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
                    TIME_NOTIMEMARKER | TIME_FORCE24HOURFORMAT,
                    pstOld, NULL, szTimeOld, sizeof (szTimeOld)) ;

    GetDateFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
DATE_SHORTDATE,
                    pstNew, NULL, szDateNew, sizeof (szDateNew)) ;
    GetTimeFormat ( LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
                    TIME_NOTIMEMARKER | TIME_FORCE24HOURFORMAT,
                    pstNew, NULL, szTimeNew, sizeof (szTimeNew)) ;

    EditPrintf (hwndEdit, TEXT ("System date and time successfully changed ")
                TEXT ("from\r\n\t%s,   %s.%03i   to\r\n\t%s,
%s.%03i."),
                szDateOld, szTimeOld, pstOld->wMilliseconds,
                szDateNew, szTimeNew, pstNew->wMilliseconds) ;
}

void EditPrintf (HWND hwndEdit, TCHAR * szFormat, ...)
{
    TCHAR          szBuffer [1024] ;
    va_list        pArgList ;

    va_start (pArgList, szFormat) ;
    wvsprintf (szBuffer, szFormat, pArgList) ;
    va_end (pArgList) ;
    SendMessage (hwndEdit, EM_SETSEL, (WPARAM) -1, (LPARAM) -1) ;
    SendMessage (hwndEdit, EM_REPLACESEL, FALSE, (LPARAM) szBuffer) ;
    SendMessage (hwn dEdit, EM_SCROLLCARET, 0, 0) ;
}

```

NETTIME.RC (摘录)

```

//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////
/
// Dialog
SERVERS DIALOG DISCARDABLE 20, 20, 274, 202
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "NIST Time Service Servers"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON        "OK", IDOK, 73, 181, 50, 14
    PUSHBUTTON           "Cancel", IDCANCEL, 150, 181, 50, 14
    CONTROL
        "time-a.timefreq.bldrdoc.gov (132.163.135.130) NIST, Boulder,

```

```

Colorado",
                                IDC_SERVER1,"Button",BS_AUTORADIOBUTTON,9,7,256,16
        CONTROL
        "time-b.timefreq.bldrdoc.gov (132.163.135.131) NIST, Boulder,
Colorado",
                                IDC_SERVER2,"Button",BS_AUTORADIOBUTTON,9,24,256,16
        CONTROL
        "time-c.timefreq.bldrdoc.gov (132.163.135.132) Boulder,
Colorado",
                                IDC_SERVER3,"Button",BS_AUTORADIOBUTTON,9,41,256,16
        CONTROL
        "utcnist.colorado.edu (128.138.140.44) University of Colorado,
Boulder",
                                IDC_SERVER4,"Button",BS_AUTORADIOBUTTON,9,58,256,16
        CONTROL
        "time.nist.gov (192.43.244.18) NCAR, Boulder, Colorado",
IDC_SERVER5,"Button",BS_AUTORADIOBUTTON,9,75,256,16
        CONTROL
        "time-a.nist.gov (129.6.16.35) NIST, Gaithersburg,
Maryland",
                                IDC_SERVER6,"Button",BS_AUTORADIOBUTTON,9,92,256,16
        CONTROL
        "time-b.nist.gov (129.6.16.36) NIST, Gaithersburg,
Maryland",
                                IDC_SERVER7,"Button",BS_AUTORADIOBUTTON,9,109,256,16
        CONTROL
        "time-nw.nist.gov (131.107.1.10) Microsoft, Redmond,
Washington",
                                IDC_SERVER8,"Button",BS_AUTORADIOBUTTON,9,126,256,16
        CONTROL
        "utcnist.reston.mci.net (204.70.131.13) MCI, Reston,
Virginia",
                                IDC_SERVER9,"Button",BS_AUTORADIOBUTTON,9,143,256,16
        CONTROL
        "nist1.data.com (209.0.72.7) Datum, San Jose,
California",
                                IDC_SERVER10,"Button",BS_AUTORADIOBUTTON,9,160,256,16
END
NETTIME DIALOG DISCARDABLE 0, 0, 270, 150 STYLE WS_CHILD FONT 8, "MS Sans Serif"
BEGIN
        DEFPUSHBUTTON        "Set Correct Time",IDOK,95,129,80,14
        PUSHBUTTON           "Close",IDC_CLOSE,183,129,80,14
        PUSHBUTTON           "Select Server...",IDC_SERVER,7,129,80,14
        EDITTEXT              IDC_TEXTOUT,7,7,253,110,ES_MULTILINE
| ES_AUTOVSCROLL |
                                ES_READONLY | WS_VSCROLL |
NOT WS_TABSTOP

```

```
END
```

RESOURCE.H (摘录)

```
// Microsoft Developer Studio generated include file.
// Used by NetTime.rc
```

```
#define IDC_TEXTOUT 101
#define IDC_SERVER1 1001
#define IDC_SERVER2 1002
#define IDC_SERVER3 1003
#define IDC_SERVER4 1004
#define IDC_SERVER5 1005
#define IDC_SERVER6 1006
#define IDC_SERVER7 1007
#define IDC_SERVER8 1008
#define IDC_SERVER9 1009
#define IDC_SERVER10 1010
#define IDC_SERVER 1011
#define IDC_CLOSE 1012
```

在结构上，NETTIME 程式建立了一个依据 NETTIME.RC 中的 NETTIME 所建立的非系统模态对话方块。程式重新定义了视窗的尺寸，以便非系统模态对话方块可以覆盖程式的整个视窗显示区域。对话方块包括一个唯读编辑区（程式用於写入文字资讯）、一个「Select Server」按钮、一个「Set Correct Time」按钮和一个「Close」按钮。「Close」按钮用於终止程式。

MainDlg 中的 szIPAddr 变数用於储存伺服器位址，内定是字串「132.163.135.130」。「Select Server」按钮启动依据 NETTIME.RC 中的 SERVERS 模板建立的对话方块。szIPAddr 变数作为最後一个参数传递给 DialogBoxParam。「Server」对话方块列出了 10 个伺服器（都是从 NIST 网站上逐字复制来的），这些伺服器提供了我们感兴趣的服务。当使用者单击一个伺服器时，ServerDlg 将分析按钮文字，以获得相应的 IP 位址。新位址储存在 szIPAddr 变数中。

当使用者按下「Set Correct Time」按钮时，按钮将产生一个 WM_COMMAND 讯息，其中 wParam 的低字组等於 IDOK。MainDlg 中的 IDOK 处理是大部分 Socket 初始行为发生的地方。

使用 Windows Sockets API 时，任何 Windows 程式必须呼叫的第一个函式是：

```
iError = WSASStartup (wVersion, &WSAData) ;
```

NETTIME 将第一个参数设定为 0x0200（表示 2.0 版本）。传回时，WSAData 结构包含了 Windows Sockets 实作的相关资讯，而且 NETTIME 将显示 szDescription 字串，并简要提供了一些版本资讯。

然後，NETTIME 如下呼叫 socket 函式：

```
sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;
```


第一个参数是一个位址种类，表示此处是某种 Internet 位址。第二个参数表示资料以资料流的形式传回，而不是以资料封包的形式传回（我们需要的资料只有 4 个位元组长，而资料封包适用於较大的资料块）。最後一个参数是一个协定，我们指定使用的 Internet 协定是 TCP。它是 RFC-868 所定义的两个协定之一。socket 函式的传回值储存在 SOCKET 型态的变数中，以便後面的 Socket 函式的呼叫。

NETTIME 下面呼叫的 WSAAsyncSelect 是另一个 Windows 特有的 Socket 函式。此函式用於避免因 Internet 回应过慢而造成应用程式当住。在 WinSock 文件中，有些函式与「阻碍性 (blocking)」有关。也就是说，它们不能保证立即把控制项权传回给程式。WSAAsyncSelect 函式强制阻碍性的函式转为非阻碍性的，即在函式执行完之前把控制项传回给程式。函式的结果以讯息的形式报告给应用程式。WSAAsyncSelect 函式让应用程式指定讯息和接收讯息的视窗的数值。通常，函式的语法如下：

```
WSAAsyncSelect (sock, hwnd, message, iConditions);
```

为此任务，NETTIME 使用程式定义的一个讯息，该讯息称为 WM_SOCKET_NOTIFY。它也用 WSAAsyncSelect 的最後一个参数来指定讯息发送的条件，特别在连结和接收资料时 (FD_CONNECT | FD_READ)。

NETTIME 呼叫的下一个 WinSock 函式是 connect。此函式需要一个指向 Socket 位址结构的指标，对於不同的协定来说，此 Socket 位址结构是不同的。NETTIME 使用为 TCP/IP 设计的结构版本：

```
struct sockaddr_in
{
    short                sin_family;
    u_short              sin_port;
    struct in_addr       sin_addr;
    char                 sin_zero[8];
};
```

其中 in_addr 是用於指定 Internet 位址，它可以用 4 个位元组，或者 2 个无正负号短整数，或者 1 个无正负号长整数来表示。

NETTIME 将 sin_family 栏位设定为 AF_INET，用於表示位址种类。将 sin_port 设定为埠号，这里是时间协定的埠号，RFC-868 显示为 37。但不要像我最初时那样，将此栏位设为 37。当大多数数字通过 Internet 时，结构的这个埠号栏位必须是「big endian」的，即最高的位元组排第一个。Intel 微处理器是 little endian。幸运的是，htons (「host-to-network short」) 函式使位元组翻转，因此 NETTIME 将 sockaddr_in 结构的 sin_port 栏位设定为：

```
htons (IPPORT_TIMESERVER)
```

WINSOCK2.H 中将常数定义为 37。NETTIME 用 inet_addr 函式将储存在

szIPAddr 字符串中的伺服器位址转化为无正负号长整数，该整数用于设定结构的 sin_addr 栏位。

如果应用程序在 Windows 98 下呼叫 connect，而且目前 Windows 没有连接到 Internet，那么将显示「拨号连线」对话方块。这就是所谓的「自动拨号」。在 Windows NT 4.0 中没有实作「自动拨号」，因此如果在 NT 环境下执行，那么在执行 NETTIME 之前，就必须先连结上 Internet。

connect 函式通常会阻碍著后面程式的执行，这是因为连结成功以前需要花些时间。然而，由於 NETTIME 呼叫了 WSAAsyncSelect，所以 connect 不会等待连结，事实上，它会立即传回 SOCKET_ERROR 的值。这并不是出现了错误，这只是表示现在还没有连线成功而已。NETTIME 也不会检查这个传回值，只是呼叫 WSAGetLastError 而已。如果 WSAGetLastError 传回 WSAEWOULDBLOCK（即函式的执行通常要受阻，但这里并没有受阻），那就一切都还很正常。NETTIME 将「Set Correct Time」按钮改成「Cancel」，并设定了一个 1 秒的计时器。WM_TIMER 的处理方式只是在程式视窗中显示句点，以告诉使用者程式仍在执行，系统没有当掉。

连结最终完成时，MainDlg 由 WM_SOCKET_NOTIFY 讯息——NETTIME 在 WSAAsyncSelect 函式中指定的程式自订讯息所通知。lParam 的低字组等於 FD_CONNECT，高字组表示错误。这时的错误可能是程式不能连结到指定的伺服器。NETTIME 还列出了其他 9 个伺服器，供您选择，让您可以试试其他的伺服器。

如果一切顺利，那么 NETTIME 将呼叫 recv（「receive：接收」）函式来读取资料：

```
recv (sock, (char *) &ulTime, 4, MSG_PEEK) ;
```

这意味著，用 4 个位元组来储存 ulTime 变数。最後一个参数表示只是读此资料，并不将其从输入伫列中删除。像 connect 函式一样，recv 传回一个错误代码，以表示函式通常受阻，但这时没有受阻。理论上来说（当然这不大可能），函式至少能传回资料的一部分，然後透过再次呼叫以获得其余的 32 个位元组值。那就是呼叫 recv 函式时带有 MSG_PEEK 选项的原因。

与 connect 函式类似，recv 函式也产生 WM_SOCKET_NOTIFY 讯息，这时带有 FD_READ 的事件代码。NETTIME 通过再次呼叫 recv 来对此回应，这时最後的参数是 0，用于从伫列中删除资料。我将简要讨论一下程式处理接收到的 ulTime 的方法。注意，NETTIME 通过向自己发送 WM_COMMAND 讯息来结束处理，该讯息中 wParam 等於 IDCANCEL。对话方块程序通过呼叫 closesocket 和 WSACleanup 来回应。

再次呼叫 NETTIME 接收的 32 位元的 ulTime 值是从 1990 年 1 月 1 日开始的

0:00 UTC 秒数。但最高顺序的位元组是第一个位元组，因此该值必须通过 `ntohl`（「network-to-host long」）函式处理来调整位元组顺序，以便 Intel 微处理器能够处理。然後，NETTIME 呼叫 `ChangeSystemTime` 函式。

`ChangeSystemTime` 首先取得目前的本地时间——即使用者所在时区和日光节约时间的目前系统时间。将 `SYSTEMTIME` 结构设定为 1900 年 1 月 1 日午夜（0 时）。并将这个 `SYSTEMTIME` 结构传递给 `SystemTimeToFileTime`，将此结构转化为 `FILETIME` 结构。`FILETIME` 实际上只是由两个 32 位元的 `DWORD` 一起组成 64 位元的整数，用来表示从 1601 年 1 月 1 日至今间隔为 100 奈秒（nanosecond）的间隔数。

`ChangeSystemTime` 函式将 `FILETIME` 结构转化为 `LARGE_INTEGER`。它是一个 union，允许 64 位元的值可以被当成两个 32 位元的值使用，或者当成一个 `__int64` 资料型态的 64 位元整数使用（`__int64` 资料型态是 Microsoft 编译器对 ANSI C 标准的扩充）。因此，此值是 1601 年 1 月 1 日到 1900 年 1 月 1 日之间间隔为 100 奈秒的间隔数。这里，添加了 1900 年 1 月 1 日至今间隔为 100 奈秒的间隔数——`ulTime` 的 10,000,000 倍。

然後通过呼叫 `FileTimeToSystemTime` 将作为结果的 `FILETIME` 值转换回 `SYSTEMTIME` 结构。因为时间协定传回目前的 UTC 时间，所以 NETTIME 通过呼叫 `SetSystemTime` 来设定时间，`SetSystemTime` 也依据 UTC。基於显示的目的，程式呼叫 `GetLocalTime` 来获得更新时间。最初的本地时间和新的本地时间一起传递给 `FormatUpdatedTime`，这个函式用 `GetTimeFormat` 函式和 `GetDateFormat` 函式将时间转化为 ASCII 字符串。

如果程式在 Windows NT 下执行，并且使用者没有取得设定时间的许可权，那么 `SetSystemTime` 函式可能失败。如果 `SetSystemTime` 失败，则 NETTIME 将发出一个新时间未设定成功的讯息来指出问题所在。

WININET 和 FTP

`WinInet`（「Windows Internet」）API 是一个高阶函式集，帮助程式写作者使用三个常见的 Internet 协定，这三个协定是：用於 World Wide Web 全球资讯网的超文字传输协定（HTTP: Hypertext Transfer Protocol）、档案传输协定（FTP: File Transfer Protocol）和另一个称为 Gopher 的档案传输协定。`WinInet` 函式的语法与常用的 Windows 档案函式的语法类似，这使得使用这些协定就像使用本地磁碟机上的档案一样容易。`WinInet` API 的文件位於 `/Platform SDK/Internet, Intranet, Extranet Services/Internet Tools and Technologies/WinInet API`。

下面的范例程式将展示如何使用 WinInet API 的 FTP 部分。许多有网站的公司也都有「匿名 FTP」伺服器，这样使用者可以在不输入使用者名称和密码的情况下下载档案。例如，如果您在 Internet Explorer 的地址栏输入 <ftp://ftp.microsoft.com>，那么您就可以浏览 FTP 伺服器上的目录并下载档案。如果进入 <ftp://ftp.cpetzold.com/cpetzold.com/ProgWin/UpdDemo>，那么您将在我的匿名 FTP 伺服器上发现与待会要提到的范例程式一块使用的档案列表。

虽然现今 FTP 服务对大多数的 Web 使用者来说并不是那么方便使用，但它仍然相当有用。例如，应用程式能利用 FTP 从匿名 FTP 伺服器上取得资料，这些取得资料的运作程序几乎完全在台面下处理，而不需要使用者操心。这就是我们将讨论的 UPDDEMO（「update demonstration: 更新范例」）程式的构想。

FTP API 概况

使用 WinInet 的程式必须在所有呼叫 WinInet 函式的原始档案中包括表头档案 WININET.H。程式还必须连结 WININET.LIB。在 Microsoft Visual C++ 中，您可以在「Project Settings」对话方块的「Link」页面标签中指定。执行时，程式将和 WININET.DLL 动态连结程式库连结。

在下面的论述中，我不会详细讨论函式的语法，因为某些函式有很多选项，这让它变得相当复杂。要掌握 WinInet，您可以将 UPDDEMO 原始码当成食谱来看待。这时最重要的是了解有关的各个步骤以及 FTP 函式的范围。

要使用 Windows Internet API，首先要呼叫 InternetOpen。然後，使用 WinInet 支援的任何一种协定。InternetOpen 给您一个 Internet 作业代号，并储存到 HINTERNET 型态的变数中。用完 WinInet API 以後，应该通过呼叫 InternetCloseHandle 来关闭代号。

要使用 FTP，您接下来就要呼叫 InternetConnect。此函式需要使用由 InternetOpen 建立 Internet 作业代号，并且传回 FTP 作业的代号。您可将此代号作为名称开头为 Ftp 的所有函式的第一个参数。InternetConnect 函式的参数指出要使用的 FTP，还提供了伺服器名称，例如，<ftp.cpetzold.com>。此函式还需要使用者名称和密码。如果存取匿名 FTP 伺服器，这些参数可以设定为 NULL。如果应用程式呼叫 InternetConnect 时 PC 并没有连结到 Internet，Windows 98 将显示「拨号连线」对话方块。当使用 FTP 的应用程式结束时，呼叫 InternetCloseHandle 来关闭代号。

这时可以开始呼叫有 Ftp 字首的函式。您将发现这些函式与标准的 Windows 档案 I/O 函式很相似。为了避免与其他协定重复，一些以 Internet 为字首的函

式也可以处理 FTP。

下面四个函式用於处理目录：

```
fSuccess = FtpCreateDirectory      (hFtpSession,      szDirectory) ;
fSuccess = FtpRemoveDirectory      (hFtpSession,      szDirectory) ;
fSuccess = FtpSetCurrentDirectory (hFtpSession,      szDirectory) ;
fSuccess = FtpGetCurrentDirectory (hFtpSession,      szDirectory,
&dwCharacterCount) ;
```

注意，这些函式很像我们所熟悉的 Windows 提供用於处理本地档案系统的 CreateDirectory 、 RemoveDirectory 、 SetCurrentDirectory 和 GetCurrentDirectory 函式。

当然，存取匿名 FTP 的应用程式不能建立或删除目录。而且，程式也不能假定 FTP 目录具有和 Windows 档案系统相同的目录结构型态。特别是用相对路径名设定目录的程式，不能假定关于新的目录全名的一切。如果程式需要知道最後所在目录的整个名称，那么呼叫了 SetCurrentDirectory 之後必须再呼叫 GetCurrentDirectory。GetCurrentDirectory 的字串参数至少包含 MAX_PATH 字元，并且最後一个参数应指向包含该值的变数。

下面两个函式让您删除或者重新命名档案（但不是在匿名 FTP 伺服器上）：

```
fSuccess = FtpDeleteFile (hFtpSession, szFileName) ;
fSuccess = FtpRenameFile (hFtpSession, szOldName, szNewName) ;
```

经由先呼叫 FtpFindFirstFile，可以查找档案（或与含有万用字元的档名样式相符的多个档案）。此函式很像 FindFirstFile 函式，甚至都使用了相同的 WIN32_FIND_DATA 结构。该档案为列举出来的档案传回了一个代号。您可以将此代号传递给 InternetFindNextFile 函式以获得额外的档案名称资讯。最後通过呼叫 InternetCloseHandle 来关闭代号。

要打开档案，可以呼叫 FtpFileOpen。这个函式传回一个档案代号，此代号可以用於 InternetReadFile、InternetReadFileEx、InternetWrite 和 InternetSetFilePointer 呼叫。最後可以通过呼叫最常用的 InternetCloseHandle 函式来关闭代号。

最後，下面两个高级函式特别有用：FtpGetFile 呼叫将档案从 FTP 伺服器复制到本地记忆体，它合并了 FtpFileOpen、FileCreate、InternetReadFile、WriteFile、InternetCloseHandle 和 CloseHandle 呼叫。FtpGetFile 的另一个参数是一个旗标，如果本地已经存在同名档案，那么该旗标将导致函式呼叫失败。FtpPutFile 与此函式类似，用於将档案从本地记忆体复制到 FTP 伺服器。

更新展示程式

UPDDemo，如程式 23-2 所示，展示了用 WinInet FTP 函式在第二个执行绪

执行期间从匿名 FTP 伺服器上下载档案的方法。

程式 23-2 UPDDEMO

```

UPDDEMO.C
/*-----
-
    UPDDEMO.C -- Demonstrates Anonymous FTP Access
                                (c) Charles Petzold, 1998
-----*/

#include      <windows.h>
#include      <wininet.h>
#include      <process.h>
#include      "resource.h"

                // User-defined messages used in WndProc

#define       WM_USER_CHECKFILES      (WM_USER + 1)
#define       WM_USER_GETFILES       (WM_USER + 2)

                // Information for FTP download

#define       FTPSERVER   TEXT   ("ftp.cpetzold.com")
#define       DIRECTORY   TEXT   ("cpetzold.com/ProgWin/UpdDemo")
#define       TEMPLATE    TEXT   ("UD?????.TXT")

                // Structures used for storing filenames and contents
typedef struct
{
    TCHAR * szFilename ;
    char * szContents ;
}
FILEINFO ;
typedef struct
{
    int                iNum ;
    FILEINFO           info[1] ;
}
FILELIST ;
                // Structure used for second thread
typedef struct
{
    BOOL bContinue ;
    HWND hwnd ;
}
PARAMS ;

                // Declarations of all functions in program

```

```

LRESULT      CALLBACK      WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL          CALLBACK      DlgProc (HWND, UINT, WPARAM, LPARAM) ;
VOID          FtpThread (PVOID) ;
VOID          ButtonSwitch (HWND, HWND, TCHAR *) ;
FILELIST *    GetFileList (VOID) ;
int           Compare (const FILEINFO *, const FILEINFO *) ;

// A couple globals

HINSTANCE hInst ;
TCHAR      szAppName[] = TEXT ("UpdDemo") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR      szCmdLine,      int
iCmdShow)
{
    HWND      hwnd ;
    MSG        msg ;
    WNDCLASS   wndclass ;

    hInst = hInstance ;
    wndclass.style          = 0 ;
    wndclass.lpfnWndProc    = WndProc ;
    wndclass.cbClsExtra     = 0 ;
    wndclass.cbWndExtra     = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor        = NULL ;
    wndclass.hbrBackground  = GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName   = NULL ;
    wndclass.lpszClassName  = szAppName ;

    if (!RegisterClass (&wndclass))
    {
        MessageBox (    NULL, TEXT ("This program requires Windows NT!"),
                        szAppName, MB_ICONERROR) ;

        return 0 ;
    }

    hwnd = CreateWindow (  szAppName, TEXT ("Update Demo with Anonymous FTP"),
        WS_OVERLAPPEDWINDOW | WS_VSCROLL,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

```

```

        // After window is displayed, check if the latest file exists
        SendMessage (hwnd, WM_USER_CHECKFILES, 0, 0) ;
        while (GetMessage (&msg, NULL, 0, 0))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
        return msg.wParam ;
    }

LRESULT CALLBACK WndProc (   HWND hwnd,   UINT message,   WPARAM wParam,   LPARAM
lParam)
{
    static FILELIST *plist ;
    static int                cxClient, cyClient, cxChar, cyChar ;
    HDC                        hdc ;
    int                        i ;
    PAINTSTRUCT                ps ;
    SCROLLINFO                 si ;
    SYSTEMTIME                 st ;
    TCHAR                      szFilename [MAX_PATH] ;

    switch (message)
    {
        case WM_CREATE:
            cxChar = LOWORD (GetDialogBaseUnits ()) ;
            cyChar = HIWORD (GetDialogBaseUnits ()) ;
            return 0 ;

        case WM_SIZE:
            cxClient      = LOWORD (lParam) ;
            cyClient      = HIWORD (lParam) ;

            si.cbSize     = sizeof (SCROLLINFO) ;
            si.fMask       = SIF_RANGE | SIF_PAGE ;
            si.nMin        = 0 ;
            si.nMax        = plist ? plist->iNum - 1 : 0 ;
            si.nPage       = cyClient / cyChar ;

            SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
            return 0 ;

        case WM_VSCROLL:
            si.cbSize     = sizeof (SCROLLINFO) ;
            si.fMask       = SIF_POS | SIF_RANGE | SIF_PAGE ;
            GetScrollInfo (hwnd, SB_VERT, &si) ;

            switch (LOWORD (wParam))

```



```

        {
        case SB_LINEDOWN: si.nPos += 1 ; break ;
        case SB_LINEUP:   si.nPos -= 1 ; break ;
        case SB_PAGEDOWN: si.nPos += si.nPage ; break ;
        case SB_PAGEUP:   si.nPos -= si.nPage ; break ;
        case SB_THUMBPOSITION: si.nPos = HIWORD (wParam) ;

break ;

        default:          return 0 ;
        }
        si.fMask = SIF_POS ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

case WM_USER_CHECKFILES:
        // Get the system date & form filename from year and month

        GetSystemTime (&st) ;
        wsprintf (szFilename, TEXT ("UD%04i%02i.TXT"), st.wYear, st.wMonth) ;

        // Check if the file exists; if so, read all the files

        if (GetFileAttributes (szFilename) != (DWORD) -1)
        {
                SendMessage (hwnd, WM_USER_GETFILES, 0, 0) ;
                return 0 ;
        }
        // Otherwise, get files from Internet.
        // But first check so we don't try to copy files to a CD-ROM!

        if (GetDriveType (NULL) == DRIVE_CDROM)
        {
                MessageBox (hwnd, TEXT ("Cannot run this program from CD-ROM!"),
                        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                return 0 ;
        }

        // Ask user if an Internet connection is desired

        if (IDYES == MessageBox (hwnd, TEXT ("Update information
from Internet?"),
                szAppName, MB_YESNO | MB_ICONQUESTION))

                // Invoke dialog box

                DialogBox (hInst, szAppName, hwnd, DlgProc) ;

                // Update display

```

```
        SendMessage (hwnd, WM_USER_GETFILES, 0, 0) ;
        return 0 ;

case WM_USER_GETFILES:
        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
        ShowCursor (TRUE) ;

        // Read in all the disk files

        plist = GetFileList () ;

        ShowCursor (FALSE) ;
        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

        // Simulate a WM_SIZE message to alter scroll bar & repaint

        SendMessage (hwnd, WM_SIZE, 0, MAKELONG (cxClient, cyClient)) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        SetTextAlign (hdc, TA_UPDATECP) ;

        si.cbSize = sizeof (SCROLLINFO) ;
        si.fMask = SIF_POS ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;

        if (plist)
        {
                for (i = 0 ; i < plist->iNum ; i++)
                {
                        MoveToEx (hdc, cxChar, (i - si.nPos) * cyChar, NULL) ;
                        TextOut (hdc, 0, 0, plist->info[i].szFilename,
                                lstrlen (plist->info[i].szFilename)) ;
                        TextOut (hdc, 0, 0, TEXT (": "), 2) ;
                        TextOutA (hdc, 0, 0, plist->info[i].szContents,
                                strlen (plist->info[i].szContents)) ;
                }
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;

case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
```

```

}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static PARAMS params ;
    switch (message)
    {
        case WM_INITDIALOG:
            params.bContinue = TRUE ;
            params.hwnd = hwnd ;

            _beginthread (FtpThread, 0, 0) ;
            return TRUE ;

        case WM_COMMAND:
            switch (LOWORD (wParam))
            {
                case IDCANCEL: // button for user to abort download
                    params.bContinue = FALSE ;
                    return TRUE ;

                case IDOK: // button to make dialog box go away
                    EndDialog (hwnd, 0) ;
                    return TRUE ;
            }
    }
    return FALSE ;
}

/*-----
-
    FtpThread: Reads files from FTP server and copies them to local disk
-----
-*/

void FtpThread (PVOID parg)
{
    BOOL                                bSuccess ;
    HINTERNET                          hIntSession, hFtpSession, hFind ;
    HWND                                hwndStatus, hwndButton ;
    PARAMS                              *    pparams ;
    TCHAR                              szBuffer [64] ;
    WIN32_FIND_DATA                    finddata ;

    pparams = parg ;
    hwndStatus = GetDlgItem (pparams->hwnd, IDC_STATUS) ;
    hwndButton = GetDlgItem (pparams->hwnd, IDCANCEL) ;

```

```
        // Open an internet session

hIntSession = InternetOpen (szAppName, INTERNET_OPEN_TYPE_PRECONFIG,
                            NULL, NULL, INTERNET_FLAG_ASYNC) ;
if (hIntSession == NULL)
{
    wsprintf (szBuffer, TEXT ("InternetOpen error %i"), GetLastError ()) ;
    ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
    _endthread () ;
}

SetWindowText (hwndStatus, TEXT ("Internet session opened...")) ;
        // Check if user has pressed Cancel
if (!pparams->bContinue)
{
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, NULL) ;
    _endthread () ;
}

        // Open an FTP session.
hFtpSession      =      InternetConnect      (hIntSession,      FTPSERVER,
INTERNET_DEFAULT_FTP_PORT,
                            NULL, NULL, INTERNET_SERVICE_FTP, 0, 0) ;
if (hFtpSession == NULL)
{
    InternetCloseHandle (hIntSession) ;
    wsprintf (szBuffer, TEXT ("InternetConnect error %i"),
                                                GetLastError ()) ;
    ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
    _endthread () ;
}

SetWindowText (hwndStatus, TEXT ("FTP Session opened...")) ;
        // Check if user has pressed Cancel
if (!pparams->bContinue)
{
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, NULL) ;
    _endthread () ;
}

        // Set the directory
bSuccess = FtpSetCurrentDirectory (hFtpSession, DIRECTORY) ;
if (!bSuccess)
{

```

```

        InternetCloseHandle (hFtpSession) ;
        InternetCloseHandle (hIntSession) ;
        wsprintf ( szBuffer, TEXT ("Cannot set directory to %s"),
                                DIRECTORY) ;
        ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
        _endthread () ;
    }

    SetWindowText (hwndStatus, TEXT ("Directory found...")) ;
        // Check if user has pressed Cancel
    if (!pparams->bContinue)
    {
        InternetCloseHandle (hFtpSession) ;
        InternetCloseHandle (hIntSession) ;
        ButtonSwitch (hwndStatus, hwndButton, NULL) ;
        _endthread () ;
    }

        // Get the first file fitting the template
    hFind = FtpFindFirstFile (hFtpSession, TEMPLATE, &finddata, 0, 0) ;
    if (hFind == NULL)
    {
        InternetCloseHandle (hFtpSession) ;
        InternetCloseHandle (hIntSession) ;
        ButtonSwitch (hwndStatus, hwndButton, TEXT ("Cannot find files")) ;
        _endthread () ;
    }

do
{
        // Check if user has pressed Cancel
    if (!pparams->bContinue)
    {
        InternetCloseHandle (hFind) ;
        InternetCloseHandle (hFtpSession) ;
        InternetCloseHandle (hIntSession) ;
        ButtonSwitch (hwndStatus, hwndButton, NULL) ;
        _endthread () ;
    }

        // Copy file from internet to local hard disk, but fail
        // if the file already exists locally

        wsprintf (szBuffer, TEXT ("Reading file %s..."),
finddata.cFileName) ;
        SetWindowText (hwndStatus, szBuffer) ;

        FtpGetFile ( hFtpSession,
finddata.cFileName, finddata.cFileName, TRUE,

```

```

        FILE_ATTRIBUTE_NORMAL, FTP_TRANSFER_TYPE_BINARY, 0) ;
    }
    while (InternetFindNextFile (hFind, &finddata)) ;
    InternetCloseHandle (hFind) ;
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;

    ButtonSwitch (hwndStatus, hwndButton, TEXT ("Internet Download Complete"));
}

/*-----
--
    ButtonSwitch:  Displays final status message and changes Cancel to OK
-----*/
VOID ButtonSwitch (HWND hwndStatus, HWND hwndButton, TCHAR * szText)
{
    if (szText)
        SetWindowText (hwndStatus, szText) ;
    else
        SetWindowText (hwndStatus, TEXT ("Internet Session
Cancelled")) ;
    SetWindowText (hwndButton, TEXT ("OK")) ;
    SetWindowLong (hwndButton, GWL_ID, IDOK) ;
}

/*-----
-
    GetFileList:  Reads files from disk and saves their names and contents
-----
-*/

FILELIST * GetFileList (void)
{
    DWORD                                dwRead ;
    FILELIST                            *   plist ;
    HANDLE                                hFile, hFind ;
    int                                    iSize, iNum ;
    WIN32_FIND_DATA                      finddata ;

    hFind = FindFirstFile (TEMPLATE, &finddata) ;
    if (hFind == INVALID_HANDLE_VALUE)
        return NULL ;

    plist      = NULL ;
    iNum       = 0 ;

    do
    {

```

```
        // Open the file and get the size
        hFile = CreateFile (finddata.cFileName,  GENERIC_READ,
FILE_SHARE_READ,
        NULL, OPEN_EXISTING, 0, NULL) ;

        if (hFile == INVALID_HANDLE_VALUE)
            continue ;

        iSize = GetFileSize (hFile, NULL) ;

        if (iSize == (DWORD) -1)
        {
            CloseHandle (hFile) ;
            continue ;
        }

        // Realloc the FILELIST structure for a new entry

        plist = realloc (plist, sizeof (FILELIST) + iNum * sizeof (FILEINFO));

        // Allocate space and save the filename

        plist->info[iNum].szFilename = malloc (lstrlen
(finddata.cFileName) + sizeof (TCHAR)) ;
        lstrcpy (plist->info[iNum].szFilename, finddata.cFileName) ;

        // Allocate space and save the contents

        plist->info[iNum].szContents = malloc (iSize + 1) ;
        ReadFile (hFile, plist->info[iNum].szContents, iSize, &dwRead, NULL);
        plist->info[iNum].szContents[iSize] = 0 ;

        CloseHandle (hFile) ;
        iNum ++ ;
    }
    while (FindNextFile (hFind, &finddata)) ;
    FindClose (hFind) ;

        // Sort the files by filename
    qsort (plist->info, iNum, sizeof (FILEINFO), Compare) ;
    plist->iNum = iNum ;
    return plist ;
}

/*-----
-
    Compare function for qsort
-----
*/
```

```

int Compare (const FILEINFO * pinfo1, const FILEINFO * pinfo2)
{
    return lstrcmp (pinfo2->szFilename, pinfo1->szFilename) ;
}

UPDDemo.RC (摘录)
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

////////////////////////////////////
/
// Dialog
UPDDemo DIALOG DISCARDABLE 20, 20, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Internet Download"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON        Cancel", IDCANCEL, 69, 74, 50, 14
    CTEXT              "", IDC_STATUS, 7, 29, 172, 21
END

RESOURCE.H (摘录)
// Microsoft Developer Studio generated include file.
// Used by UpdDemo.rc

#define IDC_STATUS      40001

```

UPDDemo 使用的档案名称是 UDyyyymm.TXT，其中 yyyy 是 4 位阿拉伯数字的年数（当然适用於 2000），mm 是 2 位阿拉伯数字的月数。这里假定程式可以享有每个月都有更新档案的好处。这些档案可能是整个月刊，而由於阅读效率上的考虑，让程式将其下载到本地储存媒体上。

因此，WinMain 在呼叫 ShowWindow 和 UpdateWindow 来显示 UPDDemo 主视窗以後，向 WndProc 发送程式定义的 WM_USER_CHECKFILES 讯息。WndProc 通过获得目前的年、月并检查该年月 UDyyyymm.TXT 档案所在的内定目录来处理此讯息。这种档案的存在意义在於 UPDDemo 会被完全更新（当然，事实并非如此。一些过时的档案将漏掉。如果要做得更完整，程式得进行更广泛的检测）。在这种情况下，UPDDemo 向自己发送一个 WM_USER_GETFILES 讯息，它通过呼叫 GetFileList 函式来处理。这是 UPDDemo.C 中稍长的一个函式，但它并不是特别有用，它所做的全部工作就是将所有的 UDyyyymm.TXT 档案读到动态配置的 FILELIST 型态结构中，该结构是在程式顶部定义的，然後让程式在其显示区域显示这些档案的内容。

如果 UPDDemo 没有最新的档案，那么它必须透过 Internet 进行更新。程式首先询问使用者这样做是否「OK」。如果是，程式将显示一个简单的对话方块，

其中只有一个「Cancel」按钮和一个 ID 为 IDC_STATUS 的静态文字区。下载时，此静态文字区向使用者提供状态报告，并且允许使用者取消过於缓慢的更新作业。此对话程序的名称是 DlgProc。

DlgProc 很短，它建立了一个包括自身视窗代号的 PARAMS 型态的结构以及一个名称为 bContinue 的 BOOL 变数，然後呼叫 `_beginthread` 来执行第二个执行绪。

FtpThread 函式透过使用下面的呼叫来完成实际的传输：`InternetOpen`、`InternetConnect`、`FtpSetCurrentDirectory`、`FtpFindFirstFile`、`InternetFindNextFile`、`FtpGetFile` 和 `InternetCloseHandle`（三次）。如同大多数程式码，该执行绪函式如果略过错误检查、让使用者了解下一步的操作情况以及允许使用者随意取消整个显示的那些步骤，那么它将变得简洁许多。FtpThread 函式透过用 `hwndStatus` 代号呼叫 `SetWindowText` 来让使用者知道进展情况，这里指的是对话方块中间的静态文字区。

执行绪可以依照下面的三种方式之一来终止：

第一种，FtpThread 可能遇到从 `WinInet` 函式传回的错误。如果是这样，它将清除并编排错误字串的格式，然後将此字串（连同对话方块文字区代号和「Cancel」按钮的代号一起）传递给 `ButtonSwitch`。`ButtonSwitch` 是一个小函式，它显示了文字字串，并将「Cancel」按钮转换成「OK」按钮——不只是按钮上的文字字串的转换，还包括控制项 ID 的转换。这样就允许使用者按下「OK」按钮来结束对话方块。

第二种方式，FtpThread 能在没有任何错误的情况下完成任务，其处理方法和遇到错误时的方法一样，只不过对话方块中显示的字串为「Internet Download Complete」。

第三种方式，使用者可以在程序中选择取消下载。这时，DlgProc 将 PARAMS 结构的 `bContinue` 栏位设定为 `FALSE`。FtpThread 频繁地检查该值，如果 `bContinue` 等於 `FALSE`，那么函式将做好应该进行的收拾工作，并以 `NULL` 文字参数呼叫 `ButtonSwitch`，此参数表示显示了字串「Internet Session Cancelled」。同样，使用者必须按下「OK」按钮来关闭对话方块。

虽然 UPDDemo 取得的每个档案只能显示一行，但我（本书的作者）可以用这个程式来告诉您（本书的读者）本书的更新内容以及其他资讯，您也可以在网站上发现更详细的资讯。因此，UPDDemo 成为我向您传送资讯的方法，并且可以让本书的内容延续到最後一页之後。

第二十四章 附录

侯捷计算机领域的术语对照（英中繁简）。

● 「式」：

constructor	建构式
declaration	宣告式
definition	定义式
destructor	解构式
expression	算式（运算式）
function	函式
pattern	范式、模式、样式
program	程式
signature	标记式

● 「件」：（这是个弹性非常大的可组合字）

assembly	（装）配件
component	组件
construct	构件
control	控件
event	事件
hardware	硬件
object	物件
part	零件、部件
singleton	单件
software	软件
work	工件、机件

● 「器」：

adapter	配接器
allocator	配置器
compiler	编译器
container	容器
iterator	迭代器
linker	联（连）结器

listener	监听器
----------	-----

● 「别」：

class	类别
type	型别

● 「化」：

generalized	泛化
specialized	特化
overloaded	多载化 (重载)

● 「型」：

polymorphism	多型
genericity	泛型

● 「程」：

process	行程 (or 进程, 大陆用语)
thread	线程 (大陆用语)
programming	编程

● 英中繁简编程术语对照表

英文术语	繁体	简体
#define	定义	预定义
abstract	抽象的	抽象的
abstraction	抽象体、抽象物、抽象性	抽象体、抽象物、抽象性
access	存取、取用	存取、访问
access function	存取函式	存取函数
activate		
active		
adapter	配接器	适配器
address	位址	地址
address space	位址空间, 定址空间	
address-of operator	取址运算符	取地址运算符
aggregation	聚合	
algorithm	演算法	算法

allocate	配置	分配
allocator	(空间) 配置器	分配器
application	應用程式	应用、应用程序
application framework	應用程式框架、应用框架	应用程序框架
argument	引数 (传给函式的值)。参见 parameter	叁数、实质叁数、实叁、自变量
array	阵列	数组
arrow operator	arrow (箭头) 运算符	箭头运算符
assembly	配件	
assembly language	组合语言	汇编语言
assign	指派、指定、设值、赋值	赋值
assignment	指派、指定	赋值、分配
assignment operator	指派 (赋值) 运算符 =	赋值运算符
associated	相应的、相关的	相关的、关联、相应的
associative container	关 联 式 容 器 (对 应 sequential container)	关联式容器
atomic	不可分割的	原子的
attribute	属性	特性
background	背景	背景 (用於图形着色) 後台 (用於行程)
base class	基础类别	基类
base type	基础型别 (等同於 base class)	
batch	批次 (意思是整批作业)	批处理
best viable function	最佳可行函式 (从 viable functions 中挑出的最佳吻合者)	最佳可行函式
binary search	二分搜寻法	二分查找
binary tree	二元树	二叉树
binary operator	二元运算符	二元运算符
binding	系结	绑定
bit	位元	位
bit field	位元栏 ?	位域
bitmap	位元图 ?	位图
bitwise	以 bit 为单元逐一---	?

bitwise copy	以 bit 为单元进行复制；位元逐一复制	位拷贝
block	区块	块、区块、语句块
boolean	布林值（真假值，true 或 false）	布尔值
border	边框、框线	边框
brace (curly brace)	大括弧、大括号	花括弧、花括号
bracket (square bracket)	中括弧、中括号	方括弧、方括号
breakpoint	中断点	断点
build-in	内建	内置
bus	汇流排	
byte	位元组（由 8 bits 组成）	字节
cache	快取	高速缓存
call	呼叫、调用	调用
callback	回呼	回调
call operator	call（函式呼叫）运算符（同 function call operator）	调用运算符
candidate function	候选函式（在函式多载决议程序中出现的候选函式）	候选函数
chain	串链（例 chain of function calls）	链
character	字元	字符
check box	核取方块（i.e. check button）	复选框
check button	方钮（i.e. check box）	复选按钮
child class	子类别（或称为 derived class, subtype）	子类
class	类别	类
class body	类别本体	类体？
class declaration	类别宣告、类别宣告式	类声明
class definition	类别定义、类别定义式	类定义
class derivation list	类别衍化列	类继承列表
class head	类别表头	类头？

class hierarchy	类别继承体系, 类别阶层	类层次体系
class library	类别程式库、类别库	类库
class template	类别模板、类别范本	类模板
class template partial specializations	类别模板偏特化	类模板部分特化
class template specializations	类别模板特化	类模板特化
cleanup	清理、善後	清理、清除
client	客端、客户端、用户端	客户端
client-server	主从架构	客户/服务器
clipboard	剪贴簿	剪贴板
clone	复制 (易与 copy 混淆)	克隆
collection	群集	集合 ?
combo box	复合方块、复合框	组合框
command line	命令列 (系统文字模式下的整行执行命令)	命令行
communication	通讯	通讯
compile time	编译期	编译期、编译时
compiler	编译器	编译器
component	组件	组件
composition	复合、合成、组合	组合
computer	电脑、计算机	计算机、电脑
concrete	具象的	实在的
concurrent	并行	并发
configuration	组态	配置
container	容器 (存放资料的某种结构如 list, vector...)	容器
context	背景关系、周遭环境、上下脉络	环境、上下文
control	控制元件、控件	控件
const	常数 (constant 的缩写, C++ 关键字)	
constant	常数 (相对於 variable)	常量、常数

constructor (ctor)	建构式 (与 class 同名的一种 member functions)	构造函数、构造器
copy	复制、拷贝	拷贝
cover	涵盖	覆盖
create	产生、生成	创建、生成
creation	产生、生成	创建、生成
data	资料	数据
data member	资料成员、成员变数	数据成员、成员变量
data structure	资料结构	数据结构
datagram	资料元	数据报文
dead lock	死结	死锁
debug	除错	调试
declaration	宣告、宣告式	声明
deduction	推导 (例: template argument deduction)	推导、推断
default	预设	缺省、默认
definition	定义、定义区、定义式	定义
delegate	委派、委托、委任	
delegation	(同上)	
dereference	提领 (取出指标所指物体的内容)	解参考
dereference operator	dereference (提领) 运算符 *	解参考算符
derived class	衍生类别	派生类
design by contract	契约式设计	
design pattern	设计样式 ※ 最近我比较喜欢「设计范式」一词	设计模式
destructor (dtor)	解构式	析构函数、析构器
device	装置、设备	设备
dialog	对话框、对话盒	对话框
directive	指令 (例: using directive)	(编译) 指示符
directory	目录	目录
distributed computing	分布式计算 (分布式电算)	分布式计算

	分散式计算 (分散式电算)	
document	文件	文档
dot operator	dot (句点) 运算符 .	(圆)点运算符
driver	驱动程序	驱动 (程序)
dynamic binding	动态系结	动态绑定
efficiency	高效、效率、效能	
entity	物体	实体、物体
encapsulation	封装	封装
enclosing class	外围类别 (与巢状类别 nested class 有关)	外围类
enum (enumeration)	列举 (一种 C++ 资料型别)	枚举
enumerators	列举元 (enum 型别中的成员)	枚举成员、枚举器
equality operator	equality(等号)运算符 ==	等号运算符
evaluate	评估、求值、核定	评估
event	事件	事件
event driven	事件驱动的	事件驱动的
exception	异常情况	异常
exception declaration	异常宣告 (ref. C++ Primer 3/e, 11.3)	异常声明
exception handling	异常处理、异常处理机制	异常处理、异常处理机制
exception specification	异常规格 (ref. C++ Primer 3/e, 11.4)	异常规范
exit	退离 (指离开函式时的那一个 执行点)	退出
explicit	明白的、明显的、显式	显式
export	汇出	引出、导出
expression	运算式、算式	表达式
facility	设施、设备	设施、设备
feature	特性	
field	栏位	字段
file	档案	文件
firmware	韧体	固件
flush	清理、扫清	刷新
form	表单 (programming 用语)	

formal parameter	形式参数	形式参数
forward declaration	前置宣告	前置声明
fractal	碎形	分形
framework	框架	框架
full specialization	全特化 (ref. partial specialization)	?
function	函式、函数	函数
function call operator	同 call operator	
function object	函式物件 (ref. C++ Primer 3/e, 12.3)	函数对象
function overloaded resolution	函式多载决议程序	函数重载解决 (方案)
function template	函式模板、函式范本	函数模板
functor	仿函式	仿函式、函子
generic	泛型、一般化的	一般化的、通用的、泛化
generic algorithm	泛型演算法	通用算法
global	全域性的 (对应於 local)	全局的
global scope resolution operator	全域生存空间 (范围决议) 运算符 ::	全局范围解析运算符
group	群组	?
group box	群组方块	分组框
hand shaking	握手协商	
handle	识别码、识别号、号码牌、权柄	句柄
handler	处理常式	处理函数
hardware	硬体	硬件
hash table	杂凑表	哈希表、散列表
header file	表头档、标头档	头文件
heap	堆积	堆
hierarchy	阶层体系	层次结构 (体系)
hook	挂钩	钩子
hyperlink	超链接	超链接
IDE	整合开发环境	集成开发环境
identifier	识别字、识别符号	标识符
immediate base	直接的 (紧临的) 上层 base class。	直接上层基类

immediate derived	直接的 (紧临的) 下层 derived class。	直接下层派生类
implement	实作	实现
implementation	实作品、实作物、实作体、实作码	实现
implicit	隐喻的、暗自的、隐式	隐式
import	汇入	导入
increment operator	累加运算符 ++	增加运算符
information	资讯	信息
infrastructure	公共基础建设	
inheritance	继承、继承机制	继承、继承机制
inline	行内	内联
inline expansion	行内展开	内联展开
initialization	初始化 (动作)	初始化
initialization list	初值列	初始值列表
initialize	初始化	初始化
instance	实体 (根据某种表述而实际产生的「东西」)	实例
instantiated	具现化、实体化 (常应用於 template)	实例化
instantiation	具现体、具现化实体 (常应用於 template)	实例
integrate	整合	集成
interface	介面	接口
invoke	唤起	调用
iterate	迭代 (回圈一个轮回一个轮回地进行)	迭代
iterative	反覆的, 迭代的	
iterator	迭代器 (一种泛型指标)	迭代器
iteration	迭代 (回圈每次轮回称为一个 iteration)	迭代
item	项目、条款	项、条款、项目
laser	雷射	激光
level	阶	层

例 high level	高阶	高层
library	程式库、函式库	库、函数库
lifetime	生命期、寿命	生命期、寿命
link	联结、连结	连接
linker	联结器、连结器	连接器
literal constant	字面常数 (例 3.14 或 "hi" 这等常数值)	字面常数
list	串列 (linked-list)	列表、表、链表
list box	列表方块、列表框	列表框
load	载入	装载、加载
loader	载入器	装载器、载入器
local	区域性的 (对应於 global)	局部的
lock	机锁	
loop	回圈	循环
lvalue	左值	左值
macro	巨集	宏
maintain	维护	维护
manipulator	操纵器 (iostream 预先定义的一种东西)	操纵器
mechanism	机制	机制
member	成员	成员
member access operator	成员取用运算符 (有 dot 和 arrow 两种)	成员存取运算符
member function	成员函式	成员函数
member initialization list	成员初值列	成员初始值列表
memberwise	以 member 为单元 ---、members 逐一---	以成员为单位
memberwise copy	以 members 为单元逐一复制	
memory	记忆体	内存
menu	表单、选单	菜单
message	讯息	消息
message based	以讯息为基础的	基於消息的
message loop	讯息回圈	消息环
method (java)	方法、行为	

micro	微	微
modeling	模塑	
modeling language	塑模语言, 建模语言	
module	模组	模块
most derived class	最末层衍生类别	最底层的派生类
mouse	滑鼠	鼠标
mutable	可变的	可变的
multi-tasking	多工	多任务
namespace	命名空间	名字空间、命名空间
nested class	巢状类别	嵌套类
object	物件	对象
object based	以物件为基础的	基於对象的
object model	物件模型	对象模型
object oriented	物件导向的	面向对象的
online	线上	在线
operand	运算元	操作数
operating system (OS)	作业系统	操作系统
operation	操作、操作行为	操作
operator	运算符	操作符、运算符
option	选项	选项
overflow	上限溢位 (相对于 underflow)	溢出 (underflow: 下溢)
overhead	额外负担	额外开销
overload	多载化、多载化、重载	重载
overloaded function	多载化函式	重载的函数
overloaded operator	多载化运算符	被重载的运算符
overloaded set	多载集合	重载集合
override	改写、覆写 (在 derived class 中重新定义虚拟函式)	重载、改写、重新定义
package	套件	
pair	对组	
palette	调色盘、组件盘、工具箱	
pane	窗格 (有时为嵌板之意, 例 Java Content Pane)	窗格
parameter	叁数 (函式参数列上的变数)	叁数、形式叁数、形叁

parameter list	参数列	参数列表
parent class	父类别 (或称 base class)	父类
parentheses	小括弧、小括号	圆括弧、圆括号
parse	解析	解析
partial specialization	偏特化 (ref. C++ Primer 3/e, 16.10) (ref. full specialization)	局部特化
pass by address	传址 (函式引数的传递方式)	传地址
pass by reference	传址 (函式引数的传递方式)	传地址
pass by value	传值 (函式引数的传递方式)	传值
pattern	样式 ※ 最近我比较喜欢「范式」一词	模式
performance	效率	性能
pixel	图素、像素	像素
placement delete	ref. C++ Primer 3/e, 15.8.2	
placement new	ref. C++ Primer 3/e, 15.8.2	
platform	平台	平台
pointer	指标 址位器 (和址叁器 reference 形成对映, 满好)	指针
polymorphism	多型	多态
pop up	冒起式、弹出式	弹出式
port	埠	
precedence	优先序 (通常用於运算符的优先执行次序)	
preprocessor	前处理器	预处理器
primitive type	基本型别 (不同於 base class)	
print	列印	打印
printer	印表机	打印机
priority	优先权 (通常用於执行绪获	

	得 CPU 时间的优先次序)	
procedure	程序	过程
procedural	程序性的、程序式的	过程式的
process	行程	进程
profile	评测	评测
profiler	效能评测器	效能评测器
programmer	程式员	程序员
programming	编程、程式设计、程式化	编程
progress bar	进度指示器	进度指示器
project	专案	项目、工程
property	???	属性
protocol	协定	协议
pseudo code	假码、虚拟码、伪码	伪码
qualified	经过资格修饰 (例如加上 scope 运算符)	限定 ?
qualifier	资格修饰词、饰词	限定修饰词 ?
quality	品质	质量
queue	伫列	队列
radio button	圆钮	单选按钮
raise	引发 (常用来表示发出一个 exception)	引起、引发
random number	随机数、乱数	随机数
range	范围、区间 (用於 STL 时)	范围、区间
rank	等级、分等 (ref. C++Primer 3/e 9, 15 章)	等级
raw	生鲜的、未经处理的	未经处理的
record	记录	记录
recordset	记录集	记录集
recursive	递归	递归
re-direction	重导向	重定向
refactoring	重构、重整	重构
refer	取用	参考
reference	(C++ 中类似指标的东西, 相当於 "化身") 址叁器, see pointer	引用、参考

register	暂存器	寄存器
relational database	关联式资料库	关系数据库
represent	表述, 表现	表述, 表现
resolve	决议 (为算式中的符号名称寻找对应之宣告式的过程)	解析
resolution	决议程序、决议过程	解析过程
return	传回、回返	返回
return type	回返型别	返回类型
return value	回返回值	返回值
robust	强固、稳健	健壮
robustness	强固性、稳健性	健壮性
routine	常式	例程
runtime	执行期	执行期、执行时
rvalue	右值	右值
save	储存	存储
schedule	排程	调度
scheduler	排程器	调度程序
scroll bar	卷轴	滚动条
scope	生存空间、生存范围、范畴	生存空间
scope operator	生存空间 (范围决议) 运算符 ::	生存空间运算符
scope resolution operator	生存空间决议运算符 (与 scope operator 同)	生存空间解析运算符
search	搜寻	查找
semantics	语意	语义
sequential container	序列式容器 (对应於 associative container)	顺序式容器
server	伺服器、伺服端	服务器、服务端
serialization (serialize)	次第读写	序列化
signature	标记式	
slider	滚轴	滑块
slot	条孔、槽	槽
smart pointer	灵巧指标、机灵指标	智能指针
specialization	特殊化、特殊化定义、特殊化宣告	特化

splitter	分裂视窗	切分窗口
software	软体	软件
source	原始码	源码、源代码
stack	堆叠	栈
stack unwinding	堆叠辗转开解 (此词用於 exception 主题)	栈辗转开解 *
standard library	标准程式库	
standard template library	标准模板程式库	
statement	述句	语句、声明
status bar	状态列、状态栏	状态栏
STL	见 standard template library	
stream	资料流、串流	流
string	字串	字符串
subscript operator	下标运算符 []	下标运算符
subtype	子型别	子类型
support	支援	支持
syntax	语法	语法
target	标的 (例 target pointer: 标的指标)	目标
task switch	工作切换	任务切换
template	模板、范本	模板
template argument deduction	模板引数推导	模板参数推导
template explicit specialization	模板显式特化 (版本)	模板显式特化 ?
template parameter	模板参数	模板参数
text file	程式本文档 (放置程式原始码的档案)	文本文件
thread	执行绪	线程
throw	丢掷 (常指发出一个 exception)	丢掷、引发
token	语汇单元	符号、标记
transaction	交易	事务
trigger	触发	触发
type	型别	类型

UML unified modeling language	统一建模语言	
unary operator	一元运算符	一元运算符
underflow	下 限 溢 位 (相 对 於 overflow)	下溢
unqualified	未经资格修饰(而直接取用)	?
unwinding	ref. stack unwinding	?
variable	变数 (相对於常数 const)	变量
vector	向量 (一种容器, 有点类似 array)	向量
viable	可实行的、可行的	可行的
viable function	可行函式 (从 candidate functions 中挑出者)	可行函数
view(document/view)		视图
virtual function	虚拟函式	虚函数
volatile	易挥发的、易变的	?
window	视窗	窗口
window function	视窗函式	窗口函数
window procedure	视窗函式	窗口过程
word	字	词
wrapper	外覆、外包	
xxx based	以 xxx 为基础	基於 xxx
xxx box	xxx 盒、xxx 方块、框	xxx 框
例如 dialog box	对话框、对话方块、对话框	对话框
xxx oriented	xxx 导向	面向 xxx