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_VCETER，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。对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。不过我建议您避免使用 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 DRAWTEXTPARAMS 结构的指标，它的定义如下：

```
typedef struct tagDRAWTEXTPARAMS
{
      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
} DRAWTEXTPARAMS, * LPDRAWTEXTPARAMS ;
```

中间的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 SerTextAlign 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著色，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著色。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著色，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 的值小於 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 DEFULT_GUI_FONT。

当您将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。

字体型态

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。但对於 TrueType 字体，更倾向於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於或小於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，它用於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。我将这种用於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，只有当应用程式呼叫 SelectObject 将它选入装置内容时，它才成为真实的物件。例如，对於逻辑画笔，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 中有许多用於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於定义 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徵。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徵最匹配的字体。由於这依赖於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 (((LPCREATESTRUCT) 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) ;

     SetDlgItem  Text (hdlg, IDC_TM_VARIABLE,
                 TMPF_FIXED_PITCH & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

     SetDlgItem  Text (hdlg, IDC_TM_VECTOR,
                 TMPF_VECTOR & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

     SetDlgItem  Text (hdlg, IDC_TM_TRUETYPE,
                 TMPF_TRUETYPE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

     SetDlgItem  Text (hdlg, IDC_TM_DEVICE,
                 TMPF_DEVICE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

     SetDlgItem  Text (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。

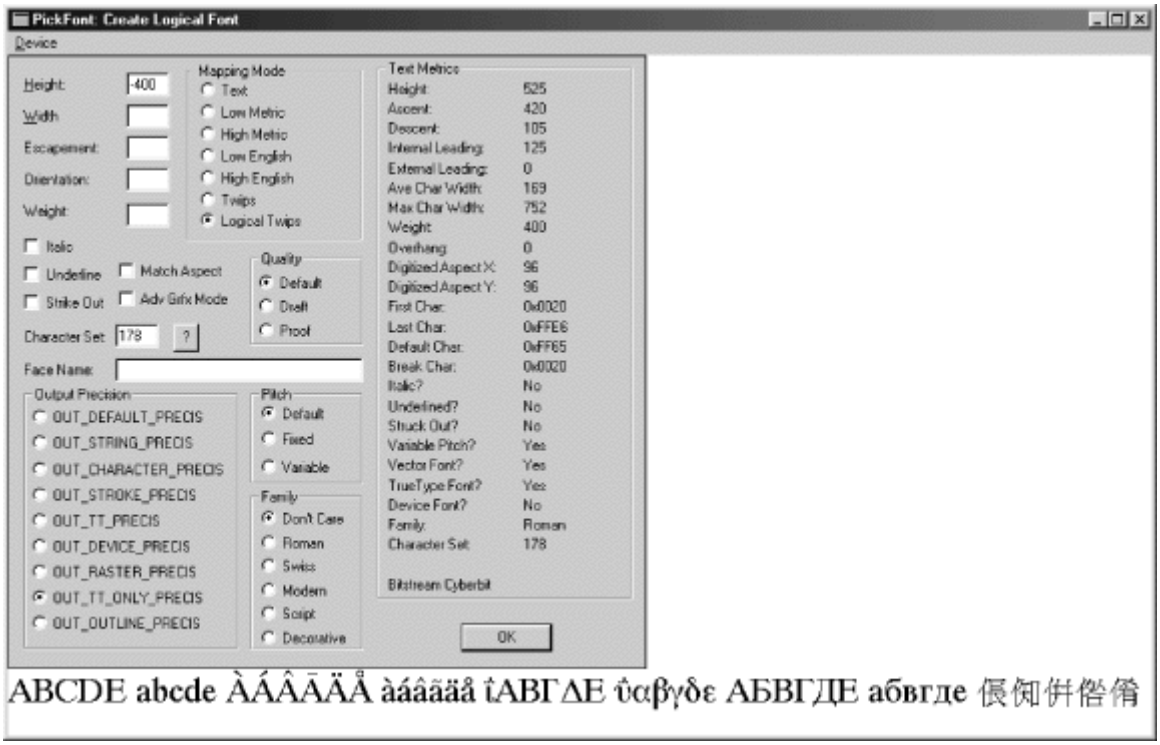

图 17-1 典型的 PICKFONT 萤幕显示（Windows NT 下的 Unicode 版本）

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 = CreatFontIndirect (&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，但对於 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 TureType 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。对於 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。零值相当於 ANSI_CHARSET，ANSI 字元在美国和西欧使用。DEFAULT_CHARSET 代码等於 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。请查看关於 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於剪裁区以外时，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，有关於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号结合用於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 这是关於字样（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後，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。
- 对於点阵字体，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於或等於所需的字体，除非没有更小的字体满足您的要求。对於笔划或 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。
- 如果指明了对於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，对於 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後从 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。它近似等於 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 栏位应近似等於 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号占据。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。对於定宽字体，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。对於粗体，tmAveCharWidth 值小於 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 字体中最後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，那么这个值可能大於 255。

tmDefaultChar Windows 用於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 和您的程式用於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 对於斜体字为非零值。

tmUnderlined 对於底线字体为非零值。

tmStruckOut 对於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）对於 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，也可以不带 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——将依赖於所选择的字元集。对於 ANSI_CHARSET，这个字元代码对应於大写和小写字母 A 的加重音版本。对於 GREEK_CHARSET，这些代码对应於希腊字母表的字母。对於 RUSSIAN_CHARSET，对应於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。由於它们在 Unicode 中有定义，这些代码总是对应於希腊字母表的字母。同样地，程式显示 0x0410 到 0x0415 以及 0x0430 到 0x0435 的字元代码，它们对应於斯拉夫字母表的字母。然而，这些字元不可能存在於内定字体中，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，它们对应於汉语、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）。

本章的後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，且这些字体可以用於显示器以及印表机，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点。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，因为围绕原点旋转座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 FONTROT（「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 FONTROT.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　FONTROT

```
FONTROT.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))) ;
      }
}
```

FONTROT 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。

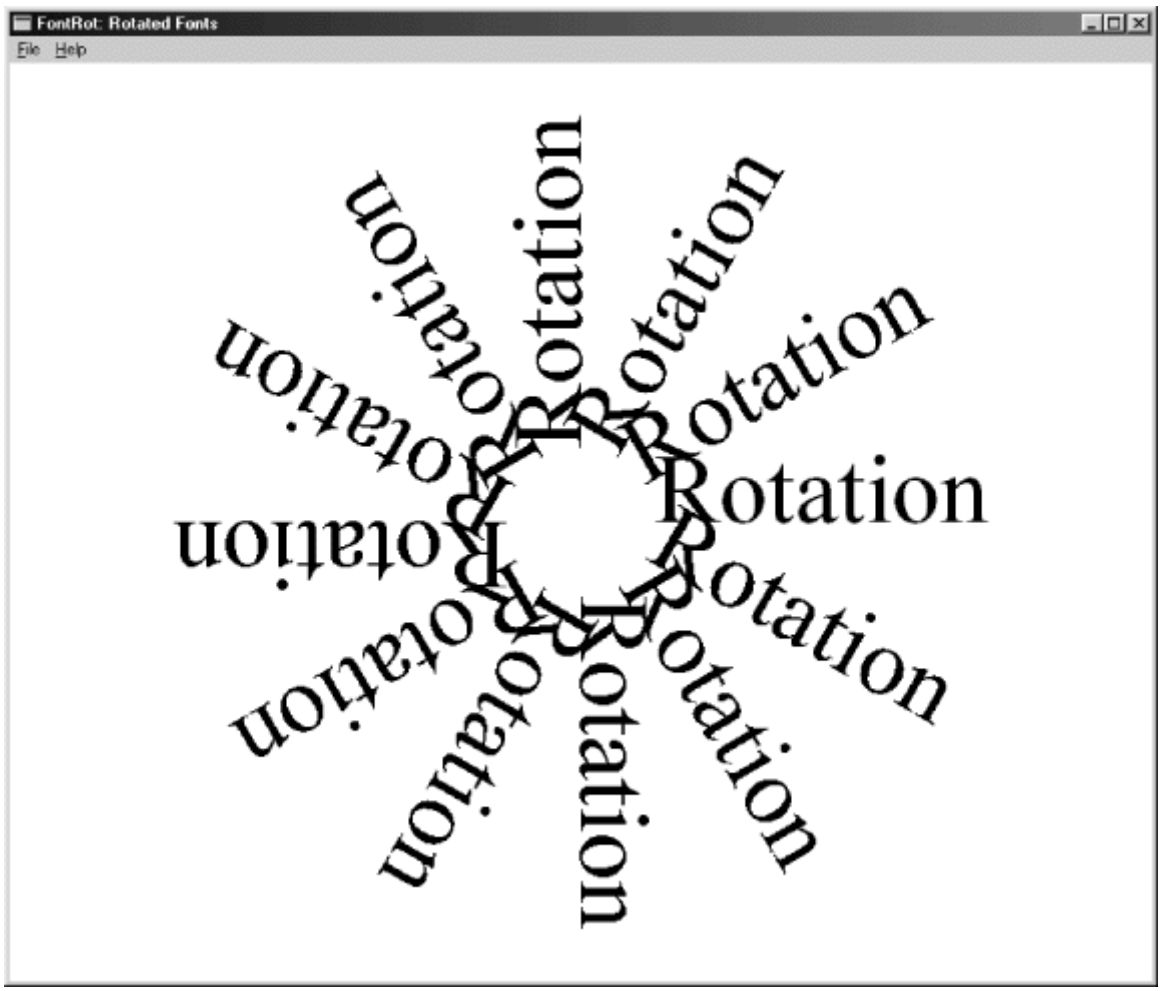

图 17-2 FONTROT 的萤幕显示

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序。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型态的旗标传递给它。

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 EnumProccallback 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。对於非 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，对於 TrueType 字体传递 NEWTEXTMETRIC 结构。NEWTEXTMETRIC 结构相对於 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              if ;
     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.lpfnHook      = 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_SCRIPTSONLY）。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於 lfHeight 栏位。在 EZFONT.C 中能找到合适的程式码，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，对於研究 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_INITTOLOGFONTSTRUCT | 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。对於前三种方式，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 对於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，然後写入文字：

```
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 ;
```

现在开始艰钜的任务——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。这将影响後面的 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串来寻找空格字元。每当碰到一个空格（或其他能用於断开一行的字元），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 ;
     DPtoLP (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_INITTOLOGFONTSTRUCT | CF_SCREENFONTS | CF_EFFECTS ;
                         cf.rgbColors                = 0 ;
                         cf.lCustData                = 0 ;
                         cf.lpfnHook                 = 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，当每行少於两个字时，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。

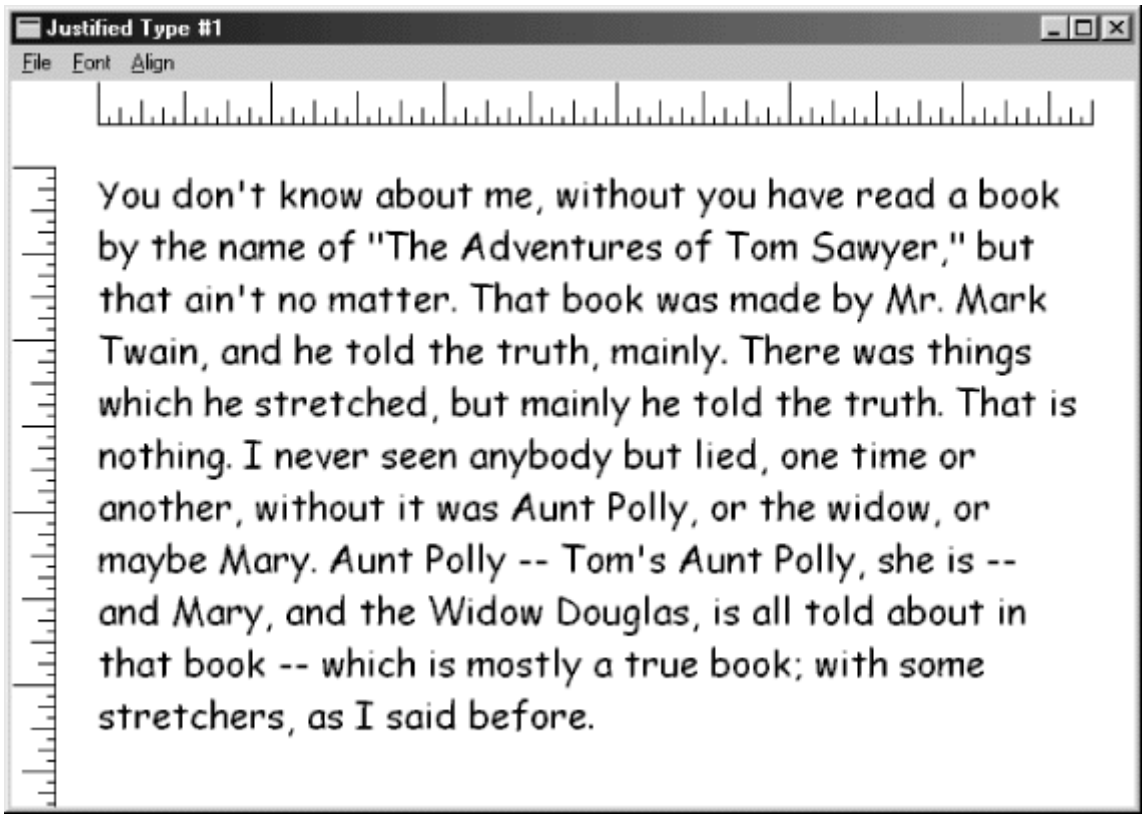

图 17-3 典型的 JUSTIFY1 萤幕显示

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, -36   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.otmEMSquare ;
}

/*-------------------------------------------------------------------------
     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                  uEMSquare ;

                     // Call function above

     uEMSquare = 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) uEMSquare ;
     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.lpfnHook           = 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 otmEMSquare 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 otmEMSquare 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：对於大多数 TrueType 字体，otmEMSquare 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 otmEMSquare 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 otmEMSquare 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。但随後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。

对於 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 画笔内部著色。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。

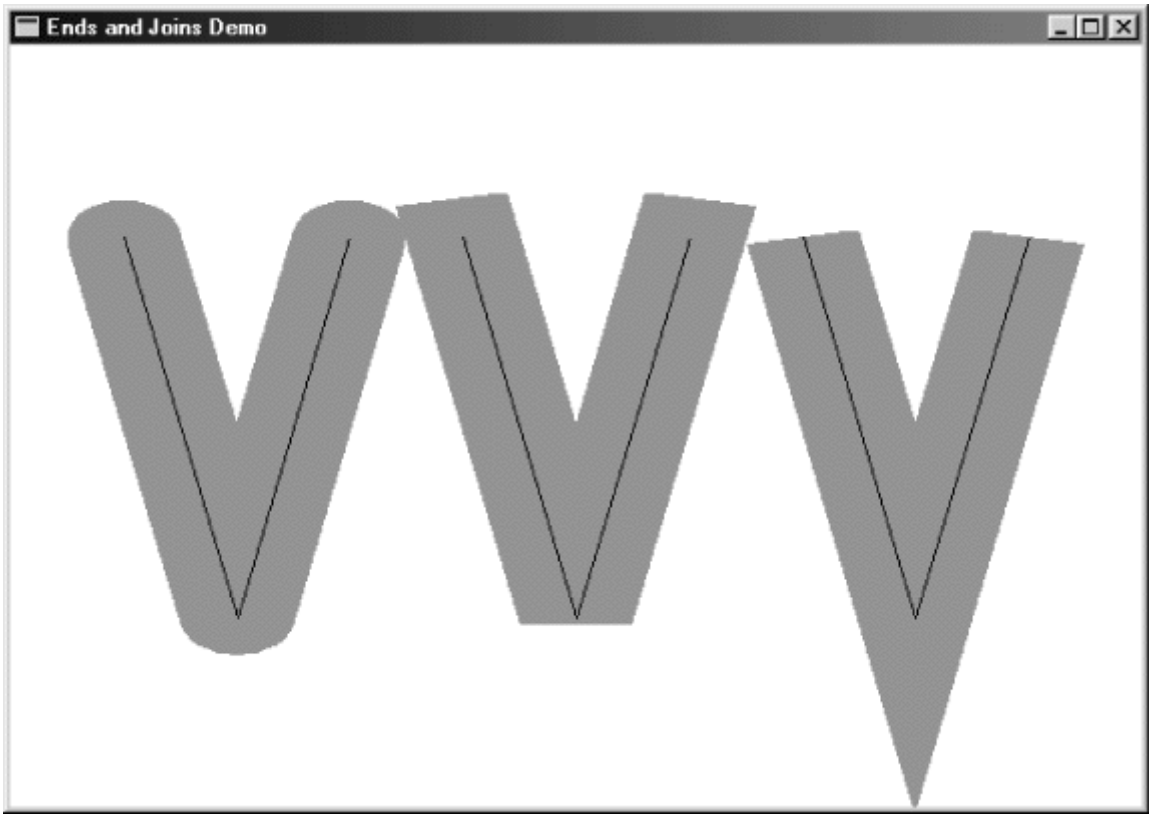

图 17-4 ENDJOIN 的萤幕显示

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

区域视窗中处於中心的位置。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後，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。

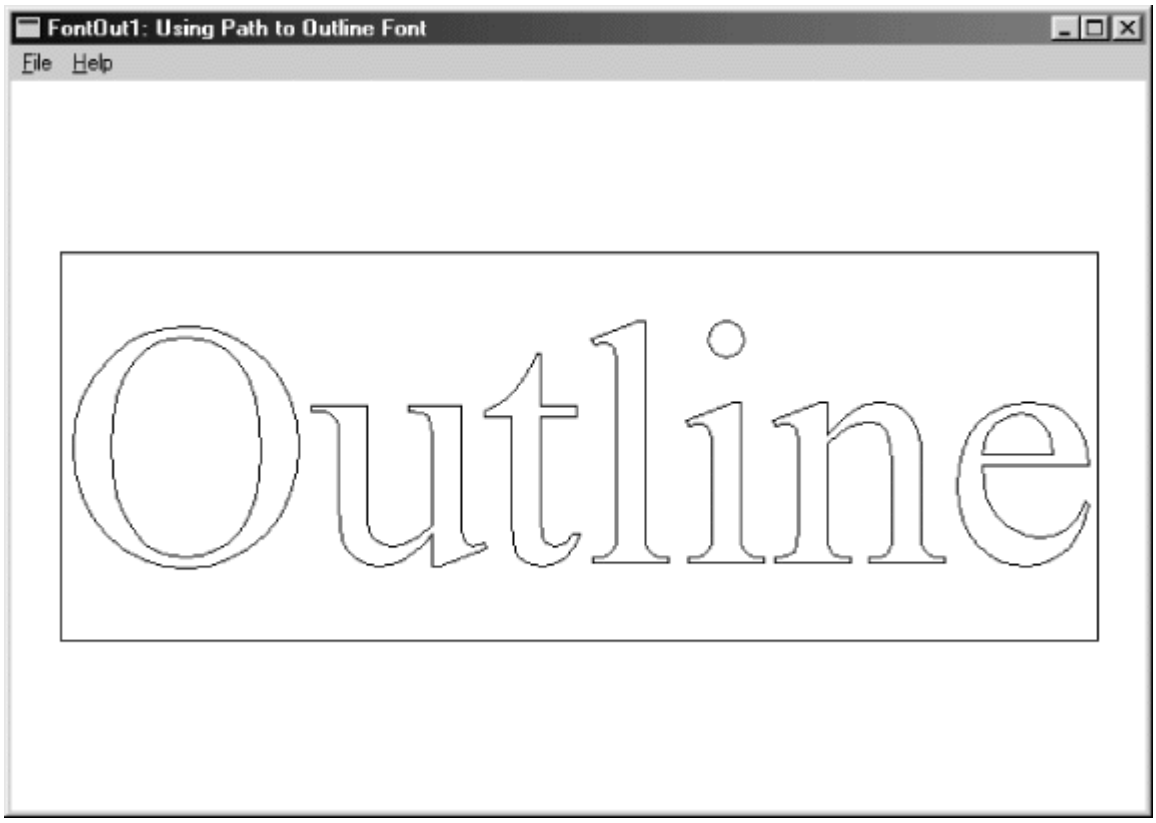

图 17-5 FONTOUT1 的萤幕显示

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。

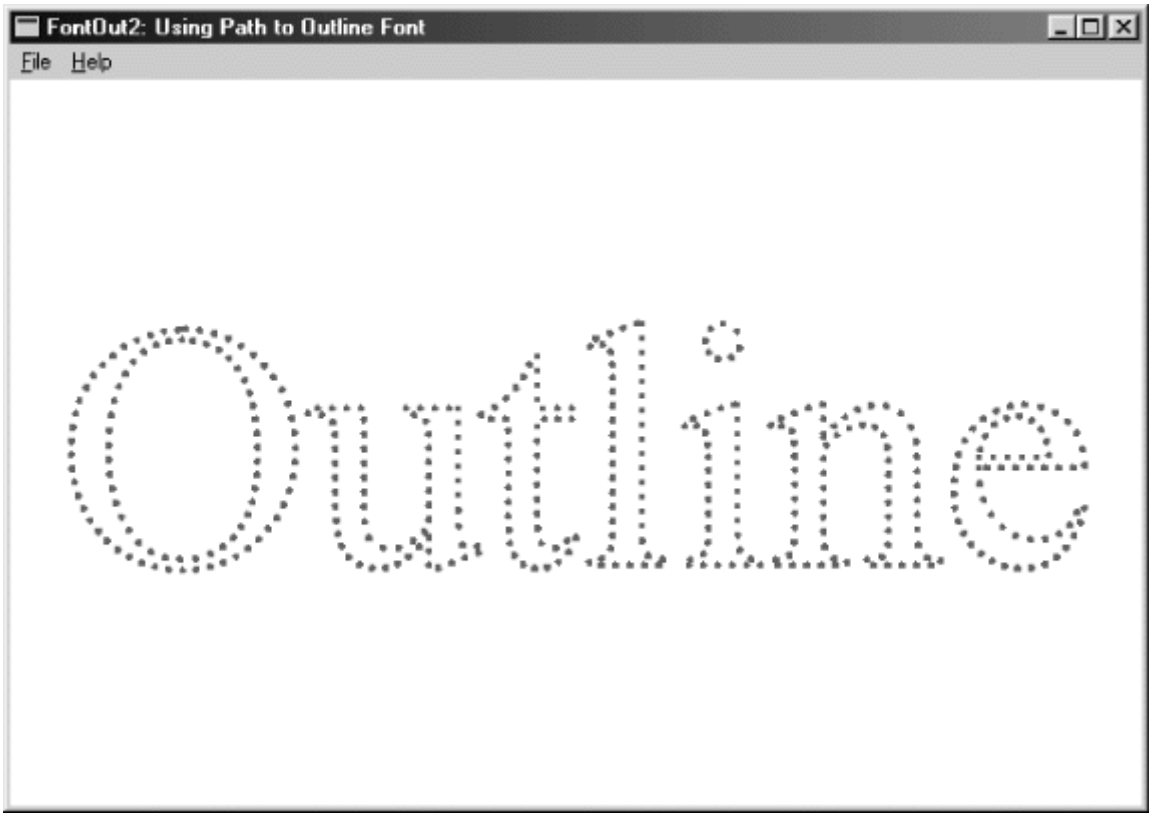

图 17-6 FONTOUT2 的萤幕显示

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 「O」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。

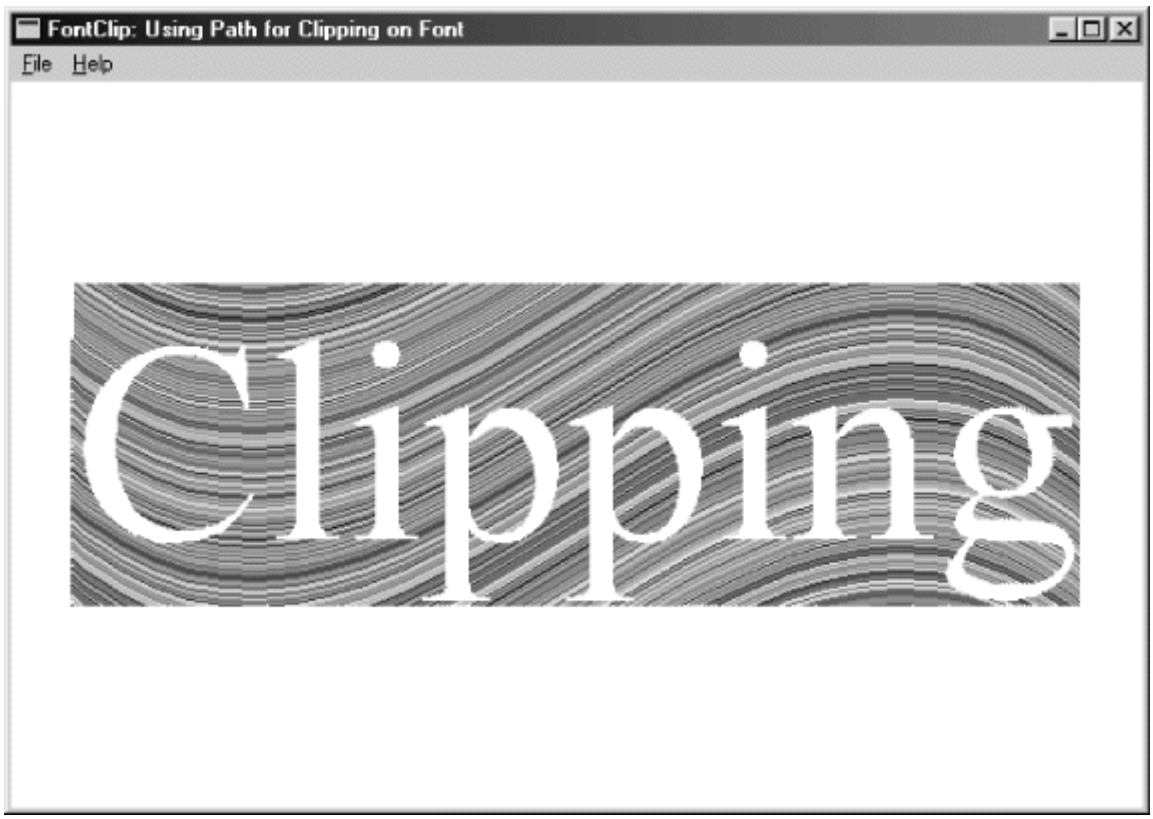

图 17-8 FONTCLIP 得萤幕显示

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。