第十五章　与装置无关的点阵图

在上一章我们了解到 Windows GDI 点阵图物件（也称为与装置相关的点阵图，或 DDB）有许多程式设计用途。但是我并没有展示把这些点阵图储存到磁片档案或把它们载入记忆体的方法。这是以前在 Windows 中使用的方法，现在根本不用了。因为点阵图的位元格式相当依赖於设备，所以 DDB 不适用於图像交换。DDB 内没有色彩对照表来指定点阵图的位与色彩之间的联系。DDB 只有在 Windows 开机到关机的生命期内被建立和清除时才有意义。

在 Windows 3.0 中发表了与装置无关的点阵图(DIB)，提供了适用於交换的图像档案格式。正如您所知的，像.GIF 或.JPEG 之类的其他图像档案格式在 Internet 上比 DIB 档案更常见。这主要是因为.GIF 和.JPEG 格式进行了压缩，明显地减少了下载的时间。尽管有一个用於 DIB 的压缩方案，但极少使用。DIB 内的点阵图几乎都没有被压缩。如果您想在程式中操作点阵图，这实际上是一个优点。DIB 不像.GIF 和.JPEG 档案，Windows API 直接支援 DIB。如果在记忆体中有 DIB，您就可以提供指向该 DIB 的指标作为某些函式的参数，来显示 DIB 或把 DIB 转化为 DDB。

DIB 档案格式

有意思的是，DIB 格式并不是源自於 Windows。它首先定义在 OS/2 的 1.1 版中，该作业系统最初由 IBM 和 Microsoft 在八十年代中期开始开发。OS/2 1.1 在 1988 年发布，并且是第一个包含了类似 Windows 的图形使用者介面的 OS/2 版本，该图形使用者介面被称之为「Presentation Manager（PM）」。「Presentation Manager」包含了定义点阵图格式的「图形程式介面」（GPI）。

然後在 Windows 3.0 中（发布於 1990）使用了 OS/2 点阵图格式，这时称之为 DIB。Windows 3.0 也包含了原始 DIB 格式的变体，并在 Windows 下成为标准。在 Windows 95（以及 Windows NT 4.0）和 Windows 98（以及 Windows NT 5.0）下也定义了一些其他的增强能力，我会在本章讨论它们。

DIB 首先作为一种档案格式，它的副档名为.BMP，在极少情况下为.DIB。Windows 应用程式使用的点阵图图像被当做 DIB 档案建立，并作为唯读资源储存在程式的可执行档案中。图示和滑鼠游标也是形式稍有不同的 DIB 档案。

程式能将 DIB 档案减去前 14 个位元组载入连续的记忆体块中。这时就可以称它为「packed DIB（packed-DIB）格式的点阵图」。在 Windows 下执行的应用程式能使用 packed DIB 格式，通过 Windows 剪贴簿来交换图像或建立画刷。

程式也可以完全存取 DIB 的内容并以任意方式修改 DIB。

程式也能在记忆体中建立自己的 DIB 然後把它们存入档案。程式使用 GDI 函式呼叫就能「绘制」这些 DIB 内的图像，也能在程序中利用别的记忆体 DIB 直接设定和操作图素位元。

在记忆体中载入了 DIB 後，程式也能通过几个 Windows API 函式呼叫来使用 DIB 资料，我将在本章中讨论有关内容。与 DIB 相关的 API 呼叫是很少的，并且主要与视讯显示器或印表机页面上显示 DIB 相关，还与转换 GDI 点阵图物件有关。

除了这些内容以外，还有许多应用程式需要完成的 DIB 任务，而这些任务 Windows 作业系统并不支援。例如，程式可能存取了 24 位元 DIB 并且想把它转化为带有最佳化的 256 色调色盘的 8 位元 DIB，而 Windows 不会为您执行这些操作。但是在本章和下一章将向您显示 Windows API 之外的操作 DIB 的方式。

OS/2 样式的 DIB

先不要陷入太多的细节，让我们看一下与首先在 OS/2 1.1 中出现的点阵图格式相容的 Windows DIB 格式。

DIB 档案有四个主要部分：

- 档案表头
- 资讯表头
- RGB 色彩对照表（不一定有）
- 点阵图图素位元

您可以把前两部分看成是 C 的资料结构，把第三部分看成是资料结构的阵列。在 Windows 表头档案 WINGDI.H 中说明了这些结构。在记忆体中的 packed DIB 格式内有三个部分：

- 资讯表头
- RGB 色彩对照表（不一定有）
- 点阵图图素位元

除了没有档案表头外，其他部分与储存在档案内的 DIB 相同。

DIB 档案（不是记忆体中的 packed DIB）以定义为如下结构的 14 个位元组的档案表头开始：

```
typedef struct tagBITMAPFILEHEADER  // bmfh
{
      WORD        bfType ;       // signature word "BM" or 0x4D42
      DWORD             bfSize ;       // entire size of file
      WORD        bfReserved1 ;   // must be zero
      WORD        bfReserved2 ;   // must be zero
```

```
      DWORD            bfOffsetBits ;  // offset in file of DIB pixel bits
}
BITMAPFILEHEADER, * PBITMAPFILEHEADER ;
```

在 WINGDI.H 内定义的结构可能与这不完全相同，但在功能上是相同的。第一个注释（就是文字「bmfh」）指出了给这种资料型态的资料变数命名时推荐的缩写。如果在我的程式内看到了名为 pbmfh 的变数，这可能是一个指向 BITMAPFILEHEADER 型态结构的指标或指向 PBITMAPFILEHEADER 型态变数的指标。

结构的长度为 14 位元组，它以两个字母「BM」开头以指明是点阵图档案。这是一个 WORD 值 0x4D42。紧跟在「BM」後的 DWORD 以位元组为单位指出了包括档案表头在内的档案大小。下两个 WORD 栏位设定为 0。（在与 DIB 档案格式相似的滑鼠游标档案内，这两个栏位指出游标的「热点（hot spot）」）。结构还包含一个 DWORD 栏位，它指出了档案中图素位元开始位置的位元组偏移量。此数值来自 DIB 资讯表头中的资讯，为了使用的方便提供在这里。

在 OS/2 样式的 DIB 内，BITMAPFILEHEADER 结构後紧跟了 BITMAPCOREHEADER 结构，它提供了关於 DIB 图像的基本资讯。紧缩的 DIB（Packed DIB）开始於 BITMAPCOREHEADER：

```
typedef struct tagBITMAPCOREHEADER  // bmch
{
      DWORD            bcSize ;               // size of the structure = 12
      WORD       bcWidth ;              // width of image in pixels
      WORD       bcHeight ;             // height of image in pixels
      WORD       bcPlanes ;             // = 1
      WORD       bcBitCount ;           // bits per pixel (1, 4, 8, or 24)
}
BITMAPCOREHEADER, * PBITMAPCOREHEADER ;
```

「core（核心）」用在这里看起来有点奇特，它是指这种格式是其他由它所衍生的点阵图格式的基础。

BITMAPCOREHEADER 结构中的 bcSize 栏位指出了资料结构的大小，在这种情况下是 12 位元组。

bcWidth 和 bcHeight 栏位包含了以图素为单位的点阵图大小。尽管这些栏位使用 WORD 意味著一个 DIB 可能为 65,535 图素高和宽，但是我们几乎不会用到那么大的单位。

bcPlanes 栏位的值始终是 1。这个栏位是我们在上一章中遇到的早期 Windows GDI 点阵图物件的残留物。

bcBitCount 栏位指出了每图素的位元数。对於 OS/2 样式的 DIB，这可能是 1、4、8 或 24。DIB 图像中的颜色数等於 2bmch.bcBitCount，或用 C 的语法表示为：

```
1 << bmch.bcBitCount
```

这样，bcBitCount 栏位等於：

- 1 代表 2 色 DIB
- 4 代表 16 色 DIB
- 8 代表 256 色 DIB
- 24 代表 full -Color DIB

当我提到「8 位元 DIB」时，就是说每图素占 8 位元的 DIB。

对於前三种情况（也就是位元数为 1、4 和 8 时），BITMAPCOREHEADER 後紧跟色彩对照表，24 位元 DIB 没有色彩对照表。色彩对照表是一个 3 位元组 RGBTRIPLE 结构的阵列，阵列中的每个元素代表图像中的每种颜色：

```
typedef struct tagRGBTRIPLE  // rgbt
{
      BYTE rgbtBlue ;        // blue level
      BYTE rgbtGreen ;       // green level
      BYTE rgbtRed ;         // red level
}
RGBTRIPLE ;
```

这样排列色彩对照表以便 DIB 中最重要的颜色首先显示，我们将在下一章说明原因。

WINGDI.H 表头档案也定义了下面的结构：

```
typedef struct tagBITMAPCOREINFO  // bmci
{
      BITMAPCOREHEADER bmciHeader ;          // core-header structure
      RGBTRIPLE        bmciColors[1] ;       // color table array
}
BITMAPCOREINFO, * PBITMAPCOREINFO ;
```

这个结构把资讯表头与色彩对照表结合起来。虽然在这个结构中 RGBTRIPLE 结构的数量等於 1，但在 DIB 档案内您绝对不会发现只有一个 RGBTRIPLE。根据每个图素的位元数，色彩对照表的大小始终是 2、16 或 256 个 RGBTRIPLE 结构。如果需要为 8 位元 DIB 配置 PBITMAPCOREINFO 结构，您可以这样做：

```
pbmci = malloc (sizeof (BITMAPCOREINFO) + 255 * sizeof (RGBTRIPLE)) ;
```

然後可以这样存取 RGBTRIPLE 结构：

```
pbmci->bmciColors[i]
```

因为 RGBTRIPLE 结构的长度是 3 位元组，许多 RGBTRIPLE 结构可能在 DIB 中以奇数位址开始。然而，因为在 DIB 档案内始终有偶数个的 RGBTRIPLE 结构，所以紧跟在色彩对照表阵列後的资料块总是以 WORD 位址边界开始。

紧跟在色彩对照表（24 位元 DIB 中是资讯表头）後的资料是图素位元本身。

由下而上

像大多数点阵图格式一样，DIB 中的图素位元是以水平行组织的，用视讯显示器硬体的术语称作「扫描线」。行数等於 BITMAPCOREHEADER 结构的 bcHeight 栏位。然而，与大多数点阵图格式不同的是，DIB 从图像的底行开始，往上表示图像。

在此应定义一些术语，当我们说「顶行」和「底行」时，指的是当其正确显示在显示器或印表机的页面上时出现在虚拟图像的顶部和底部。就好像肖像的顶行是头发，底行是下巴，在 DIB 档案中的「第一行」指的是 DIB 档案的色彩对照表後的图素行，「最後行」指的是档案最末端的图素行。

因此，在 DIB 中，图像的底行是档案的第一行，图像的顶行是档案的最後一行。这称之为由下而上的组织。因为这种组织和直觉相反，您可能会问：为什么要这么做？

好，现在我们回到 OS/2 的 Presentation Manager。IBM 的人认为 PM 内的座标系统——包括视窗、图形和点阵图——应该是一致的。这引起了争论：大多数人，包括在全画面文字方式下编程和视窗环境下工作的程式写作者认为应使用垂直座标在萤幕上向下增加的座标。然而，电脑图形程式写作者认为应使用解析几何的数学方法进行视讯显示，这是一个垂直座标在空间中向上增加的直角（或笛卡尔）座标系。

简而言之，数学方法赢了。PM 内的所有事物都以左下角为原点（包括视窗座标），因此 DIB 也就有了那种方式。

DIB 图素位元

DIB 档案的最後部分（在大多数情况下是 DIB 档案的主体）由实际的 DIB 的图素位元组成。图素位元是由从图像的底行开始并沿著图像向上增长的水平行组织的。

DIB 中的行数等於 BITMAPCOREHEADER 结构的 bcHeight 栏位。每一行的图素数等於该结构的 bcWidth 栏位。每一行从最左边的图素开始，直到图像的右边。每个图素的位元数可以从 bcBitCount 栏位取得，为 1、4、8 或 24。

以位元组为单位的每行长度始终是 4 的倍数。行的长度可以计算为：

```
RowLength = 4 * ((bmch.bcWidth * bmch.bcBitCount + 31) / 32) ;
```

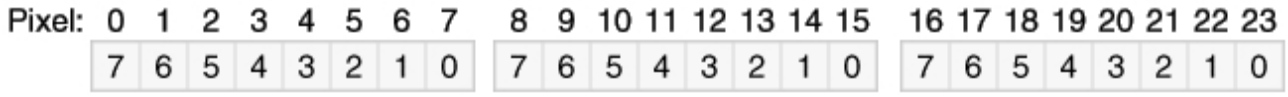
或者在 C 内用更有效的方法：

```
RowLength = ((bmch.bcWidth * bmch.bcBitCount + 31) & ~31) >> 3 ;
```

如果需要，可通过在右边补充行（通常是用零）来完成长度。图素资料的总位元组数等於 RowLength 和 bmch.bcHeight 的乘积。

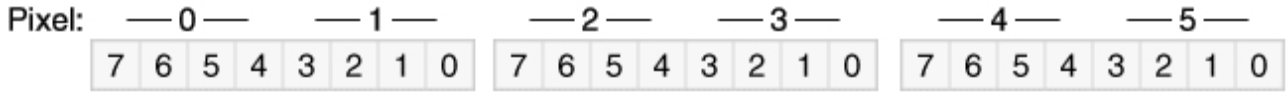
要了解图素编码的方式，让我们分别考虑四种情况。在下面的图表中，每个位元组的位元显示在框内并且编了号，7 表示最高位元，0 表示最低位元。图素也从行的最左端从 0 开始编号。

对於每图素 1 位元的 DIB，每位元组对应为 8 图素。最左边的图素是第一个位元组的最高位元：



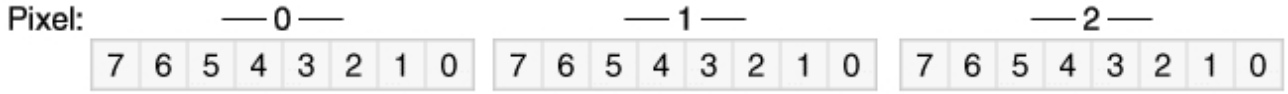
每个图素可以是 0 或 1。0 表示该图素的颜色由色彩对照表中第一个 RGBTRIPLE 项目给出。1 表示图素的颜色由色彩对照表的第二个项目给出。

对於每图素 4 位元的 DIB，每个位元组对应两个图素。最左边的图素是第一个位元组的高 4 位元，以此类推：



每图素 4 位元的值的范围从 0 到 15。此值是指向色彩对照表中 16 个项目的索引。

对於每图素 8 位元的 DIB，每个位元组为 1 个图素：



位元组的值从 0 到 255。同样，这也是指向色彩对照表中 256 个项目的索引。

对於每图素 24 位元的 DIB，每个图素需要 3 个位元组来代表红、绿和蓝的颜色值。图素位元的每一行，基本上就是 RGBTRIPLE 结构的阵列，可能需要在每行的末端补 0 以便该行为 4 位元组的倍数：

每图素 24 位元的 DIB 没有色彩对照表。

扩展的 Windows DIB

现在我们掌握了 Windows 3.0 中介绍的与 OS/2 相容的 DIB，同时也看一看 Windows 中 DIB 的扩展版本。

这种 DIB 形式跟前面的格式一样，以 BITMAPFILEHEADER 结构开始，但是接著是 BITMAPINFOHEADER 结构，而不是 BITMAPCOREHEADER 结构：

```
typedef struct tagBITMAPINFOHEADER  // bmih
{
      DWORD       biSize ;               // size of the structure = 40
      LONG  biWidth ;              // width of the image in pixels
      LONG  biHeight ;             // height of the image in pixels
```

```
      WORD  biPlanes ;             // = 1
      WORD  biBitCount ;           // bits per pixel (1, 4, 8, 16, 24, or 32)
      DWORD       biCompression ;  // compression code
      DWORD       biSizeImage ;          // number of bytes in image
      LONG  biXPelsPerMeter ;      // horizontal resolution
      LONG  biYPelsPerMeter ;      // vertical resolution
      DWORD       biClrUsed ;            // number of colors used
      DWORD       biClrImportant ;       // number of important colors
}
BITMAPINFOHEADER, * PBITMAPINFOHEADER ;
```

您可以通过检查结构的第一栏位区分与 OS/2 相容的 DIB 和 Windows DIB，前者为 12，後者为 40。

您将注意到，在这个结构内有六个附加的栏位，但是 BITMAPINFOHEADER 不是简单地由 BITMAPCOREHEADER 加上一些新栏位而成。仔细看一下：在 BITMAPCOREHEADER 结构中，bcWidth 和 bcHeight 栏位是 16 位元 WORD 值；而在 BITMAPINFOHEADER 结构中它们是 32 位元 LONG 值。这是一个令人讨厌的小变化，当心它会给您带来麻烦。

另一个变化是：对於使用 BITMAPINFOHEADER 结构的 1 位元、4 位元和 8 位元 DIB，色彩对照表不是 RGBTRIPLE 结构的阵列。相反，BITMAPINFOHEADER 结构紧跟著一个 RGBQUAD 结构的阵列：

```
typedef struct tagRGBQUAD  // rgb
{
      BYTE rgbBlue ;     // blue level
      BYTE rgbGreen ;    // green level
      BYTE rgbRed ;      // red level
      BYTE rgbReserved ; // = 0
}
RGBQUAD ;
```

除了包括总是设定为 0 的第四个栏位外，与 RGBTRIPLE 结构相同。WINGDI.H 表头档案也定义了以下结构：

```
typedef struct tagBITMAPINFO            // bmi
{
      BITMAPINFOHEADER bmiHeader ;     // info-header structure
      RGBQUAD                bmiColors[1] ;  // color table array
}
BITMAPINFO, * PBITMAPINFO ;
```

注意，如果 BITMAPINFO 结构以 32 位元的位址边界开始，因为 BITMAPINFOHEADER 结构的长度是 40 位元组，所以 RGBQUAD 阵列内的每一个项目也以 32 位边界开始。这样就确保通过 32 位元微处理器能更有效地对色彩对照表资料定址。

尽管 BITMAPINFOHEADER 最初是在 Windows 3.0 中定义的，但是许多栏位在

Windows 95 和 Windows NT 4.0 中又重新定义了，并且被带入 Windows 98 和 Windows NT 5.0 中。比如现在的文件中说：「如果 biHeight 是负数，则点阵图是由上而下的 DIB，原点在左上角」。这很好，但是在 1990 年刚开始定义 DIB 格式时，如果有人做了这个决定，那会更好。我的建议是避免建立由上而下的 DIB。有一些程式在编写时没有考虑这种新「特性」，在遇到负的 biHeight 栏位时会当掉。还有如 Microsoft Word 97 带有的 Microsoft Photo Editor 在遇到由上而下的 DIB 时会报告「图像高度不合法」（虽然 Word 97 本身不会出错）。

biPlanes 栏位始终是 1，但 biBitCount 栏位现在可以是 16 或 32 以及 1、4、8 或 24。这也是在 Windows 95 和 Windows NT 4.0 中的新特性。一会儿我将介绍这些附加格式工作的方式。

现在让我们先跳过 biCompression 和 biSizeImage 栏位，一会儿再讨论它们。

biXPelsPerMeter 和 biYPelsPerMeter 栏位以每公尺多少图素这种笨拙的单位指出图像的实际尺寸。（「pel」--picture element（图像元素）--是 IBM 对图素的称呼。）Windows 在内部不使用此类资讯。然而，应用程式能够利用它以准确的大小显示 DIB。如果 DIB 来源於没有方图素的设备，这些栏位是很有用的。在大多数 DIB 内，这些栏位设定为 0，这表示没有建议的实际大小。每英寸 72 点的解析度（有时用於视讯显示器，尽管实际解析度依赖於显示器的大小）大约相当於每公尺 2835 个图素，300 DPI 的普通印表机的解析度是每公尺 11,811 个图素。

biClrUsed 是非常重要的栏位，因为它影响色彩对照表中项目的数量。对於 4 位元和 8 位元 DIB，它能分别指出色彩对照表中包含了小於 16 或 256 个项目。虽然并不常用，但这是一种缩小 DIB 大小的方法。例如，假设 DIB 图像仅包括 64 个灰阶，biClrUsed 栏位设定为 64，并且色彩对照表为 256 个位元组大小的色彩对照表包含了 64 个 RGBQUAD 结构。图素值的范围从 0x00 到 0x3F。DIB 仍然每图素需要 1 位元组，但每个图素位元组的高 2 位元为零。如果 biClrUsed 栏位设定为 0，意味著色彩对照表包含了由 biBitCount 栏位表示的全部项目数。

从 Windows 95 开始，biClrUsed 栏位对於 16 位元、24 位元或 32 位元 DIB 可以为非零。在这些情况下，Windows 不使用色彩对照表解释图素位元。相反地，它指出 DIB 中色彩对照表的大小，程式使用该资讯来设定调色盘在 256 色视讯显示器上显示 DIB。您可能想起在 OS/2 相容格式中，24 位元 DIB 没有色彩对照表。在 Windows 3.0 中的扩展格式中，也与这一样。而在 Windows 95 中，24 位元 DIB 有色彩对照表，biClrUsed 栏位指出了它的大小。

总结如下：

对於 1 位元 DIB，biClrUsed 始终是 0 或 2。色彩对照表始终有两个项目。

对於 4 位元 DIB，如果 biClrUsed 栏位是 0 或 16，则色彩对照表有 16 个项目。如果是从 2 到 15 的数，则指的是色彩对照表中的项目数。每个图素的最大值是小於该数的 1。

对於 8 位元 DIB，如果 biClrUsed 栏位是 0 或 256，则色彩对照表有 256 个项目。如果是从 2 到 225 的数，则指的是色彩对照表中的项目数。每个图素的最大值是小於该数的 1。

对於 16 位元、24 位元或 32 位元 DIB，biClrUsed 栏位通常为 0。如果它不为 0，则指的是色彩对照表中的项目数。执行於 256 色显示卡的应用程式能使用这些项目来为 DIB 设定调色盘。

另一个警告：原先使用早期 DIB 文件编写的程式不支援 24 位元 DIB 中的色彩对照表，如果在程式使用 24 位元 DIB 的色彩对照表的话，就要冒一定的风险。

biClrImportant 栏位实际上没有 biClrUsed 栏位重要，它通常被设定为 0 以指出色彩对照表中所有的颜色都是重要的，或者它与 biClrUsed 有相同的值。两种方法意味著同一件事，如果它被设定为 0 与 biClrUsed 之间的值，就意味著 DIB 图像能仅仅通过色彩对照表中第一个 biClrImportant 项目合理地取得。当在 256 色显示卡上并排显示两个或更多 8 位元 DIB 时，这是很有用的。

对於 1 位元、4 位元、8 位元和 24 位元的 DIB，图素位元的组织和 OS/2 相容的 DIB 是相同的，一会儿我将讨论 16 位元和 32 位元 DIB。

真实检查

当遇到一个由其他程式或别人建立的 DIB 时，您希望从中发现什么内容呢？

尽管在 Windows3.0 首次推出时，OS/2 样式的 DIB 已经很普遍了，但最近这种格式却已经很少出现了。许多程式写作者在实际编写快速 DIB 常式时忽略了它们。您遇到的任何 4 位元 DIB 可能是 Windows 的「小画家」程式使用 16 色视讯显示器建立的，在这些显示器上色彩对照表具有标准的 16 种颜色。

最普遍的 DIB 可能是每图素 8 位元。8 位元 DIB 分为两类：灰阶 DIB 和混色 DIB。不幸的是，表头资讯中并没有指出 8 位元 DIB 的型态。

许多灰阶 DIB 有一个等於 64 的 biClrUsed 栏位，指出色彩对照表中的 64 个项目。这些项目通常以上升的灰阶层排列，也就是说色彩对照表以 00-00-00、04-04-04、08-08-08、0C-0C-0C 的 RGB 值开始，并包括 F0-F0-F0、F4-F4-F4、F8-F8-F8 和 FC-FC-FC 的 RGB 值。此类色彩对照表可用下列公式计算：

```
rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 256 / 64 ;
```

在这里 rgb 是 RGBQUAD 结构的阵列，i 的范围从 0 到 63。灰阶色彩对照表

可用下列公式计算：

```
rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 255 / 63 ;
```

因而此表以 FF-FF-FF 结尾。

实际上使用哪个计算公式并没有什么区别。许多视讯显示卡和显示器没有比 6 位元更大的色彩精确度。第一个公式承认了这个事实。然而当产生小於 64 的灰阶时——可能是 16 或 32（在此情况下公式的除数分别是 15 和 31）——使用第二个公式更适合，因为它确保了色彩对照表的最後一个项目是 FF-FF-FF，也就是白色。

当某些 8 位元灰阶 DIB 在色彩对照表内有 64 个项目时，其他灰阶的 DIB 会有 256 个项目。biClrUsed 栏位实际上可以为 0（指出色彩对照表中有 256 个项目）或者从 2 到 256 的数。当然，biClrUsed 值是 2 的话就没有任何意义（因为这样的 8 位元 DIB 能当作 1 位元 DIB 被重新编码）或者小於或等於 16 的值也没意义（因为它能当作 4 位元 DIB 被重新编码）。任何情况下，色彩对照表中的项目数必须与 biClrUsed 栏位相同（如果 biClrUsed 是 0，则是 256），并且图素值不能超过色彩对照表项目数减 1 的值。这是因为图素值是指向色彩对照表阵列的索引。对於 biClrUsed 值为 64 的 8 位元 DIB，图素值的范围从 0x00 到 0x3F。

在这里应记住一件重要的事情：当 8 位元 DIB 具有由整个灰阶组成的色彩对照表（也就是说，当红色、绿色和蓝色程度相等时），或当这些灰阶层在色彩对照表中递增（像上面描述的那样）时，图素值自身就代表了灰色的程度。也就是说，如果 biClrUsed 是 64，那么 0x00 图素值为黑色，0x20 的图素值是 50%的灰阶，0x3F 的图素值为白色。

这对於一些图像处理作业是很重要的，因为您可以完全忽略色彩对照表，仅需处理图素值。这是很有用的，如果让我回溯时光去对 BITMAPINFOHEADER 结构做一个简单的更改，我会添加一个旗标指出 DIB 映射是不是灰阶的，如果是，DIB 就没有色彩对照表，并且图素值直接代表灰阶。

混色的 8 位元 DIB 一般使用整个色彩对照表，它的 biClrUsed 栏位为 0 或 256。然而您也可能遇到较小的颜色数，如 236。我们应承认一个事实：程式通常只能在 Windows 颜色面内更改 236 个项目以正确显示这些 DIB，我将在下章讨论这些内容。

biXPelsPerMeter 和 biYPelsPerMeter 很少为非零值，biClrImportant 栏位不为 0 或 biClrUsed 值的情况也很少。

DIB 压缩

前面我没有讨论 BITMAPINFOHEADER 中的 biCompression 和 biSizeImage 栏位，现在我们讨论一下这些值。

biCompression 栏位可以为四个常数之一，它们是：BI_RGB、BI_RLE8、BI_RLE4 或 BI_BITFIELDS。它们定义在 WINGDI.H 表头档案中，值分别为 0 到 3。此栏位有两个用途：对於 4 位元和 8 位元 DIB，它指出图素位元被用一种运行长度（run-length）编码方式压缩了。对於 16 位元和 32 位元 DIB，它指出了颜色遮罩(color masking)是否用於对图素位元进行编码。这两个特性都是在 Windows 95 中发表的。

首先让我们看一下 RLE 压缩：

- 对於 1 位元 DIB，biCompression 栏位始终是 BI_RGB。
- 对於 4 位元 DIB，biCompression 栏位可以是 BI_RGB 或 BI_RLE4。
- 对於 8 位元 DIB，biCompression 栏位可以是 BI_RGB 或 BI_RLE8。
- 对於 24 位元 DIB，biCompression 栏位始终是 BI_RGB。

如果值是 BI_RGB，图素位元储存的方式和 OS/2 相容的 DIB 一样，否则就使用运行长度编码压缩图素位元。

运行长度编码（RLE）是一种最简单的资料压缩形式，它是根据 DIB 映射在一列内经常有相同的图素字串这个事实进行的。RLE 通过对重复图素的值及重复的次数编码来节省空间，而用於 DIB 的 RLE 方案只定义了很少的矩形 DIB 图像，也就是说，矩形的某些区域是未定义的，这能被用於表示非矩形图像。

8 位元 DIB 的运行长度编码在概念上更简单一些，因此让我们从这里入手。表 15-1 会帮助您理解当 biCompression 栏位等於 BI_RGB8 时，图素位元的编码方式。

表 15-1

位元组 1	位元组 2	位元组 3	位元组 4	含义
00	00			行尾
00	01			映射尾
00	02	dx	dy	移到（x+dx，y+dy）
00	n = 03 到 FF			使用下面 n 个图素
n = 01 到 FF	图素			重复图素 n 次

当对压缩的 DIB 解码时，可成对查看 DIB 资料位元组，例如此表内的「位元组 1」和「位元组 2」。表格以这些位元组值的递增方式排列，但由下而上讨论这个表格会更有意义。

如果第一个位元组非零（表格最後一行的情况），那么它就是运行长度的重复因数。下面的图素值被重复多次，例如，位元组对

0x05 0x27

解码後的图素值为：

0x27 0x27 0x27 0x27 0x27

当然 DIB 会有许多资料不是图素到图素的重复，表格倒数第二行处理这种情况，它指出紧跟著的图素数应逐个使用。例如：考虑序列

0x00 0x06 0x45 0x32 0x77 0x34 0x59 0x90

解码後的图素值为：

0x45 0x32 0x77 0x34 0x59 0x90

这些序列总是以 2 位元组的界限排列。如果第二个位元组是奇数，那么序列内就有一个未使用的多余位元组。例如，序列

0x00 0x05 0x45 0x32 0x77 0x34 0x59 0x00

解码後的图素值为：

0x45 0x32 0x77 0x34 0x59

这就是运行长度编码的工作方式。很明显地，如果在 DIB 图像内没有重复的图素，使用此压缩技术实际上会增加了 DIB 档案的大小。

上面表格的前三行指出了矩形 DIB 图像的某些部分可以不被定义的方法。想像一下，您写的程式对已压缩的 DIB 进行解压缩，在这个解压缩的常式中，您将保持一对数字（x,y），开始为（0,0）。每对一个图素解码，x 的值就增加 1，每完成一行就将 x 重新设为 0 并且增加 y 的值。

当遇到跟著 0x02 的位元组 0x00 时，您读取下两个位元组并把它们作为无正负号的增量添加到目前的 x 和 y 值中,然後继续解码。当遇到跟著 0x00 的 0x00 时，您就解完了一行，应将 x 设 0 并增加 y 值。当遇到跟著 0x01 的 0x00 时，您就完成解码了。这些代码准许 DIB 包含那些未定义的区域，它们用於对非矩形图像编码或在制作数位动画和电影时非常有用（因为几乎每一格影像都有来自前一格的资讯而不需重新编码）。

对於 4 位元 DIB，编码一般是相同的，但更复杂，因为位元组和图素之间不是一对一的关系。

如果读取的第一个位元组非零，那就是一个重复因数 n。第二个位元组（被重复的）包含 2 个图素，在 n 个图素的被解码的序列中交替出现。例如，位元组对

0x07 0x35

被解码为：

0x35 0x35 0x35 0x3?

其中的问号指出图素还未知，如果是上面显示的 0x07 0x35 对紧跟著下面的位元组对：

0x05 0x24

则整个解码的序列为：

0x35 0x35 0x35 0x32 0x42 0x42

如果位元组对中的第一位元组是 0x00 ，第二个位元组是 0x03 或更大，则使用第二位元组指出的图素数。例如，序列

0x00 0x05 0x23 0x57 0x10 0x00

解码为：

0x23 0x57 0x1?

注意必须填补解码的序列使其成为偶数位元组。

无论 biCompression 栏位是 BI_RLE4 或 BI_RLE8，biSizeImage 栏位都指出了位元组内 DIB 图素资料的大小。如果 biCompression 栏位是 BI_RGB，则 biSizeImage 通常为 0，但是它能被设定为行内位元组长度的 biHeight 倍，就像在本章前面计算的那样。

目前文件说「由上而下的 DIB 不能被压缩」。由上而下的 DIB 是在 biHeight 栏位为负数的情况下出现的。

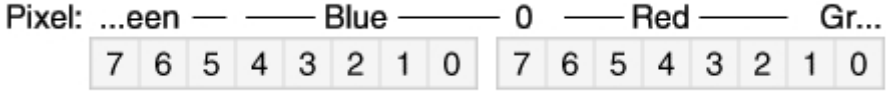
颜色遮罩（color masking）

biCompression 栏位也用於连结 Windows 95 中新出现的 16 位元和 32 位元 DIB。对於这些 DIB，biCompression 栏位可以是 BI_RGB 或 BI_BITFIELDS（均定义为值 3）。

让我们看一下 24 位元 DIB 的图素格式，它始终有一个等於 BI_RGB 的 biCompression 栏位：

也就是说，每一行基本上都是 RGBTRIPLE 结构的阵列，在每行末端有可能补充值以使行内的位元组是 4 的倍数。

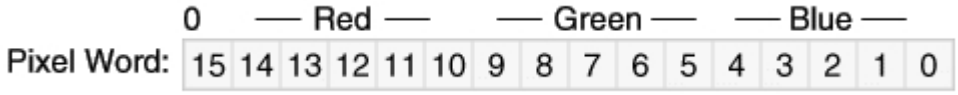
对於具有 biCompression 栏位等於 BI_RGB 的 16 位元 DIB，每个图素需要两个位元组。颜色是这样来编码的：



每种颜色使用 5 位元。对於行内的第一个图素，蓝色值是第一个位元组的

最低五位元。绿色值在第一和第二个位元组中都有位元：绿色值的两个最高位元是第二个位元组中的两个最低位元，绿色值的三个最低位元是第一个位元组中的三个最高位元。红色值是第二个位元组中的 2 到 6 位元。第二个位元组的最高位元是 0。

当以 16 位元字组存取图素值时，这会更加有意义。因为多个位元组值的最低位元首先被储存，图素字组如下：



假设在 wPixel 内储存了 16 位元图素，您能用下列公式计算红色、绿色和蓝色值：

```
Red         =     ((0x7C00 & wPixel)     >>          10)   << 3 ;
Green             =     ((0x03E0 & wPixel)     >>          5)    << 3 ;
Blue        =     ((0x001F & wPixel)     >>          0)    << 3 ;
```

首先，使用遮罩值与图素进行了位元 AND 运算。此结果是：红色向右移动 10 位元，绿色向右移动 5 位元，蓝色向右移动 0 位元。（这些移动值我称之为「右移值」）。这就产生了从 0x00 和 0x1F 的颜色值，这些值必须向左移动 3 位元以合成从 0x00 到 0xF8 的颜色值。（这些移动值我称之为「左移值」。）

请记住：如果 16 位元 DIB 的图素宽度是奇数，每行会在末端补充多余的 2 位元组以使位元组宽度能被 4 整除。

对於 32 位元 DIB，如果 biCompression 等於 BI_RGB，每个图素需要 4 位元组。蓝色值是第一个位元组，绿色为第二个，红色为第三个，第四位元组等於 0。也可这么说，图素是 RGBQUAD 结构的阵列。因为每个图素的长度是 4 位元组，在列末端就不需填补位元组。

若想以 32 位元双字组存取每个图素，它就像这样：

如果 dwPixel 是 32 位元双字组，

```
Red         =     ((0x00FF0000 & dwPixel) >>          16) << 0 ;
Green             =     ((0x0000FF00 & dwPixel) >>          8)    << 0 ;
Blue        =     ((0x000000FF & dwPixel) >>          0)    << 0 ;
```

左移值全为零，因为颜色值在 0xFF 已是最大。注意这个双字组与 Windows GDI 函式呼叫中用於指定 RGB 颜色的 32 位元 COLORREF 值不一致。在 COLORREF 值中，红色占最低位元的位元组。

到目前为止，我们讨论了当 biCompression 栏位为 BI_RGB 时，16 位元和 32 位元 DIB 的内定情况。如果 biCompression 栏位为 BI_BITFIELDS，则紧跟著 DIB 的 BITMAPINFOHEADER 结构的是三个 32 位元颜色遮罩，第一个用於红色，第

二个用於绿色，第三个用於蓝色。可以使用 C 的位元 AND 运算子（&）把这些遮罩应用於 16 位元或 32 位元的图素值上。然後通过右移值向右移动结果，这些值只有检查完遮罩後才能知道。颜色遮罩的规则应该很明确：每个颜色遮罩位元串内的 1 必须是连续的，并且 1 不能在三个遮罩位元串中重叠。

让我们来举个例子，如果您有一个 16 位元 DIB，并且 biCompression 栏位为 BI_BITFIELDS。您应该检查 BITMAPINFOHEADER 结构之後的前三个双字组：

0x0000F800

0x000007E0

0x0000001F

注意，因为这是 16 位元 DIB，所以只有位於底部 16 位元的位元值才能被设定为 1。您可以把变数 dwMask[0]、dwMask[1]和 dwMask[2]设定为这些值。现在可以编写从遮罩中计算右移和左移值的一些常式了：

```
int MaskToRShift (DWORD dwMask)
{
      int iShift ;
      if (  dwMask == 0)
                        return 0 ;

      for   (     iShift = 0 ; !(dwMask & 1)  ; iShift++)
                              dwMask >>= 1 ;

      return iShift ;
}

int MaskToLShift (DWORD dwMask)
{
      int iShift ;
      if (  dwMask == 0)
                        return 0 ;

      while (!(dwMask & 1))
                        dwMask >>= 1 ;

      for (iShift = 0 ; dwMask & 1 ; iShift++)
                        dwMask >>= 1 ;

      return 8 - iShift ;
}
```

然後呼叫 MaskToRShift 函式三次来获得右移值：

```
iRShift[0] = MaskToRShift (dwMask[0]) ;
iRShift[1] = MaskToRShift (dwMask[1]) ;
iRShift[2] = MaskToRShift (dwMask[2]) ;
```

分别得到值 11、5 和 0。然後呼叫 MaskToLShift：

```
iLShift[0] = MaskToLShift (dwMask[0]) ;
iLShift[1] = MaskToLShift (dwMask[1]) ;
iLShift[2] = MaskToLShift (dwMask[2]) ;
```

分别得到值 3、2 和 3。现在能从图素中提取每种颜色：

```
Red         = ((dwMask[0] & wPixel) >> iRShift[0]) << iLShift[0] ;
Green             = ((dwMask[1] & wPixel) >> iRShift[1]) << iLShift[1] ;
Blue        = ((dwMask[2] & wPixel) >> iRShift[2]) << iLShift[2] ;
```

除了颜色标记能大於 0x0000FFFF（这是 16 位元 DIB 的最大遮罩值）之外，程序与 32 位元 DIB 一样。

注意：

对於 16 位元或 32 位元 DIB，红色、绿色和蓝色值能大於 255。实际上，在 32 位元 DIB 中，如果遮罩中有两个为 0，第三个应为 32 位元颜色值 0xFFFFFFFF。当然，这有点荒唐，但不用担心这个问题。

不像 Windows NT，Windows 95 和 Windows 98 在使用颜色遮罩时有许多的限制。可用的值显示在表 15-2 中。

表 15-2

	16 位元 DIB	16 位元 DIB	32 位元 DIB
红色遮罩	0x00007C00	0x0000F800	0x00FF0000
绿色遮罩	0x000003E0	0x000007E0	0x0000FF00
蓝色遮罩	0x0000001F	0x0000001F	0x000000FF
速记为	5-5-5	5-6-5	8-8-8

换句话说，就是当 biCompression 是 BI_RGB 时，您能使用内定的两组遮罩，包括前面例子中显示的遮罩组。表格底行显示了一个速记符号来指出每图素红色、绿色和蓝色的位元数。

第 4 版本的 Header

我说过，Windows 95 更改了一些原始 BITMAPINFOHEADER 栏位的定义。Windows 95 也包括了一个称为 BITMAPV4HEADER 的新扩展的资讯表头。如果您知道 Windows 95 曾经称作 Windows 4.0，则就会明白此结构的名称了，Windows NT 4.0 也支援此结构。

```
typedef struct
{
DWORD           bV4Size ;               // size of the structure = 120
```

```
LONG        bV4Width ;              // width of the image in pixels
LONG              bV4Height ;             // height of the image in pixels
WORD              bV4Planes ;             // = 1
WORD              bV4BitCount ;           // bits per pixel (1, 4, 8, 16, 24, or
32)
DWORD             bV4Compression ;        // compression code
DWORD             bV4SizeImage ;          // number of bytes in image
LONG              bV4XPelsPerMeter ;      // horizontal resolution
LONG              bV4YPelsPerMeter ;      // vertical resolution
DWORD             bV4ClrUsed ;            // number of colors used
DWORD             bV4ClrImportant ;          // number of important colors
DWORD             bV4RedMask ;            // Red color mask
DWORD             bV4GreenMask ;          // Green color mask
DWORD             bV4BlueMask ;           // Blue color mask
DWORD             bV4AlphaMask ;          // Alpha mask
DWORD             bV4CSType ;             // color space type
CIEXYZTRIPLE      bV4Endpoints ;          // XYZ values
DWORD             bV4GammaRed ;           // Red gamma value
DWORD             bV4GammaGreen ;         // Green gamma value
DWORD             bV4GammaBlue ;          // Blue gamma value
}
BITMAPV4HEADER, * PBITMAPV4HEADER ;
```

注意前 11 个栏位与 BITMAPINFOHEADER 结构中的相同，後 5 个栏位支援 Windows 95 和 Windows NT 4.0 的图像颜色调配技术。除非使用 BITMAPV4HEADER 结构的後四个栏位，否则您应该使用 BITMAPINFOHEADER（或 BITMAPV5HEADER）。

当 bV4Compression 栏位等於 BI_BITFIELDS 时，bV4RedMask、bV4GreenMask 和 bV4BlueMask 可以用於 16 位元和 32 位元 DIB。它们作为定义在 BITMAPINFOHEADER 结构中的颜色遮罩用於相同的函式，并且当使用除了明确的结构栏位之外的原始结构时，它们实际上出现在 DIB 档案的相同位置。就我所知，bV4AlphaMask 栏位不被使用。

BITMAPV5HEADER 结构剩余的栏位包括「Windows 颜色管理（Image Color Management）」，它的内容超越了本书的范围，但是了解一些背景会对您有益。

为色彩使用 RGB 方案的问题在於，它依赖於视讯显示器、彩色照相机和彩色扫描器的显示技术。如果颜色指定为 RGB 值（255,0,0），意味著最大的电压应该加到阴极射线管内的红色电子枪上，RGB 值（128,0,0）表示使用一半电压。不同显示器会产生不同的效果。而且，印表机使用了不同的颜色表示方法，以青色、洋红色、黄色和黑色的组合表示颜色。这些方法称之为 CMY（cyan-magenta-yellow：青色-洋红色-黄色）和 CMYK（cyan-magenta-yellow-black：青色-洋红色-黄色-黑色）。数学公式能把 RGB 值转化为 CMY 和 CMYK，但不能保证印表机颜色与显示器颜色相符合。「色

彩调配技术」是把颜色与对装置无关的标准联系起来的一种尝试。

颜色的现象与可见光的波长有关，波长的范围从 380nm（蓝）到 780nm（红）之间。一切我们能察觉的光线是可见光谱内不同波长的组合。1931 年，Commission Internationale de L'Eclairage (International Commission on Illumination)或 CIE 开发了一种科学度量颜色的方法。这包括使用三个颜色调配函数（名称为 x、y 和 z），它们以其省略的形式（带有每 5nm 的值）发表在 CIE Publication 15.2-1986，「Colorimetry，Second Edition」的表 2.1 中。

颜色的光谱（S）是一组指出每个波长强度的值。如果知道光谱，就能够将与颜色相关的函数应用到光谱来计算 X、Y 和 Z：

$$X=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{x}(\lambda)$$

$$Y=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{y}(\lambda)$$

$$Z=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{z}(\lambda)$$

这些值称为 大 X、大 Y 和大 Z 。y 颜色匹配函式等於肉眼对范围在可见光谱内光线的反应。（他看上去像一条由 380nm 和 780nm 到 0 的时钟形曲线）。Y 称之为 CIE 亮度，因为它指出了光线的总体强度。

如果使用 BITMAPV5HEADER 结构，bV4CSType 栏位就必须设定为 LCS_CALIBRATED_RGB，其值为 0。後四个位元组必须设定为有效值。

CIEXYZTRIPLE 结构按照如下方式定义：

```
typedef struct tagCIEXYZTRIPLE
{
      CIEXYZ  ciexyzRed ;
      CIEXYZ  ciexyzGreen ;
      CIEXYZ  ciexyzBlue ;
}
CIEXYZTRIPLE, * LPCIEXYZTRIPLE ;
```

而 CIEXYZ 结构定义如下：

```
typedef struct tagCIEXYZ
{
      FXPT2DOT30 ciexyzX ;
      FXPT2DOT30 ciexyzY ;
      FXPT2DOT30 ciexyzZ ;
}
CIEXYZ, * LPCIEXYZ ;
```

这三个栏位定义为 FXPT2DOT30 值，意味著它们是带有 2 位元整数部分和 30 位元小数部分的定点值。这样，0x40000000 是 1.0，0x48000000 是 1.5。最大值 0xFFFFFFFF 仅比 4.0 小一点点。

bV4Endpoints 栏位提供了三个与 RGB 颜色（255,0,0）、（0,255,0）和（0,0,255）相关的 X、Y 和 Z 值。这些值应该由建立 DIB 的应用程式插入以指明这些 RGB 颜色的装置无关的意义。

BITMAPV4HEADER 剩余的三个栏位指「伽马值」（希腊的小写字母 γ），它指出颜色等级规格内的非线性。在 DIB 内，红、绿、蓝的范围从 0 到 225。在显示卡上，这三个数值被转化为显示器使用的三个类比电压，电压决定了每个图素的强度。然而，由於阴极射线管中电子枪的电子特性，图素的强度（I）并不与电压（V）线性相关，它们的关系为：

$$I = (V + \varepsilon)^{\gamma}$$

ε 是由显示器的「亮度」控制设定的黑色等级（理想值为 0）。指数 γ 由显示器的「图像」或「对比度」控制设定的。对於大多数显示器，γ 大约在 2.5 左右。

为了对此非线性作出补偿，摄影机在线路内包含了「伽马修正」。指数 0.45 修正了进入摄影机的光线，这意味著视讯显示器的伽马为 2.2。（视讯显示器的高伽马值增加了对比度，这通常是不需要的，因为周围的光线更适合於低对比度。）

视讯显示器的这个非线性反应实际上是很适当的，这是因为人类对光线的反应也是非线性的。我曾提过，Y 被称为 CIE 亮度，这是线性的光线度量。CIE 也定义了一个接近於人类感觉的亮度值。亮度是 L*（发音为 "ell star"），通过使用如下公式从 Y 计算得到的：

$$L^* = \begin{cases} 903.3\,\dfrac{Y}{Y_n} & \dfrac{Y}{Y_n} \le 0.008856 \\ 116\left(\dfrac{Y}{Y_n}\right)^{\frac{1}{3}} - 16 & 0.008856 < \dfrac{Y}{Y_n} \end{cases}$$

在此 Yn 是白色等级。公式的第一部分是一个小的线性部分。一般，人类的亮度感觉是与线性亮度的立方根相关的，这由第二个公式指出。L* 的范围从 0 到 100，每次 L* 的增加都假定是人类能感觉到的亮度的最小变化。

根据知觉亮度而不是线性亮度对光线强度编码要更好一些。这使得位元的数量减少到一个合理的程度并且在类比线路上也降低了杂讯。

让我们来看一下整个程序。图素值 (P)范围从 0 到 255，它被线性转化成电压等级，我们假定标准化为 0.0 到 1.0 之间的值。假设显示器的黑色级设定为 0，则图素的强度为：

$$I = V^r = \left(\frac{P}{255}\right)^r$$

这里 γ 大约为 2.5。人类感觉的亮度 (L*)依赖於此强度的立方根和变化从 0 到 100 的范围，因此大约是：

$$L^* = 100\left(\frac{P}{255}\right)^{\frac{r}{3}}$$

指数值大约为 0.85。如果指数值为 1，那么 CIE 亮度与图素值完全匹配。当然不完全是那种情况，但是如果图素值指出了线性亮度就非常接近。

BITMAPV4HEADER 的最後三个栏位为建立 DIB 的程式提供了一种为图素值指出假设的伽马值的方法。这些值由 16 位元整数值和 16 位元的小数值说明。例如，0x10000 为 1.0。如果 DIB 是捕捉实际影像而建立的，影像捕捉硬体就可能包含这个伽马值，并且可能是 2.2（编码为 0x23333）。如果 DIB 是由程式通过演算法产生的，程式会使用一个函式将它使用的任何线性亮度转化为 CIE 亮度。

第 5 版的 Header

为 Windows 98 和 Windows NT 5.0(即 Windows 2000)编写的程式能使用拥有新的 BITMAPV5HEADER 资讯结构的 DIB：

```
typedef struct
{
DWORD            bV5Size ;              // size of the structure = 120
LONG             bV5Width ;             // width of the image in pixels
LONG             bV5Height ;            // height of the image in pixels
WORD             bV5Planes ;            // = 1
WORD             bV5BitCount ;          // bits per pixel (1,4,8,16,24,or32)
```

```
DWORD            bV5Compression ;       // compression code
DWORD            bV5SizeImage ;         // number of bytes in image
LONG             bV5XPelsPerMeter ;     // horizontal resolution
LONG             bV5YPelsPerMeter ;     // vertical resolution
DWORD            bV5ClrUsed ;           // number of colors used
DWORD            bV5ClrImportant ;      // number of important colors
DWORD            bV5RedMask ;           // Red color mask
DWORD            bV5GreenMask ;         // Green color mask
DWORD            bV5BlueMask ;          // Blue color mask
DWORD            bV5AlphaMask ;         // Alpha mask
DWORD            bV5CSType ;            // color space type
CIEXYZTRIPLE     bV5Endpoints ;         // XYZ values
DWORD            bV5GammaRed ;          // Red gamma value
DWORD            bV5GammaGreen ;        // Green gamma value
DWORD            bV5GammaBlue ;         // Blue gamma value
DWORD            bV5Intent ;            // rendering intent
DWORD            bV5ProfileData ;       // profile data or filename
DWORD            bV5ProfileSize ;       // size of embedded data or filename
DWORD            bV5Reserved ;
}
BITMAPV5HEADER, * PBITMAPV5HEADER ;
```

这里有四个新栏位，只有其中三个有用。这些栏位支援 ICC Profile Format Specification，这是由「国际色彩协会（International Color Consortium）」(由 Adobe、Agfa、Apple、Kodak、Microsoft、Silicon Graphics、Sun Microsystems 及其他公司组成)建立的。您能在 http://www.icc.org 上取得这个标准的副本。基本上，每个输入（扫描器和摄影机）、输出（印表机和胶片记录器）以及显示（显示器）设备与将原始装置相关颜色（一般为 RGB 或 CMYK）联系到装置无关颜色规格的设定档案有关，最终依据 CIE XYZ 值来修正颜色。这些设定档案的副档名是.ICM（指「图像颜色管理：image color management」）。设定档案能嵌入 DIB 档案中或从 DIB 档案连结以指出建立 DIB 的方式。您能在/Platform SDK/Graphics and Multimedia Services/Color Management 中取得有关 Windows「图像颜色管理」的详细资讯。

BITMAPV5HEADER 的 bV5CSType 栏位能拥有几个不同的值。如果是 LCS_CALIBRATED_RGB，那么它就与 BITMAPV4HEADER 结构相容。bV5Endpoints 栏位和伽马栏位必须有效。

如果 bV5CSType 栏位是 LCS_sRGB，就不用设定剩余的栏位。预设的颜色空间是「标准」的 RGB 颜色空间，这是由 Microsoft 和 Hewlett-Packard 主要为 Internet 设计的，它包含装置无关的内容而不需要大量的设定档案。此文件位於 http://www.color.org/contrib/sRGB.html。

如果 bV5CSType 栏位是 LCS_Windows_COLOR_SPACE，就不用设定剩余的栏位。

Windows 通过 API 函式呼叫使用预设的颜色空间显示点阵图。

如果 bV5CSType 栏位是 PROFILE_EMBEDDED，则 DIB 档案包含一个 ICC 设定档案。如果栏位是 PROFILE_LINKED，DIB 档案就包含了 ICC 设定档案的完整路径和档案名称。在这两种情况下，bV5ProfileData 都是从 BITMAPV5HEADER 开始到设定档案资料或档案名称起始位置的偏移量。bV5ProfileSize 栏位给出了资料或档案名的大小。不必设定 bV5Endpoints 和伽马栏位。

显示 DIB 资讯

现在让我们来看一些程式码。实际上我们并不未充分了解显示 DIB 的知识，但至少能表从头结构上显示有关 DIB 的资讯。如程式 15-1 DIBHEADS 所示。

程式 15-1 DIBHEADS

```
DIBHEADS.C
/*-------------------------------------------------------------------------
-
      DIBHEADS.C --     Displays DIB Header Information
                        (c) Charles Petzold, 1998
-------------------------------------------------------------------------*
/

#include <windows.h>
#include "resource.h"

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

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

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

     hwnd = CreateWindow (szAppName, TEXT ("DIB Headers"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

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

     hAccel = LoadAccelerators (hInstance, szAppName) ;
     while (GetMessage (&msg, NULL, 0, 0))
     {
               if (!TranslateAccelerator (hwnd, hAccel, &msg))
               {
                          TranslateMessage (&msg) ;
                          DispatchMessage (&msg) ;
               }
     }
     return msg.wParam ;
}

void Printf (HWND hwnd, TCHAR * szFormat, ...)
{
     TCHAR            szBuffer [1024] ;
     va_list    pArgList ;

     va_start (pArgList, szFormat) ;
     wvsprintf (szBuffer, szFormat, pArgList) ;
     va_end (pArgList) ;

     SendMessage (hwnd, EM_SETSEL, (WPARAM) -1, (LPARAM) -1) ;
     SendMessage (hwnd, EM_REPLACESEL, FALSE, (LPARAM) szBuffer) ;
     SendMessage (hwnd, EM_SCROLLCARET, 0, 0) ;
}
void DisplayDibHeaders (HWND hwnd, TCHAR * szFileName)
{
 static TCHAR   * szInfoName []= { TEXT ("BITMAPCOREHEADER"),
                             TEXT      ("BITMAPINFOHEADER"),
                             TEXT      ("BITMAPV4HEADER"),
                             TEXT      ("BITMAPV5HEADER") } ;
Static TCHAR  * szCompression []={TEXT ("BI_RGB"),
```

```
TEXT   ("BI_RLE8"),
                              TEXT      ("BI_RLE4"),
                              TEXT      ("BI_BITFIELDS"),
                              TEXT      ("unknown") } ;
     BITMAPCOREHEADER *     pbmch ;
     BITMAPFILEHEADER *     pbmfh ;
     BITMAPV5HEADER   *     pbmih ;
     BOOL                   bSuccess ;
     DWORD             dwFileSize, dwHighSize, dwBytesRead ;
     HANDLE            hFile ;
     int               i ;
     PBYTE             pFile ;
     TCHAR            * szV ;

                       // Display the file name

     Printf (hwnd, TEXT ("File: %s\r\n\r\n"), szFileName) ;
                       // Open the file
     hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
      OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
     if (hFile == INVALID_HANDLE_VALUE)
     {
                       Printf (hwnd, TEXT ("Cannot open file.\r\n\r\n")) ;
                       return ;
   }

                       // Get the size of the file
     dwFileSize = GetFileSize (hFile, &dwHighSize) ;
     if (dwHighSize)
     {
                 Printf (hwnd, TEXT ("Cannot deal with >4G files.\r\n\r\n")) ;
                 CloseHandle (hFile) ;
                 return ;
   }
                       // Allocate memory for the file
     pFile = malloc (dwFileSize) ;
     if (!pFile)
     {
                 Printf (hwnd, TEXT ("Cannot allocate memory.\r\n\r\n")) ;
                 CloseHandle (hFile) ;
                 return ;
     }

                            // Read the file
     SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
     ShowCursor (TRUE) ;

     bSuccess = ReadFile (hFile, pFile, dwFileSize, &dwBytesRead, NULL) ;
```

```
     ShowCursor (FALSE) ;
     SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

     if (!bSuccess || (dwBytesRead != dwFileSize))
     {
                    Printf (hwnd, TEXT ("Could not read file.\r\n\r\n")) ;
                    CloseHandle (hFile) ;
               free (pFile) ;
               return ;
     }

                         // Close the file
     CloseHandle (hFile) ;
                         // Display file size
     Printf (hwnd, TEXT ("File size = %u bytes\r\n\r\n"), dwFileSize) ;
                         // Display BITMAPFILEHEADER structure
     pbmfh = (BITMAPFILEHEADER *) pFile ;
     Printf     (hwnd,     TEXT  ("BITMAPFILEHEADER\r\n")) ;
     Printf     (hwnd,     TEXT  ("\t.bfType = 0x%X\r\n"), pbmfh->bfType) ;
     Printf     (hwnd,     TEXT  ("\t.bfSize = %u\r\n"), pbmfh->bfSize) ;
     Printf     (hwnd,     TEXT  ("\t.bfReserved1        =        %u\r\n"),
pbmfh->bfReserved1) ;
     Printf     (hwnd,     TEXT  ("\t.bfReserved2        =        %u\r\n"),
pbmfh->bfReserved2) ;
     Printf     (hwnd,     TEXT  ("\t.bfOffBits       =       %u\r\n\r\n"),
pbmfh->bfOffBits) ;

                         // Determine which information structure we have

     pbmih = (BITMAPV5HEADER *) (pFile + sizeof (BITMAPFILEHEADER)) ;
     switch (pbmih->bV5Size)
     {
     case  sizeof     (BITMAPCOREHEADER):i= 0 ; break ;
     case  sizeof     (BITMAPINFOHEADER): i= 1 ; szV=
TEXT ("i") ;     break ;
     case  sizeof     (BITMAPV4HEADER):i= 2 ; szV=
TEXT ("V4") ;    break ;
     case  sizeof     (BITMAPV5HEADER):i= 3 ; szV=
TEXT ("V5") ;    break ;
     default:
Printf (hwnd,      TEXT ("Unknown header size of %u.\r\n\r\n"),
                    pbmih->bV5Size) ;
               free (pFile) ;
               return ;
     }

     Printf (hwnd, TEXT ("%s\r\n"), szInfoName[i]) ;
                                  // Display the BITMAPCOREHEADER fields
```

```
      if (pbmih->bV5Size == sizeof (BITMAPCOREHEADER))
      {
                  pbmch = (BITMAPCOREHEADER *) pbmih ;
Printf(hwnd,TEXT("\t.bcSize = %u\r\n"), pbmch->bcSize) ;
Printf(hwnd,TEXT("\t.bcWidth = %u\r\n"), pbmch->bcWidth) ;
Printf(hwnd,TEXT("\t.bcHeight = %u\r\n"), pbmch->bcHeight) ;
Printf(hwnd,TEXT("\t.bcPlanes = %u\r\n"), pbmch->bcPlanes) ;
Printf(hwnd,TEXT("\t.bcBitCount = %u\r\n\r\n"), pbmch->bcBitCount) ;
     free (pFile) ;
     return ;
    }

                        // Display the BITMAPINFOHEADER fields
Printf(hwnd,TEXT("\t.b%sSize = %u\r\n"), szV, pbmih->bV5Size) ;
Printf(hwnd,TEXT("\t.b%sWidth = %i\r\n"), szV, pbmih->bV5Width) ;
Printf(hwnd,TEXT("\t.b%sHeight = %i\r\n"), szV, pbmih->bV5Height) ;
Printf(hwnd,TEXT("\t.b%sPlanes = %u\r\n"), szV, pbmih->bV5Planes) ;
Printf(hwnd,TEXT("\t.b%sBitCount=%u\r\n"),szV, pbmih->bV5BitCount) ;
Printf(hwnd,TEXT("\t.b%sCompression = %s\r\n"), szV,
                       szCompression [min (4, pbmih->bV5Compression)]) ;
Printf(hwnd,TEXT("\t.b%sSizeImage= %u\r\n"),szV,
pbmih->bV5SizeImage) ;
Printf(hwnd,TEXT ("\t.b%sXPelsPerMeter = %i\r\n"), szV,
                  pbmih->bV5XPelsPerMeter) ;
Printf(hwnd,TEXT ("\t.b%sYPelsPerMeter = %i\r\n"), szV,
                        pbmih->bV5YPelsPerMeter) ;
Printf      (hwnd,      TEXT ("\t.b%sClrUsed = %i\r\n"), szV,
pbmih->bV5ClrUsed) ;
Printf      (hwnd,      TEXT ("\t.b%sClrImportant = %i\r\n\r\n"), szV,
               pbmih->bV5ClrImportant) ;

      if (pbmih->bV5Size == sizeof (BITMAPINFOHEADER))
    {
                        if (pbmih->bV5Compression == BI_BITFIELDS)
        {
Printf (hwnd,TEXT("Red Mask = %08X\r\n"), pbmih->bV5RedMask) ;
Printf (hwnd,TEXT ("Green Mask = %08X\r\n"), pbmih->bV5GreenMask) ;
Printf (hwnd,TEXT ("Blue Mask  = %08X\r\n\r\n"), pbmih->bV5BlueMask) ;
        }
                  free (pFile) ;
                  return ;
    }

                             // Display additional BITMAPV4HEADER fields
      Printf (hwnd,     TEXT ("\t.b%sRedMask   = %08X\r\n"), szV,
                                              pbmih->bV5RedMask) ;
      Printf (hwnd,     TEXT ("\t.b%sGreenMask = %08X\r\n"), szV,
                                              pbmih->bV5GreenMask) ;
```

```
      Printf (hwnd,    TEXT ("\t.b%sBlueMask  = %08X\r\n"), szV,
                                          pbmih->bV5BlueMask) ;
      Printf (hwnd,    TEXT ("\t.b%sAlphaMask = %08X\r\n"), szV,
                                          pbmih->bV5AlphaMask) ;
      Printf (hwnd,    TEXT ("\t.b%sCSType = %u\r\n"), szV,
                                          pbmih->bV5CSType) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzX = %08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzX) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzY = %08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzY) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzZ = %08X\r\n"),
                                      szV,
pbmih->bV5Endpoints.ciexyzRed.ciexyzZ) ;
      Printf (hwnd,    TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzX      =
%08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzX) ;
      Printf (hwnd,    TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzY      =
%08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzY) ;
      Printf (hwnd,    TEXT      ("\t.b%sEndpoints.ciexyzGreen.ciexyzZ      =
%08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzGreen.ciexyzZ) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzX = %08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzX) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzY = %08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzY) ;
      Printf (hwnd,    TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzZ = %08X\r\n"),
                                    szV,
pbmih->bV5Endpoints.ciexyzBlue.ciexyzZ) ;
      Printf (hwnd,    TEXT ("\t.b%sGammaRed = %08X\r\n"), szV,
                                    pbmih->bV5GammaRed) ;
      Printf (hwnd,    TEXT ("\t.b%sGammaGreen = %08X\r\n"), szV,
                                    pbmih->bV5GammaGreen) ;
      Printf (hwnd,    TEXT ("\t.b%sGammaBlue  = %08X\r\n\r\n"), szV,
                                    pbmih->bV5GammaBlue) ;

      if (pbmih->bV5Size == sizeof (BITMAPV4HEADER))
      {
      free (pFile) ;
      return ;
     }
```

```
            // Display additional BITMAPV5HEADER fields
      Printf           (hwnd,           TEXT ("\t.b%sIntent = %u\r\n"), szV,
pbmih->bV5Intent) ;
      Printf     (hwnd,     TEXT ("\t.b%sProfileData = %u\r\n"), szV,
                                    pbmih->bV5ProfileData) ;
      Printf     (hwnd,     TEXT ("\t.b%sProfileSize = %u\r\n"), szV,
                                    pbmih->bV5ProfileSize) ;
      Printf     (hwnd,     TEXT ("\t.b%sReserved = %u\r\n\r\n"), szV,
                                    pbmih->bV5Reserved) ;

      free (pFile) ;
      return ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      static HWND            hwndEdit ;
      static OPENFILENAME ofn ;
      static TCHAR                szFileName     [MAX_PATH],     szTitleName
[MAX_PATH] ;
      static TCHAR                szFilter[]=       TEXT("Bitmap       Files
(*.BMP)\0*.bmp\0")
                                    TEXT("All Files (*.*)\0*.*\0\0") ;

      switch (message)
      {
      case  WM_CREATE:
                  hwndEdit = CreateWindow (TEXT ("edit"), NULL,
                              WS_CHILD | WS_VISIBLE | WS_BORDER |
                              WS_VSCROLL | WS_HSCROLL |
                              ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
                              0, 0, 0, 0, hwnd, (HMENU) 1,
                               ((LPCREATESTRUCT) lParam)->hInstance, NULL) ;

            ofn.lStructSize             = sizeof (OPENFILENAME) ;
            ofn.hwndOwner          = hwnd ;
            ofn.hInstance               = NULL ;
            ofn.lpstrFilter             = szFilter ;
                  ofn.lpstrCustomFilter  = NULL ;
                  ofn.nMaxCustFilter     = 0 ;
                  ofn.nFilterIndex            = 0 ;
                  ofn.lpstrFile               = szFileName ;
                  ofn.nMaxFile                = MAX_PATH ;
                  ofn.lpstrFileTitle          = szTitleName ;
                  ofn.nMaxFileTitle           = MAX_PATH ;
                  ofn.lpstrInitialDir         = NULL ;
                  ofn.lpstrTitle              = NULL ;
```

```
                ofn.Flags                     = 0 ;
                ofn.nFileOffset               = 0 ;
                ofn.nFileExtension            = 0 ;
                     ofn.lpstrDefExt          = TEXT ("bmp") ;
                     ofn.lCustData            = 0 ;
                     ofn.lpfnHook             = NULL ;
                     ofn.lpTemplateName = NULL ;
                     return 0 ;

     case  WM_SIZE:
                MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam),
TRUE) ;
                     return 0 ;

     case  WM_COMMAND:
                     switch (LOWORD (wParam))
                     {
                     case  IDM_FILE_OPEN:
                                     if (GetOpenFileName (&ofn))
               DisplayDibHeaders (hwndEdit, szFileName) ;

                                     return 0 ;
                     }
                break ;

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

DIBHEADS.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
// Accelerator
DIBHEADS ACCELERATORS DISCARDABLE
BEGIN
     "O",          IDM_FILE_OPEN,            VIRTKEY, CONTROL, NOINVERT
END

//////////////////////////////////////////////////////////////////////////////
// Menu
DIBHEADS MENU DISCARDABLE
BEGIN
     POPUP "&File"
```

```
      BEGIN
                  MENUITEM "&Open\tCtrl+O",    IDM_FILE_OPEN
      END
END
 RESOURCE.H （摘录）
// Microsoft Developer Studio generated include file.
// Used by DibHeads.rc

#define IDM_FILE_OPEN              40001
```

此程式有一个简短的 WndProc 函式，它建立了一个唯读的编辑视窗来填满它的显示区域，它也处理功能表上的「File Open」命令。它通过呼叫 GetOpenFileName 函式使用标准的「File Open」对话方块，然後呼叫 DisplayDibHeaders 函式。此函式把整个 DIB 档案读入记忆体并逐栏地显示所有的表头资讯。

显示和列印

点阵图是用来看的。在这一节中，我们看一看 Windows 在视讯显示器上或列印页面上支援显示 DIB 的两个函式。要得到更好的性能，您可以使用一种兜圈子的方法来显示点阵图，我会在本章的後面讨论该方法，但先研究这两个函式会好一些。

这两个函式称为 SetDIBitsToDevice（发音为「set dee eye bits to device」）和 StretchDIBits （发音为「stretch dee eye bits」）。每个函式都使用储存在记忆体中的 DIB 并能显示整个 DIB 或它的矩形部分。当使用 SetDIBitsToDevice 时，以图素为单位所显示映射的大小与 DIB 的图素大小相同。例如，一个 640×480 的 DIB 会占据整个标准的 VGA 萤幕，但在 300dpi 的雷射印表机上它只有约 2.1×1.6 英寸。StretchDIBits 能延伸和缩小 DIB 尺寸的行和列从而在输出设备上显示一个特定的大小。

了解 DIB

当呼叫两个函式之一来显示 DIB 时，您需要几个关於图像的资讯。正如我前面说过的，DIB 档案包含下列部分：

DIB 档案能被载入记忆体。如果除了档案表头外，整个档案被储存在记忆体的连续区块中，指向该记忆体块开始处（也就是资讯表头的开头）的指标被称为指向 packed DIB 的指标（见下图）。

这是通过剪贴簿传输 DIB 时所用的格式，并且也是您从 DIB 建立画刷时所用的格式。因为整个 DIB 由单个指标（如 pPackedDib）引用，所以 packed DIB 是在记忆体中储存 DIB 的方便方法，您可以把指标定义为指向 BYTE 的指标。使用本章前面所示的结构定义，能得到所有储存在 DIB 内的资讯，包括色彩对照表和个别图素位元。

然而，要想得到这么多资讯，还需要一些程式码。例如，您不能通过以下叙述简单地取得 DIB 的图素宽度：

```
iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;
```

DIB 有可能是 OS/2 相容格式的。在那种格式中，packed DIB 以 BITMAPCOREHEADER 结构开始，并且 DIB 的图素宽度和高度以 16 位元 WORD，而不是 32 位元 LONG 储存。因此，首先必须检查 DIB 是否为旧的格式，然後进行相对应的操作：

```
if (((PBITMAPCOREHEADER) pPackedDib)->bcSize == sizeof (BITMAPCOREHEADER))
      iWidth = ((PBITMAPCOREHEADER) pPackedDib)->bcWidth ;
else
      iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;
```

当然，这不很糟，但它不如我们所喜好的清晰。

现在有一个很有趣的实验：给定一个指向 packed DIB 的指标，我们要找出

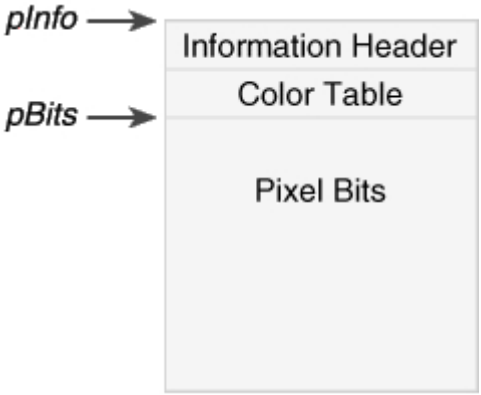
位於座标（5,27）的图素值。即使假定 DIB 不是 OS/2 相容的格式，您也需要了解 DIB 的宽度、高度和位元数。您需要计算每一列图素的位元组长度，确定色彩对照表内的项目数，以及色彩对照表是否包括三个 32 位元的颜色遮罩。您还需检查 DIB 是否被压缩，在这种情况下图素是不能直接由位址得到的。

如果您需要直接存取所有的 DIB 图素（就像许多图形处理工作一样），这可能会增加一点处理时间。由於这个原因，储存一个指向 packed DIB 的指标就很方便了，不过这并不是一种有效率的解决方式。另一个漂亮的解决方法是为 DIB 定义一个包含足够成员资料的 C++类别，从而允许快速随机地存取 DIB 图素。然而，我曾经答应读者在本书内无需了解 C++，我将在下一章说明一个 C 的解决方法。

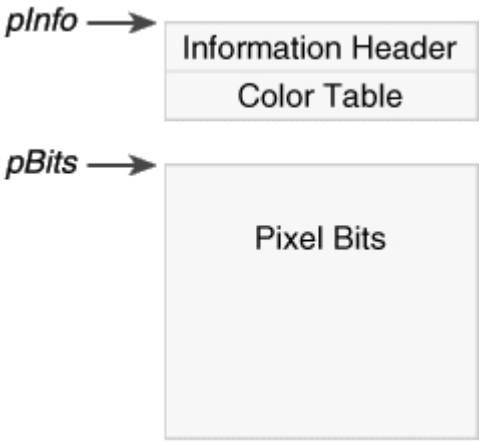
对於 SetDIBitsToDevice 和 StretchDIBits 函式，需要的资讯包括一个指向 DIB 的 BITMAPINFO 结构的指标。您应回想起，BITMAPINFO 结构由 BITMAPINFOHEADER 结构和色彩对照表组成。因此这仅是一个指向 packed DIB 的指标。

函式也需要一个指向图素位元的指标。尽管程式码写得很不漂亮，但这个指标还是可以从资讯表头内的资讯推出。注意，当您存取 BITMAPFILEHEADER 结构的 bfOffBits 栏位时，这个指标能很容易地计算出。bfOffBits 栏位指出了从 DIB 档案的开头到图素位元的偏移量。您可以简单地把此偏移量加到 BITMAPINFO 指标中，然後减去 BITMAPFILEHEADER 结构的大小。然而，当您从剪贴簿上得到指向 packed DIB 的指标时，这并不起作用，因为没有 BITMAPFILEHEADER 结构。

此图表显示了两个所需的指标：



SetDIBitsToDevice 和 StretchDIBits 函式需要两个指向 DIB 的指标，因为这两个部分不在连续的记忆体块内。您可能有如下所示的两块记忆体：



确实，把 DIB 分成两个记忆体块是很有用的，只是我们更喜欢与整个 DIB 储存在单个记忆体块的 packed DIB 打交道。

除了这两个指标，SetDIBitsToDevice 和 StretchDIBits 函式通常也需要 DIB 的图素宽度和高度。如只想显示 DIB 的一部分，就不必明确地知道这些值，但它们会定义您在 DIB 图素位元阵列内定义的矩形的上限。

点对点图素显示

SetDIBitsToDevice 函式显示没有延伸和缩小的 DIB。DIB 的每个图素对应到输出设备的一个图素上，而且 DIB 中的图像一定会被正确显示出来——也就是说，图像的顶列在上方。任何会影响装置内容的座标转换都影响了显示 DIB 的开始位置，但不影响显示出来的图片大小和方向。该函式如下：

```
iLines =    SetDIBitsToDevice (
                              hdc,              // device context handle
                              xDst,             // x destination coordinate
                              yDst,             // y destination coordinate
                              cxSrc,            // source rectangle width
                              cySrc,            // source rectangle height
                              xSrc,             // x source coordinate
                              ySrc,             // y source coordinate
                              yScan,            // first scan line to draw
                              cyScans,          // number of scan lines to draw
                              pBits,            // pointer to DIB pixel bits
                              pInfo,            // pointer to DIB information
                              fClrUse) ;        // color use flag
```

不要对参数的数量感到厌烦，在多数情况下，函式用起来比看起来要简单。不过在其他用途上来说，它的用法真的是乱七八糟，不过我们将学会怎么用它。

和 GDI 显示函式一样，SetDIBitsToDevice 的第一个参数是装置内容代号，它指出显示 DIB 的设备。下面两个参数 xDst 和 yDst，是输出设备的逻辑座标，并指出了显示 DIB 图像左上角的座标（「上端」指的是视觉上的上方，并不是 DIB 图素的第一行）。注意，这些都是逻辑座标，因此它们附属於实际上起作用

的任何座标转换方式或——在 Windows NT 的情况下——设定的任何空间转换。在内定的 MM_TEXT 映射方式下，可以把这些参数设为 0，从显示平面上向左向上显示 DIB 图像。

您可以显示整个 DIB 图像或仅显示其中的一部分，这就是後四个参数的作用。但是 DIB 图素资料的由上而下的方向产生了许多误解，待会儿会谈到这些。现在应该清楚当显示整个 DIB 时，应把 xSrc 和 ySrc 设定为 0，并且 cxSrc 和 cySrc 应分别等於 DIB 的图素宽度和高度。注意，因为 BITMAPINFOHEADER 结构的 biHeight 栏位对於由上而下的 DIB 来说是负的，cySrc 应设定为 biHeight 栏位的绝对值。

此函式的文件（/Platform SDK/Graphics and Multimedia Services/GDI/Bitmaps/Bitmap Reference/Bitmap Functions/SetDIBitsToDevice）中说 xSrc、ySrc、cxSrc 和 cySrc 参数是逻辑-单位。这是不正确的，它们是图素的座标和尺寸。对於 DIB 内的图素，拥有逻辑座标和单位是没有什么意义的。而且，不管是什么映射方式，在输出设备上显示的 DIB 始终是 cxSrc 图素宽和 cySrc 图素高。

现在先不详细讨论这两个参数 yScan 和 cyScan。这些参数在您从磁片档案或通过数据机读取资料时，透过每次显示 DIB 的一小部分减少对记忆体的需求。通常，yScan 设定为 0，cyScan 设定为 DIB 的高度。

pBits 参数是指向 DIB 图素位元的指标。pInfo 参数是指向 DIB 的 BITMAPINFO 结构的指标。虽然 BITMAPINFO 结构的位址与 BITMAPINFOHEADER 结构的位址相同，但是 SetDIBitsToDevice 结构被定义为使用 BITMAPINFO 结构，暗示著：对於 1 位元、4 位元和 8 位元 DIB，点阵图资讯表头後必须跟著色彩对照表。尽管 pInfo 参数被定义为指向 BITMAPINFO 结构的指标，它也是指向 BITMAPCOREINFO、BITMAPV4HEADER 或 BITMAPV5HEADER 结构的指标。

最後一个参数是 DIB_RGB_COLORS 或 DIB_PAL_COLORS，在 WINGDI.H 内分别定义为 0 和 1。如果您使用 DIB_RGB_COLORS，这意味著 DIB 包含了色彩对照表。DIB_PAL_COLORS 旗标指出，DIB 内的色彩对照表已经被指向在装置内容内选定并识别的调色盘的 16 位元索引代替。在下一章我们将学习这个选项。现在先使用 DIB_RGB_COLORS，或者是 0。

SetDIBitsToDevice 函式传回所显示的扫描行的数目。

因此，要呼叫 SetDIBitsToDevice 来显示整个 DIB 图像，您需要下列资讯：

- hdc 目的表面的装置内容代号
- xDst 和 yDst 图像左上角的目的座标
- cxDib 和 cyDib DIB 的图素宽度和高度，在这里，cyDib 是

BITMAPINFOHEADER 结构内 biHeight 栏位的绝对值。

- pInfo 和 pBits 指向点阵图资讯部分和图素位元的指标

然後用下列方法呼叫 SetDIBitsToDevice：

```
SetDIBitsToDevice (
      hdc,        // device context handle
      xDst,       // x destination coordinate
      yDst,       // y destination coordinate
      cxDib,      // source rectangle width
      cyDib,      // source rectangle height
      0,          // x source coordinate
      0,          // y source coordinate
      0,          // first scan line to draw
      cyDib,      // number of scan lines to draw
      pBits,      // pointer to DIB pixel bits
      pInfo,         // pointer to DIB information
      0) ;        // color use flag
```

因此，在 DIB 的 12 个参数中，四个设定为 0，一个是重复的。

程式 15-2 SHOWDIB1 通过使用 SetDIBitsToDevice 函式显示 DIB。

程式 15-2 SHOWDIB1

```
SHOWDIB1.C
/*------------------------------------------------------------------------
      SHOWDIB1.C --          Shows a DIB in the client area
                             (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
      HACCEL            hAccel ;
      HWND              hwnd ;
      MSG                     msg ;
      WNDCLASS          wndclass ;

      wndclass.style                    = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc        = WndProc ;
      wndclass.cbClsExtra         = 0 ;
      wndclass.cbWndExtra         = 0 ;
      wndclass.hInstance          = hInstance ;
      wndclass.hIcon              = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor            = LoadCursor (NULL, IDC_ARROW) ;
```

```
     wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName        = szAppName ;
     wndclass.lpszClassName       = szAppName ;

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

     hwnd = CreateWindow (szAppName, TEXT ("Show DIB #1"),
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

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

     hAccel = LoadAccelerators (hInstance, szAppName) ;
     while (GetMessage (&msg, NULL, 0, 0))
     {
                          if (!TranslateAccelerator (hwnd, hAccel, &msg))
                          {
                                    TranslateMessage (&msg) ;
                                    DispatchMessage (&msg) ;
                 }
     }
     return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
     static     BITMAPFILEHEADER * pbmfh ;
     static     BITMAPINFO       * pbmi ;
     static     BYTE             * pBits ;
     static     int              cxClient, cyClient, cxDib, cyDib ;
     static          TCHAR              szFileName [MAX_PATH], szTitleName
[MAX_PATH] ;
     BOOL                  bSuccess ;
     HDC                    hdc ;
     PAINTSTRUCT                 ps ;

     switch (message)
     {
     case  WM_CREATE:
                          DibFileInitialize (hwnd) ;
```

```
                    return 0 ;

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

     case  WM_INITMENUPOPUP:
                    EnableMenuItem ((HMENU) wParam, IDM_FILE_SAVE,
               pbmfh ? MF_ENABLED : MF_GRAYED) ;
                    return 0 ;

     case  WM_COMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_OPEN:
                          // Show the File Open dialog box

                                if (!DibFileOpenDlg (hwnd, szFileName,
szTitleName))
                                return 0 ;

                          // If there's an existing DIB, free the memory

                                if (pbmfh)
                                {
                                free (pbmfh) ;
                          pbmfh = NULL ;
                          }
                                // Load the entire DIB into memory

                          SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                          ShowCursor (TRUE) ;

                          pbmfh = DibLoadImage (szFileName) ;
                          ShowCursor (FALSE) ;
                          SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                    // Invalidate the client area for later update

                          InvalidateRect (hwnd, NULL, TRUE) ;

                          if (pbmfh == NULL)
                          {
            MessageBox (   hwnd, TEXT ("Cannot load DIB file"),
                          szAppName, 0) ;
                          return 0 ;
                          }
```

```
                    // Get pointers to the info structure & the bits

                                    pbmi  = (BITMAPINFO *) (pbmfh + 1) ;
                                    pBits      =     (BYTE     *)     pbmfh     +
pbmfh->bfOffBits ;

                                    // Get the DIB width and height

                   if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
                                    {
                    cxDib     = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
                    cyDib     = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
                                    }
                                    else
                                   {
                 cxDib =     pbmi->bmiHeader.biWidth ;
                 cyDib       = abs(      pbmi->bmiHeader.biHeight) ;
                                    }
                                    return 0 ;

                                   case  IDM_FILE_SAVE:
                 // Show the File Save dialog box

                                   if     (!DibFileSaveDlg     (hwnd,     szFileName,
szTitleName))
                                    return 0 ;

                                    // Save the DIB to memory

                                    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                                    ShowCursor (TRUE) ;

                                    bSuccess = DibSaveImage (szFileName, pbmfh) ;
                                    ShowCursor (FALSE) ;
                                    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                                    if (!bSuccess)
               MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
                           szAppName, 0) ;
                                   return 0 ;
                   }
                   break ;

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

                         if (pbmfh)
                   SetDIBitsToDevice (hdc,
```

```
                              0,               // xDst
                              0,               // yDst
                              cxDib,           // cxSrc
                              cyDib,           // cySrc
                              0,               // xSrc
                              0,               // ySrc
                              0,               // first scan line
                              cyDib,           // number of scan lines
                              pBits,
                              pbmi,
                              DIB_RGB_COLORS) ;
                    EndPaint (hwnd, &ps) ;
                    return 0 ;

     case  WM_DESTROY:
                    if (pbmfh)
                              free (pbmfh) ;

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

DIBFILE.H

```
/*------------------------------------------------------------------------
     DIBFILE.H -- Header File for DIBFILE.C
  ------------------------------------------------------------------------*/
void DibFileInitialize (HWND hwnd) ;
BOOL DibFileOpenDlg    (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;
BOOL DibFileSaveDlg    (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;

BITMAPFILEHEADER *     DibLoadImage      (PTSTR pstrFileName) ;
BOOL           DibSaveImage(PTSTR pstrFileName, BITMAPFILEHEADER *) ;
```

DIBFILE.C

```
/*------------------------------------------------------------------------
-
     DIBFILE.C -- DIB File Functions
------------------------------------------------------------------------
*/

#include <windows.h>
#include <commdlg.h>
#include "dibfile.h"

static OPENFILENAME ofn ;
void DibFileInitialize (HWND hwnd)
{
     static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0") \
```

```
              TEXT ("All Files (*.*)\0*.*\0\0") ;
      ofn.lStructSize             = sizeof (OPENFILENAME) ;
      ofn.hwndOwner          = hwnd ;
      ofn.hInstance               = NULL ;
      ofn.lpstrFilter             = szFilter ;
      ofn.lpstrCustomFilter  = NULL ;
      ofn.nMaxCustFilter     = 0 ;
      ofn.nFilterIndex       = 0 ;
      ofn.lpstrFile          = NULL ;  // Set in Open and Close functions
      ofn.nMaxFile           = MAX_PATH ;
      ofn.lpstrFileTitle     = NULL ;   // Set in Open and Close functions
      ofn.nMaxFileTitle           = MAX_PATH ;
      ofn.lpstrInitialDir         = NULL ;
      ofn.lpstrTitle              = NULL ;
      ofn.Flags                   = 0 ;      // Set in Open and Close functions
      ofn.nFileOffset             = 0 ;
      ofn.nFileExtension          = 0 ;
      ofn.lpstrDefExt             = TEXT ("bmp") ;
      ofn.lCustData               = 0 ;
      ofn.lpfnHook                = NULL ;
      ofn.lpTemplateName      = NULL ;
}
BOOL DibFileOpenDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
      ofn.hwndOwner          = hwnd ;
      ofn.lpstrFile               = pstrFileName ;
      ofn.lpstrFileTitle          = pstrTitleName ;
      ofn.Flags                   = 0 ;

      return GetOpenFileName (&ofn) ;
}

BOOL DibFileSaveDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
      ofn.hwndOwner          = hwnd ;
      ofn.lpstrFile               = pstrFileName ;
      ofn.lpstrFileTitle          = pstrTitleName ;
      ofn.Flags                   = OFN_OVERWRITEPROMPT ;

      return GetSaveFileName (&ofn) ;
}

BITMAPFILEHEADER * DibLoadImage (PTSTR pstrFileName)
{
      BOOL                  bSuccess ;
      DWORD                 dwFileSize, dwHighSize, dwBytesRead ;
      HANDLE                hFile ;
      BITMAPFILEHEADER *    pbmfh ;
```

```
     hFile = CreateFile (   pstrFileName, GENERIC_READ, FILE_SHARE_READ,NULL,
      OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN,NULL) ;

     if    (     hFile == INVALID_HANDLE_VALUE)
                      return NULL ;

     dwFileSize = GetFileSize (hFile, &dwHighSize) ;

     if (dwHighSize)
     {
                      CloseHandle (hFile) ;
                      return NULL ;
     }

     pbmfh = malloc (dwFileSize) ;
     if          (!pbmfh)
     {
                      CloseHandle (hFile) ;
                      return NULL ;
    }

     bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
     CloseHandle (hFile) ;

     if (!bSuccess     || (dwBytesRead != dwFileSize)
                       || (pbmfh->bfType != * (WORD *) "BM")
                       || (pbmfh->bfSize != dwFileSize))
     {
                       free (pbmfh) ;
                       return NULL ;
     }
     return pbmfh ;
}

BOOL DibSaveImage (PTSTR pstrFileName, BITMAPFILEHEADER * pbmfh)
{
     BOOL              bSuccess ;
     DWORD       dwBytesWritten ;
     HANDLE      hFile ;

     hFile = CreateFile (   pstrFileName, GENERIC_WRITE, 0, NULL,
   CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;

     if (hFile == INVALID_HANDLE_VALUE)
                       return FALSE ;
     bSuccess = WriteFile (hFile, pbmfh, pbmfh->bfSize, &dwBytesWritten, NULL) ;
     CloseHandle (hFile) ;
```

```
      if (!bSuccess || (dwBytesWritten != pbmfh->bfSize))
      {
                  DeleteFile (pstrFileName) ;
                  return FALSE ;
     }
      return TRUE ;
}
```

SHOWDIB1.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Menu
SHOWDIB1 MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
            MENUITEM "&Open...", IDM_FILE_OPEN
            MENUITEM "&Save...", IDM_FILE_SAVE
      END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib1.rc

#define IDM_FILE_OPEN        40001
#define IDM_FILE_SAVE        40002
```

DIBFILE.C 档案包含了显示「File Open」和「File Save」对话方块的常式，以及把整个 DIB 档案（拥有 BITMAPFILEHEADER 结构）载入单个记忆体块的常式。程式也会将这样一个记忆体区写出到档案。

当在 SHOWDIB1.C 内执行「File Open」命令载入 DIB 档案後，程式计算记忆体块中 BITMAPINFOHEADER 结构和图素位元的偏移量，程式也获得 DIB 的图素宽度和高度。所有资讯都储存在静态变数中。在处理 WM_PAINT 讯息处理期间，程式通过呼叫 SetDIBitsToDevice 显示 DIB。

当然，SHOWDIB1 还缺少一些功能。例如，如果 DIB 对显示区域来说太大，则没有卷动列可用来移动查看。在下一章的末尾将修改这些缺陷。

DIB 的颠倒世界

我们将得到一个重要的教训，它不仅在生活中重要，而且在作业系统的应用程式介面的设计中也重要。这个教训是：覆水难收。

回到 OS/2 Presentation Manager 那由下而上的 DIB 图素位元的定义处，这样的定义是有点道理的,因为 PM 内的任何座标系都有一个内定的左下角原点。例如：在 PM 视窗内，内定的（0,0）原点是视窗的左下角。（如果您觉得这很古怪，很多人和您的感觉一样。如果您不觉得古怪，那您可能是位数学家。）点阵图的绘制函式也根据左下角指定目的地。

因此，在 OS/2 内如果给点阵图指定了目的座标(0,0)，则图像将从视窗的左下角向上向右显示，如图 15-1 所示。

图 15-1 在 OS/2 中以（0，0）为目的点显示的点阵图

在够慢的机器上，您能实际看到电脑由下而上绘制点阵图。

尽管 OS/2 座标系统显得很古怪,但它的优点是高度的一致。点阵图的 (0,0) 原点是点阵图档案中第一行的第一个图素，并且此图素被映射到在点阵图绘制函式中指定的目的座标上。

Windows 存在的问题是不能保持内部的一致性。当您只要显示整个 DIB 图像中的一小块矩形时，就要使用参数 xSrc、ySrc、cxSrc 和 cySrc。这些来源座标和大小与 DIB 资料的第一行（图像的最後一行）相关。这方面与 OS/2 相似，与 OS/2 不同的是，Windows 在目的座标上显示图像的顶列。因此，如果显示整个 DIB 图像，显示在（xDst,yDst）的图素是位於座标（0,cyDib - 1）处的图素。DIB 资料的最後一列就是图形的顶列。如果仅显示图像的一部分，则在（xDst,yDst）显示的图素是位於座标（xSrc, ySrc + cySrc - 1）处的 DIB 图素。

图 15-2 显示的图表将帮助您理解这方面的内容。您可以把下面显示的 DIB 当成是储存在记忆体中的——就是说，上下颠倒。座标的原点与 DIB 图素资料的第一个位元是一致的。SetDIBitsToDevice 的 xSrc 参数是以 DIB 的左边为基准，并且 cxSrc 是 xSrc 右边的图像宽度，这很直观。ySrc 参数以 DIB 资料的首列（也就是图像的底部）为基准，并且 cySrc 是从 ySrc 到资料的末列（图像的顶端）的图像高度。

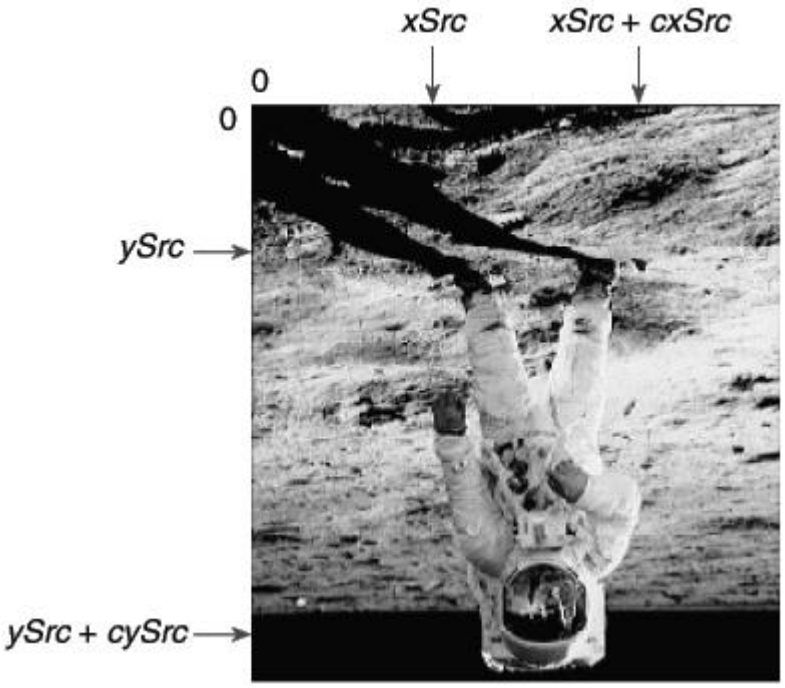


图 15-2 正常 DIB（由下而上）的座标

如果目的装置内容具有使用 MM_TEXT 映射方式的内定图素座标，来源矩形和目的矩形角落座标之间的关系显示在表 15-3 中。

表 15-3

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cySrc - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxSrc - 1, yDst + cySrc - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxSrc - 1, yDst)

(xSrc,ySrc)不映射到(xDst,yDst)，使得表格显得很混乱。在其他映射方式中，点(xSrc,ySrc + cySrc - 1)总是映射到逻辑点(xDst,yDst)，图像也与 MM_TEXT 所显示的一样。

到目前为止，我们讨论了当 BITMAPINFOHEADER 结构的 biHeight 栏位是正值时的正常情况。如果 biHeight 栏位是负值，则 DIB 资料会以合理的由上而下的方式排列。您可能会认为这样将解决所有问题，如果您真地这样认为，那您就错了。

很明显地，有人会认为如果把 DIB 上下倒置，旋转每一行，然後给 biHeight 设定一个正值，它将像正常的由下而上的 DIB 一样操作，所有与 DIB 矩形相关的现存程式码就不必修改。我认为这是一个合理的目的，但它忘记了一个事实，程式需要修改以处理由上而下的 DIB，这样就不会使用一个负高度。

而且，此决定的结果意味著由上而下的 DIB 的来源座标在 DIB 资料的最後一列有一个原点，它也是图像的底列。这与我们遇到的情况完全不同。位於(0,0)原点的 DIB 图素不再是 pBits 指标引用的第一个图素，也不是 DIB 档案的最後一个图素，它位於两者之间。

图 15-3 显示的图表说明了在由上而下的 DIB 中指定矩形的方法，也是它储存在档案或记忆体中的样子。

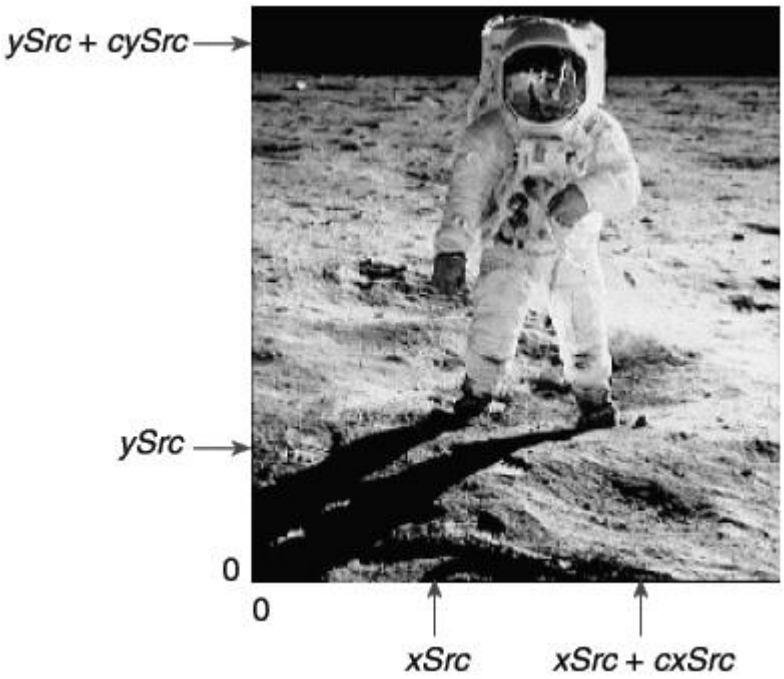


图 15-3 指定由上而下的 DIB 的座标

无论如何，这个方案的实际优点是 SetDIBitsToDevice 函式的参数与 DIB 资料的方向无关。如果有显示了同一图像的两个 DIB（一个由下而上，另一个由上而下。表示在两个 DIB 档案内的列顺序相反），您可以使用相同的参数呼叫 SetDIBitsToDevice 来选择显示图像的相同部分。

如程式 15-3 APOLLO11 中所示。

程式 15-3 APOLLO11

```
APOLLO11.C
/*-----------------------------------------------------------------------
      APOLLO11.C --     Program for screen captures
                        (c) Charles Petzold, 1998
  ----------------------------------------------------------------------*/

#include <windows.h>
```

```
#include "dibfile.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Apollo11") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
      HWND             hwnd ;
      MSG                    msg ;
      WNDCLAS          wndclass ;

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

      if (!RegisterClass (&wndclass))
     {
                         MessageBox (NULL, TEXT ("This program requires Windows
NT!"),
                                     szAppName, MB_ICONERROR) ;
                         return 0 ;
      }
      hwnd = CreateWindow (szAppName, TEXT ("Apollo 11"),
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;

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

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

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
```

```
     static BITMAPFILEHEADER     * pbmfh [2] ;
     static BITMAPINFO      * pbmi  [2] ;
     static BYTE           * pBits [2] ;
     static int            cxClient, cyClient, cxDib[2], cyDib[2] ;
     HDC                   hdc ;
     PAINTSTRUCT             ps ;

     switch (message)
     {
     case  WM_CREATE:
                       pbmfh[0] = DibLoadImage (TEXT ("Apollo11.bmp")) ;
                       pbmfh[1] = DibLoadImage (TEXT ("ApolloTD.bmp")) ;

                       if (pbmfh[0] == NULL || pbmfh[1] == NULL)
                       {
          MessageBox (hwnd, TEXT ("Cannot load DIB file"),
                 szAppName, 0) ;
                                 return 0 ;
                       }
                                 // Get pointers to the info structure & the
bits

                       pbmi  [0] = (BITMAPINFO *) (pbmfh[0] + 1) ;
                       pbmi  [1] = (BITMAPINFO *) (pbmfh[1] + 1) ;
                       pBits       [0] = (BYTE *) pbmfh[0] + pbmfh[0]->bfOffBits ;
                       pBits       [1] = (BYTE *) pbmfh[1] + pbmfh[1]->bfOffBits ;

                       // Get the DIB width and height (assume BITMAPINFOHEADER)
                       // Note that cyDib is the absolute value of the header
value!!!

                       cxDib [0] =  pbmi[0]->bmiHeader.biWidth ;
                       cxDib [1] =  pbmi[1]->bmiHeader.biWidth ;

                       cyDib [0] = abs (pbmi[0]->bmiHeader.biHeight) ;
                       cyDib [1] = abs (pbmi[1]->bmiHeader.biHeight) ;
                       return 0 ;

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

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

                       // Bottom-up DIB full size
```

```
            SetDIBitsToDevice (hdc,
        0,                                  // xDst
        cyClient / 4,                       // yDst
        cxDib[0],                           // cxSrc
        cyDib[0],                           // cySrc
        0,                                  // xSrc
        0,                                  // ySrc
        0,                                  // first scan line
        cyDib[0],                           // number of scan lines
        pBits[0],
        pbmi[0],
        DIB_RGB_COLORS) ;

                    // Bottom-up DIB partial

         SetDIBitsToDevice (hdc,
    240,                                // xDst
    cyClient / 4,                       // yDst
    80,                                 // cxSrc
    166,                                // cySrc
    80,                                 // xSrc
    60,                                 // ySrc
    0,                                  // first scan line
    cyDib[0],                           // number of scan lines
    pBits[0],
    pbmi[0],
    DIB_RGB_COLORS) ;

                    // Top-down DIB full size

            SetDIBitsToDevice      (hdc,
           340,              // xDst
           cyClient / 4,     // yDst
           cxDib[0],       // cxSrc
           cyDib[0],       // cySrc
           0,                // xSrc
           0,                // ySrc
           0,                // first scan line
           cyDib[0],      // number of scan lines
           pBits[0],
           pbmi[0],
           DIB_RGB_COLORS) ;

                    // Top-down DIB partial

            SetDIBitsToDevice      (hdc,
        580,                    // xDst
        cyClient / 4,           // yDst
```

```
                    80,                     // cxSrc
                    166,                    // cySrc
                    80,                     // xSrc
                    60,                     // ySrc
                    0,                      // first scan line
                    cyDib[1],               // number of scan lines
                    pBits[1],
                    pbmi[1],
                    DIB_RGB_COLORS) ;

                         EndPaint (hwnd, &ps) ;
                         return 0 ;

     case  WM_DESTROY:
                         if (pbmfh[0])
                                          free (pbmfh[0]) ;
                         if (pbmfh[1])
                                          free (pbmfh[1]) ;

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

程式载入了名为 APOLLO11.BMP（由下而上版本）和 APOLLOTD.BMP（由上而下版本）的两个 DIB。它们都是 220 图素宽和 240 图素高。注意，在程式从表头资讯结构中确定 DIB 的宽度和高度时，它使用 abs 函式得到 biHeight 栏位的绝对值。当以全部大小或范围显示 DIB 时，不管显示点阵图的种类，xSrc、ySrc、cxSrc 和 cySrc 座标都是相同的。结果如图 15-4 所示。

图 15-4 APOLLO11 的萤幕显示

注意，「第一条扫描线」和「扫描线数目」参数保持不变，我将在以後简短说明。pBits 参数也不变，不要只为了使它指向您需要显示的区域而试图更改 pBits。

我在这个问题上花了这么多时间，并不是因为要让那些试图跟 API 定义中有问题的部分妥协的 Windows 程式写作者难堪，而是想让您不至於因为这个令人混淆的问题而紧张起来。这个问题之所以令人困惑，是因为它本身早就被搞混了。

我也想让您留意 Windows 文件中的某些叙述，例如对 SetDIBitsToDevice，文件说：「由下而上 DIB 的原点是点阵图的左下角；由上而下 DIB 的原点是左上角」。这不仅模糊，而且是错误的。我可以用更好的方式来讲述：由下而上 DIB 的原点是点阵图图像的左下角，它是点阵图资料的第一列的第一个图素。由上而下 DIB 的原点也是点阵图图像的左下角，但在这种情况下，左下角是点阵图资料的最後一列的第一个图素。

如果要撰写存取 DIB 个别位元的函式，问题会变的更糟。这应该与您为显示部分 DIB 映射而指定的座标一致，我的解决方法是（我将在第十六章的 DIB 程式库中使用）以统一的手法参考 DIB 图素和座标，就像在图像被正确显示时（0,0）原点所指的是 DIB 图像顶行的最左边的图素一样。

循序显示

拥有大量记忆体能确保程式更容易地执行。要显示磁片档案内的 DIB，可以分为两个独立的工作：将 DIB 载入记忆体，然後显示它。

然而，您也可能在不把整个档案载入记忆体的情况下显示 DIB。即使有足够的实体记忆体提供给 DIB，把 DIB 移入记忆体也会迫使 Windows 的虚拟记忆体系统把记忆体中别的资料和程式码移到磁片上。如果 DIB 仅用於显示并立即从记忆体中消除，这就非常讨厌。

还有另一个问题：假设 DIB 位於例如软碟的慢速储存媒体上，或由数据机传输过来，或者来自扫描器或视频截取程式取得图素资料的转换常式。您是否得等到整个 DIB 被载入记忆体後才显示它？还是从磁片或电话线或扫描器上得到 DIB 时，就开始显示它？

解决这些问题是 SetDIBitsToDevice 函式中 yScan 和 cyScans 参数的目的。要使用这个功能，需要多次呼叫 SetDIBitsToDevice，大多数情况下使用同样的参数。然而对於每次呼叫，pBits 参数指向点阵图图素总体排列的不同部分。yScans 参数指出了 pBits 指向图素资料的行，cyScans 参数是被 pBits 引用的行数。这大量地减少了记忆体需求。您仅需要为储存 DIB 的资讯部分（BITMAPINFOHEADER 结构和色彩对照表）和至少一行图素资料配置足够的记忆体。

例如，假设 DIB 有 23 行图素，您希望每次最多 5 行的分段显示这个 DIB。您可能想配置一个由变数 pInfo 引用的记忆体块来储存 DIB 的 BITMAPINFO 部分，然後从档案中读取该 DIB。在检查完此结构的栏位後，能够计算出一行的位元组长度。乘以 5 并配置该大小的另一个记忆体块（pBits）。现在读取前 5 行，呼叫您正常使用的函式，把 yScan 设定为 0，把 cyScans 设定为 5。现在从档案中读取下 5 行，这一次将 yScan 设定为 5，继续将 yScan 设定为 10，然後为 15。最後，将最後 3 行读入 pBits 指向的记忆体块，并将 yScan 设定为 20，将 cyScans 设定为 3，以呼叫 SetDIBitsToDevice。

现在有一个不好的讯息。首先，使用 SetDIBitsToDevice 的这个功能要求程式的资料取得和资料显示元素之间结合得相当紧密。这通常是不理想的，因为您必须在获得资料和显示资料之间切换。首先，您将延缓整个程序；第二，SetDIBitsToDevice 是唯一具有这个功能的点阵图显示函式。StretchDIBits 函式不包括这个功能，因此您不能使用它以不同图素大小显示发表的 DIB。您必须呼叫 StretchDIBits 多次，每次更改 BITMAPINFOHEADER 结构中的资讯，并在萤幕的不同区域显示结果。

程式 15-4 SEQDISP 展示了这个功能的使用方法。

程式 15-4 SEQDISP

```
SEQDISP.C
/*-------------------------------------------------------------------------
--
      SEQDISP.C --      Sequential Display of DIBs
                        (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

#include <windows.h>
#include "resource.h"

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

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

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

      hwnd = CreateWindow (szAppName, TEXT ("DIB Sequential Display"),
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     NULL, NULL, hInstance, NULL) ;

      ShowWindow (hwnd, iCmdShow) ;
```

```
     UpdateWindow (hwnd) ;

     hAccel = LoadAccelerators (hInstance, szAppName) ;
     while (GetMessage (&msg, NULL, 0, 0))
     {
                    if (!TranslateAccelerator (hwnd, hAccel, &msg))
                    {
                                   TranslateMessage (&msg) ;
                                   DispatchMessage (&msg) ;
                    }
     }
     return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
     static BITMAPINFO      *     pbmi ;
     static BYTE            *     pBits ;
     static int                   cxDib, cyDib, cBits ;
     static OPENFILENAME ofn ;
     static TCHAR                 szFileName  [MAX_PATH],  szTitleName
[MAX_PATH] ;
     static TCHAR                 szFilter[]=  TEXT  ("Bitmap  Files
(*.BMP)\0*.bmp\0")
      TEXT ("All Files (*.*)\0*.*\0\0") ;
     BITMAPFILEHEADER             bmfh ;
     BOOL                         bSuccess, bTopDown ;
     DWORD                        dwBytesRead ;
     HANDLE                       hFile ;
     HDC                          hdc ;
     HMENU                        hMenu ;
     int                          iInfoSize, iBitsSize, iRowLength, y ;
     PAINTSTRUCT                  ps ;

     switch (message)
     {
     case  WM_CREATE:
                    ofn.lStructSize           = sizeof (OPENFILENAME) ;
                    ofn.hwndOwner          = hwnd ;
                    ofn.hInstance             = NULL ;
                    ofn.lpstrFilter           = szFilter ;
                    ofn.lpstrCustomFilter    = NULL ;
                    ofn.nMaxCustFilter       = 0 ;
                    ofn.nFilterIndex          = 0 ;
                    ofn.lpstrFile             = szFileName ;
                    ofn.nMaxFile             = MAX_PATH ;
                    ofn.lpstrFileTitle        = szTitleName ;
                    ofn.nMaxFileTitle         = MAX_PATH ;
```

```
                    ofn.lpstrInitialDir        = NULL ;
                    ofn.lpstrTitle             = NULL ;
                    ofn.Flags                   = 0 ;
                    ofn.nFileOffset            = 0 ;
                    ofn.nFileExtension        = 0 ;
                    ofn.lpstrDefExt            = TEXT ("bmp") ;
                    ofn.lCustData              = 0 ;
                    ofn.lpfnHook               = NULL ;
                    ofn.lpTemplateName     = NULL ;
                    return 0 ;

     case  WM_COMMAND:
                    hMenu = GetMenu (hwnd) ;

                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_OPEN:
                                        // Display File Open dialog
                                  if (!GetOpenFileName (&ofn))
                                        return 0 ;

                                        // Get rid of old DIB

                                  if (pbmi)
                                  {
                                        free (pbmi) ;
                                        pbmi = NULL ;
                                  }

                                  if (pBits)
                                  {
                                        free (pBits) ;
                                        pBits = NULL ;
                                  }

                    // Generate WM_PAINT message to erase background

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

                                        // Open the file

                                  hFile   =   CreateFile   (szFileName,
GENERIC_READ,
                        FILE_SHARE_READ, NULL, OPEN_EXISTING,
                        FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;

                                  if (hFile == INVALID_HANDLE_VALUE)
```

```
                                        {
                MessageBox ( hwnd, TEXT ("Cannot open file."),
                                        szAppName, MB_ICONWARNING | MB_OK) ;
                                   return 0 ;
                                        }

                                        // Read in the BITMAPFILEHEADER

                          bSuccess    =    ReadFile    (hFile,&bmfh,    sizeof
(BITMAPFILEHEADER),
                             &dwBytesRead, NULL) ;

                          if(!bSuccess||dwBytesRead!=sizeof(BITMAPFILEHEADER))
                          {
                    MessageBox (hwnd, TEXT ("Cannot read file."),
                              szAppName, MB_ICONWARNING | MB_OK) ;
                                CloseHandle (hFile) ;
                                return 0 ;
                     }

                                        // Check that it's a bitmap

                          if (bmfh.bfType != * (WORD *) "BM")
                          {
                    MessageBox (hwnd, TEXT ("File is not a bitmap."),
                              szAppName, MB_ICONWARNING | MB_OK) ;
                                CloseHandle (hFile) ;
                                return 0 ;
                          }

                                        // Allocate memory for header and bits

               iInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
                     iBitsSize = bmfh.bfSize - bmfh.bfOffBits ;

                          pbmi  = malloc (iInfoSize) ;
                          pBits       = malloc (iBitsSize) ;

                          if (pbmi == NULL || pBits == NULL)
                                        {
                    MessageBox (hwnd, TEXT ("Cannot allocate memory."),
                              szAppName, MB_ICONWARNING | MB_OK) ;
                                if (pbmi)
                                free (pbmi) ;
                                if (pBits)
                                free (pBits) ;
                                CloseHandle (hFile) ;
                    return 0 ;
```

```
                                    }

                    // Read in the Information Header

          bSuccess = ReadFile (hFile, pbmi, iInfoSize,    &dwBytesRead,
NULL) ;

                                    if (!bSuccess || (int) dwBytesRead != iInfoSize)
                                    {
                     MessageBox (hwnd, TEXT ("Cannot read file."),
                                szAppName, MB_ICONWARNING | MB_OK) ;
                                    if (pbmi)
                                    free (pbmi) ;
                                    if (pBits)
                                    free (pBits) ;
                                    CloseHandle (hFile) ;
                    return 0 ;
                                    }

                    // Get the DIB width and height

                                    bTopDown = FALSE ;

                       if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
                                    {
                    cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
                    cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
                    cBits = ((BITMAPCOREHEADER *) pbmi)->bcBitCount ;
                                            }
                                    else
                                            {
                                    if (pbmi->bmiHeader.biHeight < 0)
                                                bTopDown = TRUE ;

                    cxDib =           pbmi->bmiHeader.biWidth ;
                    cyDib = abs  (pbmi->bmiHeader.biHeight) ;
                    cBits =           pbmi->bmiHeader.biBitCount ;

                    if (pbmi->bmiHeader.biCompression != BI_RGB &&
                       pbmi->bmiHeader.biCompression != BI_BITFIELDS)
                     {
                    MessageBox (hwnd, TEXT ("File is compressed."),
                                szAppName, MB_ICONWARNING | MB_OK) ;
                        if (pbmi)
                        free (pbmi) ;
                        if (pBits)
                        free (pBits) ;
                        CloseHandle (hFile) ;
```

```
                return 0 ;
                 }
                                    }

                                    // Get the row length

                                iRowLength = ((cxDib * cBits + 31) & ~31) >> 3 ;

                                    // Read and display
                                    SetCursor     (LoadCursor     (NULL,
IDC_WAIT)) ;
                                    ShowCursor (TRUE) ;

                                    hdc = GetDC (hwnd) ;

                                    for (y = 0 ; y < cyDib ; y++)
                                    {
           ReadFile (hFile, pBits + y * iRowLength,iRowLength,&dwBytesRead,
NULL) ;
SetDIBitsToDevice (hdc,
               0,                       // xDst
               0,                       // yDst
               cxDib,              // cxSrc
               cyDib,              // cySrc
               0,                       // xSrc
               0,                       // ySrc
               bTopDown ? cyDib - y - 1 : y,
                            // first scan line
                 1,                     // number of scan lines
                 pBits + y * iRowLength,
                 pbmi,
                 DIB_RGB_COLORS) ;
                       }
                       ReleaseDC (hwnd, hdc) ;
                       CloseHandle (hFile) ;
                       ShowCursor (FALSE) ;
                       SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
                                  return 0 ;
                       }
                       break ;

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

                       if (pbmi && pBits)
                                       SetDIBitsToDevice (hdc,
                          0,                          // xDst
                          0,                          // yDst
```

```
                              cxDib,                    // cxSrc
                              cyDib,                    // cySrc
                              0,                        // xSrc
                              0,                        // ySrc
                              0,                        // first scan line
                              cyDib,                    // number of scan lines
                              pBits,
                    DIB_RGB_COLORS) ;
                         EndPaint (hwnd, &ps) ;
                         return 0 ;

     case  WM_DESTROY:
                         if (pbmi)
                                             free (pbmi) ;

                         if (pBits)
                                             free (pBits) ;

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

SEQDISP.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Accelerator
SEQDISP ACCELERATORS DISCARDABLE
BEGIN
     "O",     IDM_FILE_OPEN,      VIRTKEY, CONTROL, NOINVERT
END

//////////////////////////////////////////////////////////////////////////////
/
// Menu
SEQDISP MENU DISCARDABLE
BEGIN
     POPUP "&File"
     BEGIN
      MENUITEM "&Open...\tCtrl+O",  IDM_FILE_OPEN
     END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
```

```
// Used by SeqDisp.rc

#define IDM_FILE_OPEN            40001
```

在处理「File Open」功能表命令期间，在 SEQDISP.C 内的所有档案 I/O 都会发生。在处理 WM_COMMAND 的最後，程式进入读取单行图素并用 SetDIBitsToDevice 显示该行图素的回圈。整个 DIB 储存在记忆体中以便在处理 WM_PAINT 期间也能显示它。

缩放到合适尺寸

SetDIBitsToDevice 完成了将 DIB 的图素对点送入输出设备的显示程序。这对於列印 DIB 用处不大。印表机的解析度越高，得到的图像就越小，您最终会得到如邮票大小的图像。

要通过缩小或放大 DIB，在输出设备上以特定的大小显示它，可以使用 StretchDIBits：

```
iLines = StretchDIBits (
                              hdc,              // device context handle
                              xDst,             // x destination coordinate
                              yDst,             // y destination coordinate
                              cxDst,            // destination rectangle width
                              cyDst,            // destination rectangle height
                              xSrc,             // x source coordinate
                              ySrc,             // y source coordinate
                              cxSrc,            // source rectangle width
                              cySrc,            // source rectangle height
                              pBits,            // pointer to DIB pixel bits
                              pInfo,            // pointer to DIB information
                              fClrUse,          // color use flag
                              dwRop) ;    // raster operation
```

- 函式参数除了下列三个方面，均与 SetDIBitsToDevice 相同。
- 目的座标包括逻辑宽度(cxDst)和高度(cyDst)，以及开始点。
- 不能通过持续显示 DIB 来减少记忆体需求。

最後一个参数是位元映射操作方式，它指出了 DIB 图素与输出设备图素结合的方式，在最後一章将学到这些内容。现在我们为此参数设定为 SRCCOPY。

还有另一个更细微的差别。如果查看 SetDIBitsToDevice 的宣告，您会发现 cxSrc 和 cySrc 是 DWORD,这是 32 位元无正负号长整数型态。在 StretchDIBits 中，cxSrc 和 cySrc（以及 cxDst 和 cyDst）定义为带正负号的整数型态，这意味著它们可以为负数，实际上等一下就会看到，它们确实能为负数。如果您已经开始检查是否别的参数也可以为负数，就让我声明一下：在两个函式中，xSrc 和 ySrc 均定义为 int，但这是错的，这些值始终是非负数。

DIB 内的来源矩形被映射到目的矩形的座标显示如表 15-4 所示。

表 15-4

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cyDst - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxDst - 1, yDst + cyDst - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxDst - 1, yDst)

右列中的-1 项是不精确的，因为放大的程度（以及映射方式和其他变换）能产生略微不同的结果。

例如，考虑一个 2×2 的 DIB，这里 StretchDIBits 的 xSrc 和 ySrc 参数均为 0，cxSrc 和 cySrc 均为 2。假定我们显示到的装置内容具有 MM_TEXT 映射方式并且不进行变换。如果 xDst 和 yDst 均为 0，cxDst 和 cyDst 均为 4，那么我们将以倍数 2 放大 DIB。每个来源图素（x,y）将映射到下面所示的四个目的图素上：

```
(0,0) --> (0,2) and (1,2) and (0,3) and (1,3)
(1,0) --> (2,2) and (3,2) and (2,3) and (3,3)
(0,1) --> (0,0) and (1,0) and (0,1) and (1,1)
(1,1) --> (2,0) and (3,0) and (2,1) and (3,1)
```

上表正确地指出了目的的角，(0,3)、(3,3)、(0,0)和(3,0)。在其他情况下，座标可能是个大概值。

目的装置内容的映射方式对 SetDIBitsToDevice 的影响仅是由於 xDst 和 yDst 是逻辑座标。StretchDIBits 完全受映射方式的影响。例如，如果您设定了 y 值向上递增的一种度量映射方式，DIB 就会颠倒显示。

您可以通过把 cyDst 设定为负数来弥补这种情况。实际上，您可以将任何参数的宽度和高度变为负值来水平或垂直·转 DIB。在 MM_TEXT 映射方式下，如果 cySrc 和 cyDst 符号相反，DIB 会沿著水平轴·转并颠倒显示。如果 cxSrc 和 cxDst 符号相反，DIB 会沿著垂直轴·转并显示它的镜面图像。

下面是总结这些内容的运算式，xMM 和 yMM 指出映射方式的方向，如果 x 值向右增长，则 xMM 值为 1；如果 x 值向左增长，则值为-1。同样，如果 y 值向下增长，则 yMM 值为 1；如果 y 值向上增长，则值为-1。Sign 函式对於正值传回 TURE，对於负值传回 FALSE。

```
if (!Sign (xMM × cxSrc × cxDst))
      DIB is flipped on its vertical axis (mirror image)
if (!Sign (yMM × cySrc × cyDst))
      DIB is flipped on its horizontal axis (upside down)
```

若有疑问，请查阅表 15-4。

程式 15-5 SHOWDIB 以实际尺寸显示 DIB、放大至显示区域视窗的大小、列印 DIB 以及把 DIB 传输到剪贴簿。

程式 15-5 SHOWDIB

```
SHOWDIB2.C
/*------------------------------------------------------------------------
      SHOWDIB2.C --     Shows a DIB in the client area
                        (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
      HACCEL            hAccel ;
      HWND              hwnd ;
      MSG                     msg ;
      WNDCLASS          wndclass ;
      wndclass.style              = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc   = WndProc ;
      wndclass.cbClsExtra    = 0 ;
      wndclass.cbWndExtra    = 0 ;
      wndclass.hInstance     = hInstance ;
      wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName  = szAppName ;
      wndclass.lpszClassName = szAppName ;

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

      hwnd = CreateWindow (szAppName, TEXT ("Show DIB #2"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;
```

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

      hAccel = LoadAccelerators (hInstance, szAppName) ;
      while (GetMessage (&msg, NULL, 0, 0))
     {
                         if (!TranslateAccelerator (hwnd, hAccel, &msg))
                         {
                                        TranslateMessage (&msg) ;
                                        DispatchMessage (&msg) ;
                         }
     }
      return msg.wParam ;
}

int ShowDib (HDC hdc, BITMAPINFO * pbmi, BYTE * pBits, int cxDib, int cyDib,
                              int cxClient, int cyClient, WORD wShow)
{
      switch (wShow)
      {
      case  IDM_SHOW_NORMAL:
                         return SetDIBitsToDevice (hdc, 0, 0, cxDib, cyDib, 0, 0,
                        0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

      case  IDM_SHOW_CENTER:
                         return SetDIBitsToDevice (hdc, (cxClient - cxDib) / 2,
                         (cyClient - cyDib) / 2,
                  cxDib, cyDib, 0, 0, 0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

      case  IDM_SHOW_STRETCH:
                         SetStretchBltMode (hdc, COLORONCOLOR) ;
return StretchDIBits(hdc,0, 0, cxClient, cyClient, 0, 0, cxDib, cyDib,
                          pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;

      case  IDM_SHOW_ISOSTRETCH:
                         SetStretchBltMode (hdc, COLORONCOLOR) ;
                         SetMapMode (hdc, MM_ISOTROPIC) ;
                         SetWindowExtEx (hdc, cxDib, cyDib, NULL) ;
                         SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
                         SetWindowOrgEx (hdc, cxDib / 2, cyDib / 2, NULL) ;
                         SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

return StretchDIBits(hdc,0, 0, cxDib, cyDib, 0, 0, cxDib, cyDib,
                                    pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;
            }
}
```

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
     static BITMAPFILEHEADER      *     pbmfh ;
     static BITMAPINFO      *     pbmi ;
     static BYTE            *     pBits ;
     static DOCINFO               di = {sizeof (DOCINFO),
                            TEXT ("ShowDib2: Printing") } ;
     static     int              cxClient, cyClient, cxDib, cyDib ;
     static     PRINTDLG            printdlg = { sizeof (PRINTDLG) } ;
     static     TCHAR                    szFileName [MAX_PATH], szTitleName
[MAX_PATH] ;
     static     WORD             wShow = IDM_SHOW_NORMAL ;
     BOOL                    bSuccess ;
     HDC                     hdc, hdcPrn ;
     HGLOBAL                 hGlobal ;
     HMENU                   hMenu ;
     int                     cxPage, cyPage, iEnable ;
     PAINTSTRUCT              ps ;
     BYTE                    * pGlobal ;

     switch (message)
     {
     case  WM_CREATE:
                     DibFileInitialize (hwnd) ;
                     return 0 ;

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

     case  WM_INITMENUPOPUP:
                     hMenu = GetMenu (hwnd) ;

                     if (pbmfh)
                                     iEnable = MF_ENABLED ;
                     else
                                     iEnable = MF_GRAYED ;

                     EnableMenuItem   (hMenu,               IDM_FILE_SAVE,
     iEnable) ;
                     EnableMenuItem   (hMenu,              IDM_FILE_PRINT,
     iEnable) ;
                     EnableMenuItem   (hMenu,                IDM_EDIT_CUT,
     iEnable) ;
                     EnableMenuItem   (hMenu,               IDM_EDIT_COPY,
     iEnable) ;
                     EnableMenuItem   (hMenu, IDM_EDIT_DELETE, iEnable) ;
```

```
                    return 0 ;

     case  WM_COMMAND:
                    hMenu = GetMenu (hwnd) ;

                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_OPEN:
                              // Show the File Open dialog box

                         if    (!DibFileOpenDlg    (hwnd,    szFileName,
szTitleName))
                                        return 0 ;

                              // If there's an existing DIB, free the memory
                                   if (pbmfh)
                                   {
                                        free (pbmfh) ;
                                        pbmfh = NULL ;
                                   }
                              // Load the entire DIB into memory

               SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
               ShowCursor (TRUE) ;

                              pbmfh = DibLoadImage (szFileName) ;

                              ShowCursor (FALSE) ;
                              SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                         // Invalidate the client area for later update

                              InvalidateRect (hwnd, NULL, TRUE) ;

                              if (pbmfh == NULL)
                              {
                        MessageBox (      hwnd, TEXT ("Cannot load DIB file"),
                              szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                                   return 0 ;
                              }
               // Get pointers to the info structure & the bits

                              pbmi  = (BITMAPINFO *) (pbmfh + 1) ;
                              pBits       = (BYTE *) pbmfh + pbmfh->bfOffBits ;

               // Get the DIB width and height

                         if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
```

```
                              {
                         cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
                         cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
                              }
                                   else
                              {
                         cxDib =    pbmi->bmiHeader.biWidth ;
                         cyDib = abs (pbmi->bmiHeader.biHeight) ;
                              }
                                   return 0 ;

                  case  IDM_FILE_SAVE:
                                        // Show the File Save dialog box

                                   if    (!DibFileSaveDlg    (hwnd,    szFileName,
szTitleName))
                         return 0 ;
                                   // Save the DIB to a disk file

                                   SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                                   ShowCursor (TRUE) ;

                                   bSuccess = DibSaveImage (szFileName, pbmfh) ;

                                   ShowCursor (FALSE) ;
                                   SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                                   if (!bSuccess)
                MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
                                        szAppName,    MB_ICONEXCLAMATION    |
MB_OK) ;
                                   return 0 ;

                   case  IDM_FILE_PRINT:
                                   if (!pbmfh)
                                        return 0 ;

                                        // Get printer DC

            printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

                                   if (!PrintDlg (&printdlg))
                                             return 0 ;

                                   if    (NULL == (hdcPrn = printdlg.hDC))
                                   {
             MessageBox (    hwnd, TEXT ("Cannot obtain Printer DC"),
                                   szAppName, MB_ICONEXCLAMATION | MB_OK) ;
```

```
                                        return 0 ;
                                   }

                    // Check whether the printer can print bitmaps

               if   (!(RC_BITBLT & GetDeviceCaps (hdcPrn, RASTERCAPS)))
                                   {
                                   DeleteDC (hdcPrn) ;
              MessageBox (hwnd, TEXT ("Printer cannot print bitmaps"),
                          szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                                        return 0 ;
                                   }
                                        // Get size of printable area of page

                                   cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
                                   cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

                                   bSuccess = FALSE ;
                                        // Send the DIB to the printer

                                   SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                              ShowCursor (TRUE) ;

if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
                                   {
                    ShowDib (   hdcPrn, pbmi, pBits, cxDib, cyDib,
                         cxPage, cyPage, wShow) ;

                                        if (EndPage (hdcPrn) > 0)
                                        {
                                                  bSuccess = TRUE ;
                                                  EndDoc (hdcPrn) ;
                                        }
                                   }
                                   ShowCursor (FALSE) ;
                                   SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                                   DeleteDC (hdcPrn) ;

                                   if (!bSuccess)
                MessageBox (hwnd, TEXT ("Could not print bitmap"),
                                   szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                                   return 0 ;

                    case  IDM_EDIT_COPY:
                    case  IDM_EDIT_CUT:
                                   if (!pbmfh)
                                        return 0 ;
```

```
                                    // Make a copy of the packed DIB

     hGlobal = GlobalAlloc (GHND | GMEM_SHARE, pbmfh->bfSize -
                           sizeof (BITMAPFILEHEADER)) ;

                                pGlobal = GlobalLock (hGlobal) ;

      CopyMemory ( pGlobal, (BYTE *) pbmfh + sizeof (BITMAPFILEHEADER),
             pbmfh->bfSize - sizeof (BITMAPFILEHEADER)) ;

                                GlobalUnlock (hGlobal) ;

                                      // Transfer it to the clipboard

                                OpenClipboard (hwnd) ;
                                EmptyClipboard () ;
                                SetClipboardData (CF_DIB, hGlobal) ;
                                CloseClipboard () ;

                                if (LOWORD (wParam) == IDM_EDIT_COPY)
                                      return 0 ;
                        // fall through if IDM_EDIT_CUT
               case  IDM_EDIT_DELETE:
                                if (pbmfh)
                                {
                                free (pbmfh) ;
                                pbmfh = NULL ;
                                InvalidateRect (hwnd, NULL, TRUE) ;
                                }
                                return 0 ;

               case  IDM_SHOW_NORMAL:
               case  IDM_SHOW_CENTER:
               case  IDM_SHOW_STRETCH:
               case  IDM_SHOW_ISOSTRETCH:
                                CheckMenuItem (hMenu, wShow, MF_UNCHECKED) ;
                                wShow = LOWORD (wParam) ;
                                CheckMenuItem (hMenu, wShow, MF_CHECKED) ;
                                InvalidateRect (hwnd, NULL, TRUE) ;
                                return 0 ;
               }
               break ;

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

                     if (pbmfh)
```

```
                ShowDib (   hdc, pbmi, pBits, cxDib, cyDib,
                 cxClient, cyClient, wShow) ;

                     EndPaint (hwnd, &ps) ;
                     return 0 ;

     case  WM_DESTROY:
                     if (pbmfh)
                                free (pbmfh) ;

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

SHOWDIB2.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Menu
SHOWDIB2 MENU DISCARDABLE
BEGIN
     POPUP "&File"
     BEGIN
    MENUITEM "&Open...\tCtrl+O",IDM_FILE_OPEN
    MENUITEM "&Save...\tCtrl+S",  IDM_FILE_SAVE
    MENUITEM SEPARATOR
    MENUITEM "&Print\tCtrl+P",    IDM_FILE_PRINT
     END
     POPUP "&Edit"
     BEGIN
    MENUITEM          "Cu&t\tCtrl+X", IDM_EDIT_CUT
    MENUITEM          "&Copy\tCtrl+C", IDM_EDIT_COPY
    MENUITEM          "&Delete\tDelete", IDM_EDIT_DELETE
     END
     POPUP "&Show"
     BEGIN
    MENUITEM          "&Actual Size",  IDM_SHOW_NORMAL, CHECKED
    MENUITEM          "&Center",       IDM_SHOW_CENTER
    MENUITEM          "&Stretch to Window", IDM_SHOW_STRETCH
    MENUITEM          "Stretch &Isotropically",    IDM_SHOW_ISOSTRETCH
     END
END

//////////////////////////////////////////////////////////////////////////////
```

```
/
// Accelerator
SHOWDIB2 ACCELERATORS DISCARDABLE
BEGIN
          "C",   IDM_EDIT_COPY,        VIRTKEY, CONTROL, NOINVERT
       "O",   IDM_FILE_OPEN,        VIRTKEY, CONTROL, NOINVERT
      "P",   IDM_FILE_PRINT,       VIRTKEY, CONTROL, NOINVERT
      "S",   IDM_FILE_SAVE,        VIRTKEY, CONTROL, NOINVERT
      VK_DELETE,      IDM_EDIT_DELETE,      VIRTKEY, NOINVERT
      "X",   IDM_EDIT_CUT,         VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib2.rc

#define IDM_FILE_OPEN        40001
#define IDM_SHOW_NORMAL      40002
#define IDM_SHOW_CENTER      40003
#define IDM_SHOW_STRETCH     40004
#define IDM_SHOW_ISOSTRETCH  40005
#define IDM_FILE_PRINT       40006
#define IDM_EDIT_COPY        40007
#define IDM_EDIT_CUT         40008
#define IDM_EDIT_DELETE      40009
#define IDM_FILE_SAVE        40010
```

有意思的是 ShowDib 函式，它依赖於功能表选择以四种不同的方式之一在程式的显示区域显示 DIB。可以使用 SetDIBitsToDevice 从显示区域的左上角或在显示区域的中心显示 DIB。程式也有两个使用 StretchDIBits 的选项，DIB 能放大填充整个显示区域。在此情况下它可能会变形，或它能等比例显示，也就是说不会变形。

把 DIB 复制到剪贴簿包括：在整体共用记忆体中制作 packed DIB 记忆体块的副本。剪贴簿资料型态为 CF_DIB。程式没有列出从剪贴簿复制 DIB 的方法，因为在仅有指向 packed DIB 的指标的情况下这样做需要更多步骤来确定图素位元的偏移量。我将在下一章的末尾示范如何做到这点的方法。

您可能注意到了 SHOWDIB2 中的一些不足之处。如果您以 256 色显示模式执行 Windows，就会看到显示除了单色或 4 位元 DIB 以外的其他图形出现的问题，您看不到真正的颜色。存取那些颜色需要使用调色盘，在下一章会做这些工作。您也可能注意到速度问题，尤其在 Windows NT 下执行 SHOWDIB2 时。在下一章 packed DIB 和点阵图时，我会展示处理的方法。我也给 DIB 显示添加卷动列，这样也能以实际尺寸查看大於萤幕的 DIB。

色彩转换、调色盘和显示效能

记得在虎豹小霸王编剧 William Goldman 的另一出电影剧本《All the President's Men》中，Deep Throat 告诉 Bob Woodward 揭开水门秘密的关键是「跟著钱走」。那么在点阵图显示中获得高级性能的关键就是「跟著图素位元走」以及理解色彩转换发生的时机。DIB 是装置无关的格式，视讯显示器记忆体几乎总是与图素格式不同。在 SetDIBitsToDevice 或 StretchDIBits 函式呼叫期间，每个图素（可能有几百万个）必须从装置无关的格式转换成设备相关格式。

在许多情况下，这种转换是很繁琐的。例如，在 24 位元视讯显示器上显示 24 位元 DIB，显示驱动程式最多是切换红、绿、蓝的位元组顺序而已。在 24 位元设备上显示 16 位元 DIB 就需要位元的搬移和修剪了。在 24 位元设备上显示 4 位元或 8 位元 DIB 要求在 DIB 色彩对照表内查找 DIB 图素位元，然後对位元组重新排列。

但是要在 4 位元或 8 位元视讯显示器上显示 16 位元、24 位元或 32 位元 DIB 时，会发生什么事情呢？一种完全不一样的颜色转换发生了。对於 DIB 内的每个图素，装置驱动程式必须在图素和显示器上可用的颜色之间「找寻最接近的色彩」，这包括回圈和计算。（GDI 函式 GetNearestColor 进行「最接近色彩搜寻」。）

整个 RGB 色彩的三维阵列可用立方体表示。曲线内任意两点之间的距离是：

$$\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

在这里两个颜色是 R1G1B1 和 R2G2B2。执行最接近色彩搜寻包括从一种颜色到其他颜色集合中找寻最短距离。幸运的是，在 RGB 颜色立方体中「比较」距离时，并不需要计算平方根部分。但是需转换的每个图素必须与设备的所有颜色相比较以发现最接近的颜色。这是个工作量相当大的工作。（尽管在 8 位元设备上显示 8 位元 DIB 也得进行最接近色彩搜寻，但它不必对每个图素都进行，它仅需对 DIB 色彩对照表中的每种颜色进行寻找。）

正是由於以上原因，应该避免使用 SetDIBitsToDevice 或 StretchDIBits 在 8 位元视讯显示卡上显示 16 位元、24 位元或 32 位元 DIB。DIB 应转换为 8 位元 DIB，或者 8 位元 DDB，以求得更好的显示效能。实际上，您可以经由将 DIB 转换为 DDB 并使用 BitBlt 和 StretchBlt 显示图像，来加快显示任何 DIB 的速

度。

如果在 8 位元视讯显示器上执行 Windows（或仅仅切换到 8 位元模式来观察在显示 True-ColorDIB 时的效能变化），您可能会注意到另一个问题：DIB 不会使用所有颜色来显示。任何在 8 位元视讯显示器上的 DIB 刚好限制在以 20 种颜色显示。如何获得多於 20 种颜色是「调色盘管理器」的任务，这将在下一章提到。

最後，如果在同一台机器上执行 Windows 98 和 Windows NT，您可能会注意到：对於同样的显示模式，Windows NT 显示大型 DIB 花费的时间较长。这是 Windows NT 的客户/伺服器体系结构的结果，它使大量资料在传输给 API 函式时耗费更多时间。解决方法是将 DIB 转换为 DDB。而我等一下将谈到的 CreateDIBSection 函式对这种情况特别有用。

DIB 和 DDB 的结合

您可以做许多事情去发掘 DIB 的格式，并呼叫两个 DIB 绘图函式：SetDIBitsToDevice 和 StretchDIBits。您可以直接存取 DIB 中的各个位元、位元组和图素，且一旦您有了一堆能让您以结构化的方式检查和更改资料的函式，您要怎么处理 DIB 就没人管了。

实际上，我们发现还是有一些限制。在上一章，我们了解了使用 GDI 函式在 DDB 上绘制图像的方法。到目前为止，还没有在 DIB 上绘图的方法。另一个问题是 SetDIBitsToDevice 和 StretchDIBits 没有 BitBlt 和 StretchBlt 速度快，尤其在 Windows NT 环境下以及执行许多最接近颜色搜寻（例如，在 8 位元视频卡上显示 24 位元 DIB）时。

因此，在 DIB 和 DDB 之间进行转换是有好处的。例如，如果我们有一个需要在萤幕上显示许多次的 DIB，那么把 DIB 转换为 DDB 就很有意义，这样我们就能够使用快速的 BitBlt 和 StretchBlt 函式来显示它了。

从 DIB 建立 DDB

从 DIB 中建立 GDI 点阵图物件可能吗？基本上我们已经知道了方法：如果有 DIB，您就能够使用 CreateCompatibleBitmap 来建立与 DIB 大小相同并与视讯显示器相容的 GDI 点阵图物件。然後将该点阵图物件选入记忆体装置内容并呼叫 SetDIBitsToDevice 在那个记忆体 DC 上绘图。结果就是 DDB 具有与 DIB 相同的图像，但具有与视讯显示器相容的颜色组织。

您也可以通过呼叫 CreateDIBitmap 用几个步骤完成上述工作。函式的语法为：

```
hBitmap = CreateDIBitmap (
                hdc,            // device context handle

                                   pInfoHdr,              //  pointer  to  DIB
information header
                    fInit,         // 0 or CBM_INIT
                    pBits,         // pointer to DIB pixel bits
                    pInfo,         // pointer to DIB information
                    fClrUse) ;     // color use flag
```

请注意 pInfoHdr 和 pInfo 这两个参数，它们分别定义为指向 BITMAPINFOHEADER 结构和 BITMAPINFO 结构的指标。正如我们所知，BITMAPINFO 结构是後面紧跟色彩对照表的 BITMAPINFOHEADER 结构。我们一会儿会看到这种区别所起的作用。最後一个参数是 DIB_RGB_ COLORS(等於 0)或 DIB_PAL_COLORS，它们在 SetDIBitsToDevice 函式中使用。下一章我将讨论更多这方面的内容。

理解 Windows 中点阵图函式的作用是很重要的。不要考虑 CreateDIBitmap 函式的名称，它不建立与「装置无关的点阵图」，它从装置无关的规格中建立「设备相关的点阵图」。注意该函式传回 GDI 点阵图物件的代号，CreateBitmap、CreateBitmapIndirect 和 CreateCompatibleBitmap 也与它一样。

呼叫 CreateDIBitmap 函式最简单的方法是：

```
hBitmap = CreateDIBitmap (NULL, pbmih, 0, NULL, NULL, 0) ;
```

唯一的参数是指向 BITMAPINFOHEADER 结构（不带色彩对照表）的指标。在这个形式中，函式建立单色 GDI 点阵图物件。第二种简单的方法是：

```
hBitmap = CreateDIBitmap (hdc, pbmih, 0, NULL, NULL, 0) ;
```

在这个形式中，函式建立了与装置内容相容的 DDB，该装置内容由 hdc 参数指出。到目前为止，我们都是透过 CreateBitmap（建立单色点阵图）或 CreateCompatibleBitmap（建立与视讯显示器相容的点阵图）来完成一些工作。

在 CreateDIBitmap 的这两个简化模式中，图素还未被初始化。如果 CreateDIBitmap 的第三个参数是 CBM_INIT，Windows 就会建立 DDB 并使用最後三个参数初始化点阵图位元。pInfo 参数是指向包括色彩对照表的 BITMAPINFO 结构的指标。pBits 参数是指向由 BITMAPINFO 结构指出的色彩格式中的位元阵列的指标，根据色彩对照表这些位元被转换为设备的颜色格式，这与 SetDIBitsToDevice 的情况相同。实际上，整个 CreateDIBitmap 函式可以用下列程式码来实作：

```
HBITMAP CreateDIBitmap (     HDC hdc, CONST BITMAPINFOHEADER * pbmih,
                    DWORD fInit, CONST VOID * pBits,
                    CONST BITMAPINFO * pbmi, UINT fUsage)
{
      HBITMAP     hBitmap ;
      HDC               hdc ;
```

```
     int               cx, cy, iBitCount ;

     if (pbmih->biSize == sizeof (BITMAPCOREHEADER))
     {
                       cx   = ((PBITMAPCOREHEADER) pbmih)->bcWidth ;
                       cy   = ((PBITMAPCOREHEADER) pbmih)->bcHeight ;
                       iBitCount   = ((PBITMAPCOREHEADER) pbmih)->bcBitCount ;
     }
     else
    {
                       cx  = pbmih->biWidth ;
                       cy  = pbmih->biHeight ;
                       iBitCount   = pbmih->biBitCount ;
    }
     if (hdc)
                       hBitmap = CreateCompatibleBitmap (hdc, cx, cy) ;
     else
                       hBitmap = CreateBitmap (cx, cy, 1, 1, NULL) ;
     if    (fInit == CBM_INIT)
     {
                       hdcMem = CreateCompatibleDC (hdc) ;
                       SelectObject (hdcMem, hBitmap) ;
                       SetDIBitsToDevice (    hdcMem, 0, 0, cx, cy, 0, 0, 0 cy,
               pBits, pbmi, fUsage) ;
                       DeleteDC (hdcMem) ;
    }

return hBitmap ;
}
```

如果仅需显示 DIB 一次，并担心 SetDIBitsToDevice 显示太慢，则呼叫 CreateDIBitmap 并使用 BitBlt 或 StretchBlt 来显示 DDB 就没有什么意义。因为 SetDIBitsToDevice 和 CreateDIBitmap 都执行颜色转换，这两个工作会占用同样长的时间。只有在多次显示 DIB 时（例如在处理 WM_PAINT 讯息时）进行这种转换才有意义。

程式 15-6 DIBCONV 展示了利用 SetDIBitsToDevice 把 DIB 档案转换为 DDB 的方法。

程式 15-6 DIBCONV

```
DIBCONV.C
/*------------------------------------------------------------------------
     DIBCONV.C --      Converts a DIB to a DDB
                       (c) Charles Petzold, 1998
------------------------------------------------------------------------
*/

#include <windows.h>
```

```
#include <commdlg.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("DibConv") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
      HWND             hwnd ;
      MSG                    msg ;
      WNDCLASS         wndclass ;

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

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

      hwnd = CreateWindow (szAppName, TEXT ("DIB to DDB Conversion"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

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

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

      }
      return msg.wParam ;
```

```
}
HBITMAP CreateBitmapObjectFromDibFile (HDC hdc, PTSTR szFileName)
{
      BITMAPFILEHEADER *     pbmfh ;
      BOOL             bSuccess ;
      DWORD            dwFileSize, dwHighSize, dwBytesRead ;
      HANDLE           hFile ;
      HBITMAP          hBitmap ;

                       // Open the file: read access, prohibit write access

      hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
       OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
      if     (hFile == INVALID_HANDLE_VALUE)
                             return NULL ;

                       // Read in the whole file

      dwFileSize = GetFileSize (hFile, &dwHighSize) ;

      if (dwHighSize)
      {
                       CloseHandle (hFile) ;
                       return NULL ;
      }

      pbmfh = malloc (dwFileSize) ;

      if (!pbmfh)
      {
                       CloseHandle (hFile) ;
                       return NULL ;
    }

      bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
      CloseHandle (hFile) ;

                       // Verify the file
      if (!bSuccess          || (dwBytesRead != dwFileSize)
                             || (pbmfh->bfType != * (WORD *) "BM")
                             || (pbmfh->bfSize != dwFileSize))
    {
                       free (pbmfh) ;
                       return NULL ;
    }
                       // Create the DDB
      hBitmap = CreateDIBitmap     (hdc,
                                          (BITMAPINFOHEADER *) (pbmfh + 1),
```

```
                                    CBM_INIT,
                                         (BYTE *) pbmfh + pbmfh->bfOffBits,
                                         (BITMAPINFO *) (pbmfh + 1),
                                         DIB_RGB_COLORS) ;
     free (pbmfh) ;
     return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
     static HBITMAP         hBitmap ;
     static int             cxClient, cyClient ;
     static OPENFILENAME    ofn ;
     static TCHAR  szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
     static TCHAR  szFilter[]=TEXT("Bitmap Files(*.BMP)\0*.bmp\0")
      TEXT ("All Files (*.*)\0*.*\0\0") ;
     BITMAP                         bitmap ;
     HDC                            hdc, hdcMem ;
     PAINTSTRUCT                    ps ;

     switch (message)
     {
     case  WM_CREATE:
                    ofn.lStructSize             = sizeof (OPENFILENAME) ;
                    ofn.hwndOwner          = hwnd ;
                    ofn.hInstance               = NULL ;
                    ofn.lpstrFilter             = szFilter ;
                    ofn.lpstrCustomFilter  = NULL ;
                    ofn.nMaxCustFilter     = 0 ;
                    ofn.nFilterIndex            = 0 ;
                    ofn.lpstrFile               = szFileName ;
                    ofn.nMaxFile                = MAX_PATH ;
                    ofn.lpstrFileTitle          = szTitleName ;
                    ofn.nMaxFileTitle           = MAX_PATH ;
                    ofn.lpstrInitialDir                   = NULL ;
                    ofn.lpstrTitle              = NULL ;
                    ofn.Flags                   = 0 ;
                    ofn.nFileOffset             = 0 ;
                    ofn.nFileExtension          = 0 ;
                    ofn.lpstrDefExt             = TEXT ("bmp") ;
                    ofn.lCustData               = 0 ;
                    ofn.lpfnHook                = NULL ;
                    ofn.lpTemplateName     = NULL ;

                    return 0 ;

     case  WM_SIZE:
                    cxClient = LOWORD (lParam) ;
```

```
                    cyClient = HIWORD (lParam) ;
                    return 0 ;

     case  WM_COMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_OPEN:

                         // Show the File Open dialog box

                                   if (!GetOpenFileName (&ofn))
                                        return 0 ;

                         // If there's an existing DIB, delete it

                                   if (hBitmap)
                                   {
                                        DeleteObject (hBitmap) ;
                                        hBitmap = NULL ;
                                   }
                                        // Create the DDB from the DIB

                                   SetCursor     (LoadCursor     (NULL,
IDC_WAIT)) ;
                                   ShowCursor (TRUE) ;

                       hdc = GetDC (hwnd) ;
                       hBitmap   =   CreateBitmapObjectFromDibFile   (hdc,
szFileName) ;
                       ReleaseDC (hwnd, hdc) ;

                                   ShowCursor (FALSE) ;
                                   SetCursor     (LoadCursor     (NULL,
IDC_ARROW)) ;

                // Invalidate the client area for later update

                                   InvalidateRect (hwnd, NULL, TRUE) ;
                                   if (hBitmap == NULL)
                                   {
                  MessageBox (hwnd, TEXT ("Cannot load DIB file"),
                           szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                                   }
                                   return 0 ;
                    }
                    break ;

          case  WM_PAINT:
```

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

                        if (hBitmap)
                        {
                    GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

                    hdcMem = CreateCompatibleDC (hdc) ;
                    SelectObject (hdcMem, hBitmap) ;

                    BitBlt (hdc,0, 0, bitmap.bmWidth, bitmap.bmHeight,
                        hdcMem, 0, 0, SRCCOPY) ;

                                    DeleteDC (hdcMem) ;
                        }

                    EndPaint (hwnd, &ps) ;
                        return 0 ;

        case  WM_DESTROY:
                        if (hBitmap)
                            DeleteObject (hBitmap) ;

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

DIBCONV.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
/
// Menu
DIBCONV MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",       IDM_FILE_OPEN
    END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DibConv.rc

#define IDM_FILE_OPEN    40001
```

DIBCONV.C 本身就是完整的，并不需要前面的档案。在对它仅有的功能表命

令（「File Open」）的回应中，WndProc 呼叫程式的 CreateBitmapObjectFromDibFile 函式。此函式将整个档案读入记忆体并将指向记忆体块的指标传递给 CreateDIBitmap 函式，函式传回点阵图的代号，然後包含 DIB 的记忆体块被释放。在 WM_PAINT 讯息处理期间，WndProc 将点阵图选入相容的记忆体装置内容并使用 BitBlt（不是 SetDIBitsToDevice）在显示区域显示点阵图。它通过使用点阵图代号呼叫带有 BITMAP 结构的 GetObject 函式来取得点阵图的宽度和高度。

在从 CreateDIBitmap 建立点阵图时不必初始化 DDB 图素位元，之後您可以呼叫 SetDIBits 初始化图素位元。该函式的语法如下：

```
iLines = SetDIBits (
                        hdc,         // device context handle
                        hBitmap,    // bitmap handle
                        yScan,      // first scan line to convert
                        cyScans,   // number of scan lines to convert
                        pBits,      // pointer to pixel bits
                        pInfo,      // pointer to DIB information
                        fClrUse) ;  // color use flag
```

函式使用了 BITMAPINFO 结构中的色彩对照表把位元转换为设备相关的格式。只有在最後一个参数设定为 DIB_PAL_COLORS 时，才需要装置内容代号。

从 DDB 到 DIB

与 SetDIBits 函式相似的函式是 GetDIBits，您可以使用此函式把 DDB 转化为 DIB：

```
int WINAPI GetDIBits (
                  hdc,              // device context handle
                  hBitmap,    // bitmap handle
                  yScan,      // first scan line to convert
                  cyScans,    // number of scan lines to convert
                  pBits,            // pointer to pixel bits (out)
                  pInfo,            // pointer to DIB information (out)
                  fClrUse) ;        // color use flag
```

然而，此函式产生的恐怕不是 SetDIBits 的反运算结果。在一般情况下，如果使用 CreateDIBitmap 和 SetDIBits 将 DIB 转换为 DDB，然後使用 GetDIBits 把 DDB 转换回 DIB，您就不会得到原来的图像。这是因为在 DIB 被转换为设备相关的格式时，有一些资讯遗失了。遗失的资讯数量取决於进行转换时 Windows 所执行的显示模式。

您可能会发现没有使用 GetDIBits 的必要性。考虑一下：在什么环境下您的程式发现自身带有点阵图代号，但没有用於在起始的位置建立点阵图的资料？剪贴簿？但是剪贴簿为 DIB 提供了自动的转换。GetDIBits 函式的一个例子

是在捕捉萤幕显示内容的情况下，例如第十四章中 BLOWUP 程式所做的。我不示范这个函式，但在 Microsoft 网站的 Knowledge Base 文章 Q80080 中有一些资讯。

DIB 区块

我希望您已经对设备相关和装置无关点阵图的区别有了清晰的概念。DIB 能拥有几种色彩组织中的一种，DDB 必须是单色的或是与真实输出设备相同的格式。DIB 是一个档案或记忆体块；DDB 是 GDI 点阵图物件并由点阵图代号表示。DIB 能被显示或转换为 DDB 并转换回 DIB，但是这里包含了装置无关位元和设备相关位元之间的转换程序。

现在您将遇到一个函式，它打破了这些规则。该函式在 32 位元 Windows 版本中发表，称为 CreateDIBSection，语法为：

```
hBitmap = CreateDIBSection (
                hdc,        // device context handle
                pInfo,      // pointer to DIB information
                 fClrUse,         // color use flag
                 ppBits,          // pointer to pointer variable
                 hSection,        // file-mapping object handle
                 dwOffset) ;      // offset to bits in file-mapping object
```

CreateDIBSection 是 Windows API 中最重要的函式之一（至少在使用点阵图时），然而您会发现它很深奥并难以理解。

让我们从它的名称开始，我们知道 DIB 是什么，但「DIB section」到底是什么呢？当您第一次检查 CreateDIBSection 时，可能会寻找该函式与 DIB 区块工作的方式。这是正确的，CreateDIBSection 所做的就是建立了 DIB 的一部分（点阵图图素位元的记忆体块）。

现在我们看一下传回值，它是 GDI 点阵图物件的代号，这个传回值可能是该函式呼叫最会拐人的部分。传回值似乎暗示著 CreateDIBSection 在功能上与 CreateDIBitmap 相同。事实上，它只是相似但完全不同。实际上，从 CreateDIBSection 传回的点阵图代号与我们在本章和上一章遇到的所有点阵图建立函式传回的点阵图代号在本质上不同。

一旦理解了 CreateDIBSection 的真实特性，您可能觉得奇怪为什么不把传回值定义得有所区别。您也可能得出结论：CreateDIBSection 应该称之为 CreateDIBitmap，并且如同我前面所指出的 CreateDIBitmap 应该称之为 CreateDDBitmap。

首先让我们检查一下如何简化 CreateDIBSection，并正确地使用它。首先，把最後两个参数 hSection 和 dwOffset，分别设定为 NULL 和 0，我将在本章最

後讨论这些参数的用法。第二，仅在 fColorUse 参数设定为 DIB_ PAL_COLORS 时，才使用 hdc 参数，如果 fColorUse 为 DIB_RGB_COLORS（或 0），hdc 将被忽略（这与 CreateDIBitmap 不同，hdc 参数用於取得与 DDB 相容的设备的色彩格式）。

因此，CreateDIBSection 最简单的形式仅需要第二和第四个参数。第二个参数是指向 BITMAPINFO 结构的指标，我们以前曾使用过。我希望指向第四个参数的指标定义的指标不会使您困惑，它实际上很简单。

假设要建立每图素 24 位元的 384×256 位元 DIB，24 位元格式不需要色彩对照表，因此它是最简单的，所以我们可以为 BITMAPINFO 参数使用 BITMAPINFOHEADER 结构。

您需要定义三个变数：BITMAPINFOHEADER 结构、BYTE 指标和点阵图代号：

```
BITMAPINFOHEADER bmih ;
BYTE                  * pBits ;
HBITMAP               hBitmap ;
```

现在初始化 BITMAPINFOHEADER 结构的栏位：

```
bmih->biSize          = sizeof (BITMAPINFOHEADER) ;
bmih->biWidth         = 384 ;
bmih->biHeight        = 256 ;
bmih->biPlanes        = 1 ;
bmih->biBitCount      = 24 ;
bmih->biCompression   = BI_RGB ;
bmih->biSizeImage     = 0 ;
bmih->biXPelsPerMeter = 0 ;
bmih->biYPelsPerMeter = 0 ;
bmih->biClrUsed       = 0 ;
bmih->biClrImportant  = 0 ;
```

在基本准备後，我们呼叫该函式：

```
hBitmap = CreateDIBSection (NULL, (BITMAPINFO *)  &bmih, 0, &pBits, NULL, 0) ;
```

注意，我们为第二个参数赋予 BITMAPINFOHEADER 结构的位址。这是常见的，但一个 BYIE 指标 pBits 的位址，就不常见了。这样，第四个参数是函式需要的指向指标的指标。

这是函式呼叫所做的：CreateDIBSection 检查 BITMAPINFOHEADER 结构并配置足够的记忆体块来载入 DIB 图素位元。（在这个例子里，记忆体块的大小为 384×256×3 位元组。）它在您提供的 pBits 参数中储存了指向此记忆体块的指标。函式传回点阵图代号，正如我说的，它与 CreateDIBitmap 和其他点阵图建立函式传回的代号不一样。

然而，我们还没有做完，点阵图图素是未初始化的。如果正在读取 DIB 档案，可以简单地把 pBits 参数传递给 ReadFile 函式并读取它们。或者可以使用

一些程式码「人工」设定。

程式 15-7 DIBSECT 除了呼叫 CreateDIBSection 而不是 CreateDIBitmap 之外，与 DIBCONV 程式相似。

程式 15-7 DIBSECT

```
DIBSECT.C
/*-------------------------------------------------------------------------
--
      DIBSECT.C --      Displays a DIB Section in the client area
                        (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

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

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

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

      hwnd = CreateWindow (szAppName, TEXT ("DIB Section Display"),
                              WS_OVERLAPPEDWINDOW,
                              CW_USEDEFAULT, CW_USEDEFAULT,
```

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

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

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

     }
     return msg.wParam ;
}
HBITMAP CreateDIBsectionFromDibFile (PTSTR szFileName)
{
     BITMAPFILEHEADER               bmfh ;
     BITMAPINFO             *    pbmi ;
     BYTE                   *    pBits ;
     BOOL                bSuccess ;
     DWORD               dwInfoSize, dwBytesRead ;
     HANDLE              hFile ;
     HBITMAP             hBitmap ;

                  // Open the file: read access, prohibit write access

     hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ,
                         NULL, OPEN_EXISTING, 0, NULL) ;
     if (hFile == INVALID_HANDLE_VALUE)
                         return NULL ;
                         // Read in the BITMAPFILEHEADER
     bSuccess = ReadFile (hFile, &bmfh, sizeof (BITMAPFILEHEADER),
                                   &dwBytesRead, NULL) ;

     if (!bSuccess    || (dwBytesRead != sizeof (BITMAPFILEHEADER))
                      || (bmfh.bfType != * (WORD *) "BM"))
     {
                         CloseHandle (hFile) ;
                         return NULL ;
     }

                         // Allocate memory for the BITMAPINFO structure & read
it in
     dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
     pbmi = malloc (dwInfoSize) ;
     bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;
     if (!bSuccess || (dwBytesRead != dwInfoSize))
     {
```

```
                        free (pbmi) ;
                        CloseHandle (hFile) ;
                        return NULL ;
     }
                        // Create the DIB Section
      hBitmap = CreateDIBSection (NULL, pbmi, DIB_RGB_COLORS, &pBits, NULL, 0) ;
      if (hBitmap == NULL)
      {
            free (pbmi) ;
            CloseHandle (hFile) ;
            return NULL ;
       }

                        // Read in the bitmap bits
      ReadFile (hFile, pBits, bmfh.bfSize - bmfh.bfOffBits, &dwBytesRead, NULL) ;
      free (pbmi) ;
      CloseHandle (hFile) ;

      return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
      static HBITMAP              hBitmap ;
      static int                  cxClient, cyClient ;
      static OPENFILENAME ofn ;
      static TCHAR       szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
      static TCHAR       szFilter[] =    TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
       TEXT ("All Files (*.*)\0*.*\0\0") ;
      BITMAP                      bitmap ;
      HDC                         hdc, hdcMem ;
      PAINTSTRUCT           ps ;

      switch (message)
      {
      case  WM_CREATE:
                        ofn.lStructSize             = sizeof (OPENFILENAME) ;
                        ofn.hwndOwner          = hwnd ;
                        ofn.hInstance               = NULL ;
                        ofn.lpstrFilter             = szFilter ;
                        ofn.lpstrCustomFilter  = NULL ;
                        ofn.nMaxCustFilter     = 0 ;
                        ofn.nFilterIndex            = 0 ;
                        ofn.lpstrFile               = szFileName ;
                        ofn.nMaxFile                = MAX_PATH ;
                        ofn.lpstrFileTitle          = szTitleName ;
                        ofn.nMaxFileTitle           = MAX_PATH ;
                        ofn.lpstrInitialDir         = NULL ;
```

```
                    ofn.lpstrTitle              = NULL ;
                    ofn.Flags                   = 0 ;
                    ofn.nFileOffset             = 0 ;
                    ofn.nFileExtension          = 0 ;
                    ofn.lpstrDefExt             = TEXT ("bmp") ;
                    ofn.lCustData               = 0 ;
                    ofn.lpfnHook                = NULL ;
                    ofn.lpTemplateName     = NULL ;

                    return 0 ;

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

     case  WM_COMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_OPEN:

                    // Show the File Open dialog box

                                   if    (!GetOpenFileName (&ofn))
                                         return 0 ;

                    // If there's an existing bitmap, delete it

                                   if    (hBitmap)
                                   {
                                         DeleteObject (hBitmap) ;
                                         hBitmap = NULL ;
                                   }
                    // Create the DIB Section from the DIB file

                                   SetCursor     (LoadCursor     (NULL,
IDC_WAIT)) ;
                                   ShowCursor (TRUE) ;

                              hBitmap      =      CreateDIBsectionFromDibFile
(szFileName) ;
                                   ShowCursor (FALSE) ;
                                   SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

                    // Invalidate the client area for later update

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

```
                                        if (hBitmap == NULL)
                                        {
                MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
                                        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                                        }
                                        return 0 ;
                        }
                        break ;

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

                        if (hBitmap)
                        {
                        GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

                                        hdcMem = CreateCompatibleDC (hdc) ;
                                        SelectObject (hdcMem, hBitmap) ;

                  BitBlt (    hdc,  0, 0, bitmap.bmWidth, bitmap.bmHeight,
                                        hdcMem, 0, 0, SRCCOPY) ;

                                        DeleteDC (hdcMem) ;
                        }

                        EndPaint (hwnd, &ps) ;
                        return 0 ;

      case  WM_DESTROY:
                        if (hBitmap)
                                        DeleteObject (hBitmap) ;

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

DIBSECT.RC（摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Menu
DIBSECT MENU DISCARDABLE
BEGIN
      POPUP "&File"
```

```
      BEGIN
            MENUITEM "&Open",      IDM_FILE_OPEN
      END
END
```

RESOURCE.H（摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DIBsect.rc

#define IDM_FILE_OPEN         40001
```

注意 DIBCONV 中的 CreateBitmapObjectFromDibFile 函式和 DIBSECT 中的 CreateDIbsectionFromDibFile 函式之间的区别。DIBCONV 读入整个档案，然後把指向 DIB 记忆体块的指标传递给 CreateDIBitmap 函式。DIBSECT 首先读取 BITMAPFILEHEADER 结构中的资讯，然後确定 BITMAPINFO 结构的大小，为此配置记忆体，并在第二个 ReadFile 呼叫中将它读入记忆体。然後，函式把指向 BITMAPINFO 结构和指标变数 pBits 的指标传递给 CreateDIBSection。函式传回点阵图代号并设定 pBits 指向函式将要读取 DIB 图素位元的记忆体块。

pBits 指向的记忆体块归系统所有。当通过呼叫 DeleteObject 删除点阵图时，记忆体会被自动释放。然而，程式能利用该指标直接改变 DIB 位元。当应用程式透过 API 传递大量记忆体块时，只要系统拥有这些记忆体块，在 WINDOWS NT 下就不会影响速度。

我之前曾说过，当在视讯显示器上显示 DIB 时，某些时候必须进行从装置无关图素到设备相关图素的转换，有时这些格式转换可能相当费时。来看一看三种用於显示 DIB 的方法：

- 当使用 SetDIBitsToDevice 或 StretchDIBits 来把 DIB 直接显示在萤幕上，格式转换在 SetDIBitsToDevice 或 StretchDIBits 呼叫期间发生。
- 当使用 CreateDIBitmap 和（可能是）SetDIBits 把 DIB 转换为 DDB，然後使用 BitBlt 或 StretchBlt 来显示它时，如果设定了 CBM_INIT 旗标，格式转换在 CreateDIBitmap 或 SetDIBits 期间发生。
- 当使用 CreateDIBSection 建立 DIB 区块，然後使用 BitBlt 或 StretchBlt 显示它时，格式转换在 BitBlt 对 StretchBlt 的呼叫期间发生。

再读一下上面这些叙述，确定您不会误解它的意思。这是从 CreateDIBSection 传回的点阵图代号不同於我们所遇到的其他点阵图代号的一个地方。此点阵图代号实际上指向储存在记忆体中由系统维护但应用程式能存取的 DIB。在需要的时候，DIB 会转化为特定的色彩格式，通常是在用 BitBlt 或 StretchBlt 显示点阵图时。

您也可以将点阵图代号选入记忆体装置内容并使用 GDI 函式来绘制。在 pBits 变数指向的 DIB 图素内将反映出结果。因为 Windows NT 下的 GDI 函式分

批呼叫，在记忆体设备背景上绘制之後和「人为」的存取位元之前会呼叫 GdiFlush。

在 DIBSECT，我们清除 pBits 变数，因为程式不再需要这个变数了。您会使用 CreateDIBSection 的主要原因在於您有需要直接更改位元值。在 CreateDIBSection 呼叫之後似乎就没有别的方法来取得位元指标了。

DIB 区块的其他区别

从 CreateDIBitmap 传回的点阵图代号与函式的 hdc 参数引用的设备有相同的平面和图素位元组织。您能通过具有 BITMAP 结构的 GetObject 呼叫来检验这一点。

CreateDIBSection 就不同了。如果以该函式传回的点阵图代号的 BITMAP 结构呼叫 GetObject，您会发现点阵图具有的色彩组织与 BITMAPINFOHEADER 结构的栏位指出的色彩组织相同。您能将这个代号选入与视讯显示器相容的记忆体装置内容。这与上一章关於 DDB 的内容相矛盾，但这也就是我说此 DIB 区块点阵图代号不同的原因。

另一个奇妙之处是：你可能还记得，DIB 中图素资料行的位元组长度始终是 4 的倍数。GDI 点阵图物件中行的位元组长度，就是使用 GetObject 从 BITMAP 结构的 bmWidthBytes 栏位中得到的长度，始终是 2 的倍数。如果用每图素 24 位元和宽度 2 图素设定 BITMAPINFOHEADER 结构并随後呼叫 GetObject，您就会发现 bmWidthBytes 栏位是 8 而不是 6。

使用从 CreateDIBSection 传回的点阵图代号，也可以使用 DIBSECTION 结构呼叫 GetObject：

```
GetObject (hBitmap, sizeof (DIBSECTION), &dibsection) ;
```

此函式不能处理其他点阵图建立函式传回的点阵图代号。DIBSECTION 结构定义如下：

```
typedef struct tagDIBSECTION  // ds
{
      BITMAP                  dsBm ;            // BITMAP structure
      BITMAPINFOHEADER dsBmih ;                 // DIB information header
      DWORD                   dsBitfields [3] ;// color masks
      HANDLE                  dshSection ;           //   file-mapping   object
handle
      DWORD                   dsOffset ;             // offset to bitmap bits
}
DIBSECTION, * PDIBSECTION ;
```

此结构包含 BITMAP 结构和 BITMAPINFOHEADER 结构。最後两个栏位是传递给 CreateDIBSection 的最後两个参数，等一下将会讨论它们。

DIBSECTION 结构中包含除了色彩对照表以外有关点阵图的许多内容。当把 DIB 区块点阵图代号选入记忆体装置内容时，可以通过呼叫 GetDIBColorTable 来得到色彩对照表：

```
hdcMem = CreateCompatibleDC (NULL) ;
SelectObject (hdcMem, hBitmap) ;
GetDIBColorTable (hdcMem, uFirstIndex, uNumEntries, &rgb) ;
DeleteDC (hdcMem) ;
```

同样，您可以通过呼叫 SetDIBColorTable 来设定色彩对照表中的项目。

档案映射选项

我们还没有讨论 CreateDIBSection 的最後两个参数，它们是档案映射物件的代号和档案中点阵图位元开始的偏移量。档案映射物件使您能够像档案位於记忆体中一样处理档案。也就是说，可以通过使用记忆体指标来存取档案，但档案不需要整个载入记忆体中。

在大型 DIB 的情况下，此技术对於减少记忆体需求是很有帮助的。DIB 图素位元能够储存在磁片上，但仍然可以当作位於记忆体中一样进行存取，虽然会影响程式执行效能。问题是，当图素位元实际上储存在磁片上时，它们不可能是实际 DIB 档案的一部分。它们必须位於其他的档案内。

为了展示这个程序，下面显示的函式除了不把图素位元读入记忆体以外，与 DIBSECT 中建立 DIB 区块的函式很相似。然而，它提供了档案映射物件和传递给 CreateDIBSection 函式的偏移量：

```
HBITMAP CreateDIBsectionMappingFromFile (PTSTR szFileName)
{
      BITMAPFILEHEADER       bmfh ;
      BITMAPINFO                   *     pbmi ;
      BYTE                         *     pBits ;
      BOOL                   bSuccess ;
      DWORD                  dwInfoSize, dwBytesRead ;
      HANDLE                 hFile, hFileMap ;
      HBITMAP                hBitmap ;

      hFile = CreateFile (szFileName, GENERIC_READ | GENERIC_WRITE,
                          0,           // No sharing!
                          NULL, OPEN_EXISTING, 0, NULL) ;

      if (hFile == INVALID_HANDLE_VALUE)
                        return NULL ;
      bSuccess = ReadFile (  hFile, &bmfh, sizeof (BITMAPFILEHEADER),
                          &dwBytesRead, NULL) ;

      if (!bSuccess    || (dwBytesRead != sizeof (BITMAPFILEHEADER))
```

```
                        || (bmfh.bfType != * (WORD *) "BM"))
      {
                        CloseHandle (hFile) ;
                        return NULL ;
      }
      dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
      pbmi = malloc (dwInfoSize) ;
      bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;

      if (!bSuccess || (dwBytesRead != dwInfoSize))
      {
                        free (pbmi) ;
                        CloseHandle (hFile) ;
                        return NULL ;
      }
      hFileMap = CreateFileMapping (hFile, NULL, PAGE_READWRITE, 0, 0, NULL) ;
      hBitmap = CreateDIBSection ( NULL, pbmi, DIB_RGB_COLORS, &pBits,hFileMap,
bmfh.bfOffBits) ;
      free (pbmi) ;
      return hBitmap ;
}
```

啊哈！这个程式不会动。CreateDIBSection 的文件指出「dwOffset [函式的最後一个参数]必须是 DWORD 大小的倍数」。尽管资讯表头的大小始终是 4 的倍数并且色彩对照表的大小也始终是 4 的倍数，但点阵图档案表头却不是，它是 14 位元组。因此 bmfh.bfOffBits 永远不会是 4 的倍数。

总结

如果您有小型的 DIB 并且需要频繁地操作图素位元，您可以使用 SetDIBitsToDevice 和 StretchDIBits 来显示它们。然而，对於大型的 DIB，此技术会遇到显示效能的问题，尤其在 8 位元视讯显示器上和 Windows NT 环境下。

您可以使用 CreateDIBitmap 和 SetDIBits 把 DIB 转化为 DDB。现在，显示点阵图可以使用快速的 BitBlt 和 StretchBlt 函式来进行了。然而，您不能直接存取这些与装置无关的图素位元。

CreateDIBSection 是一个很好的折衷方案。在 Windows NT 下通过 BitBlt 和 StretchBlt 使用点阵图代号比使用 SetDIBitsToDevice 和 StretchDIBits(但没有 DDB 的缺陷）会得到更好的效能。您仍然可以存取 DIB 图素位元。

下一章，在讨论「Windows 调色盘管理器」之後会进入点阵图的探索。