第十八章 Metafile

Metafile 和向量图形的关系，就像点阵图和位元映射图形的关系一样。点阵图通常来自实际的图像，而 metafile 则大多是通过电脑程式人为建立的。Metafile 由一系列与图形函式呼叫相同的二进位记录组成，这些记录一般用於绘制直线、曲线、填入的区域和文字等。

「画图（paint）」程式建立点阵图，而「绘图(draw)」程式建立 metafile。在优秀的绘图程式中，能轻易地「抓住」某个独立的图形物件（例如一条直线）并将它移动到其他位置。这是因为组成图形的每个成员都是以单独的记录储存的。在画图程式中，这是不可能的——您通常都会局限於删除或插入点阵图矩形块。

由於 metafile 以图形绘制命令描述图像，因此可以对图像进行缩放而不会失真。点阵图则不然，如果以二倍大小来显示点阵图，您却无法得到二倍的解析度，而只是在水平和垂直方向上重复点阵图的位元。

Metafile 可以转换为点阵图，但是会丢失一些资讯：组成 metafile 的图形物件将不再是独立的，而是被合并进大的图像。将点阵图转换为 metafile 要艰难得多，一般仅限於非常简单的图像，而且它需要大量处理来分析边界和轮廓。而 metafile 可以包含绘制点阵图的命令。

虽然 metafile 可以作为图片剪辑储存在磁片上，但是它们大多用於程式通过剪贴簿共用图片的情况。由於 metafile 将图片描述为图像函式呼叫的集合，因而它们既比点阵图占用更少的空间，又比点阵图更与装置无关。

Microsoft Windows 支援两种 metafile 格式和支援这些格式的两组函式。我首先讨论从 Windows 1.0 到目前的 32 位元 Windows 版本都支援的 metafile 函式，然後讨论为 32 位元 Windows 系统开发的「增强型 metafile」。增强型 metafile 在原有 metafile 的基础上有了一些改进，应该尽可能地加以利用。

旧的 metafile 格式

Metafile 既能够暂时储存在记忆体中，也能够以档案的形式储存在磁片上。对应用程式来说，两者区别不大，尤其是由 Windows 来处理磁片上储存和载入 metafile 资料的档案 I/O 时，更是如此。

记忆体 metafile 的简单利用

如果呼叫 CreateMetaFile 函式来建立 metafile 装置内容，Windows 就会以

早期的格式建立一个 metafile，然後您可以使用大部分 GDI 绘图函式在该 metafile 装置内容上进行绘图。这些 GDI 呼叫并不在任何具体的装置上绘图，相反地，它们被储存在 metafile 中。当关闭 metafile 装置内容时，会得到 metafile 的代号。这时就可以在某个具体的装置内容上「播放」这个 metafile，这与直接执行 metafile 中 GDI 函式的效果等同。

CreateMetaFile 只有一个参数，它可以是 NULL 或档案名称。如果是 NULL，则 metafile 储存在记忆体中。如果是档案名称(以.WMF 作为「Windows Metafile」的副档名），则 metafile 储存在磁片档案中。

程式 18-1 中的 METAFILE 显示了在 WM_CREATE 讯息处理期间建立记忆体 metafile 的方法，并在 WM_PAINT 讯息处理期间将图像显示 100 遍。

程式 18-1　METAFILE

```
METAFILE.C
/*------------------------------------------------------------------------
      METAFILE.C --          Metafile Demonstration Program
                                                    (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      static TCHAR szAppName [] = TEXT ("Metafile") ;
      HWND                                                hwnd ;
      MSG                                           msg ;
      WNDCLASS                                      wndclass ;

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

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

```
                                                    szAppName,
MB_ICONERROR) ;
                return 0 ;
     }

     hwnd = CreateWindow (  szAppName, TEXT ("Metafile Demonstration"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

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

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

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     static HMETAFILE       hmf ;
     static int                         cxClient, cyClient ;
     HBRUSH                                   hBrush ;
     HDC                                            hdc, hdcMeta ;
     int                                            x, y ;
     PAINTSTRUCT                        ps ;

     switch (message)
     {
     case  WM_CREATE:
                    hdcMeta     =     CreateMetaFile (NULL) ;
                    hBrush      =     CreateSolidBrush (RGB (0, 0, 255)) ;
                    Rectangle   (hdcMeta, 0, 0, 100, 100) ;

                    MoveToEx          (hdcMeta,   0,    0,    NULL) ;
                    LineTo            (hdcMeta,   100,  100)  ;
                    MoveToEx          (hdcMeta,   0,    100,  NULL) ;
                    LineTo            (hdcMeta,   100,  0)    ;

                    SelectObject (hdcMeta, hBrush) ;
                    Ellipse (hdcMeta, 20, 20, 80, 80) ;

                    hmf = CloseMetaFile (hdcMeta) ;
```

```
                    DeleteObject (hBrush) ;
                    return 0 ;

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

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

                    SetMapMode (hdc, MM_ANISOTROPIC) ;
                    SetWindowExtEx (hdc, 1000, 1000, NULL) ;
                    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

                    for (x = 0 ; x < 10 ; x++)
                    for (y = 0 ; y < 10 ; y++)
                    {
                              SetWindowOrgEx (hdc, -100 * x, -100 * y, NULL) ;
                              PlayMetaFile (hdc, hmf) ;
                    }
                    EndPaint (hwnd, &ps) ;
                    return 0 ;

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

这个程式展示了在使用记忆体 metafile 时所涉及的 4 个 metafile 函式的用法。第一个是 CreateMetaFile。在 WM_CREATE 讯息处理期间用 NULL 参数呼叫该函式，并传回 metafile 装置内容的代号。然後，METAFILE 利用这个 metafileDC 来绘制两条直线和一个蓝色椭圆。这些函式呼叫以二进位形式储存在 metafile 中。CloseMetaFile 函式传回 metafile 的代号。因为以後还要用到该 metafile 代号，所以把它储存在静态变数。

该 metafile 包含 GDI 函式呼叫的二进位表示码，它们是两个 MoveToEx 呼叫、两个 LineTo 呼叫、一个 SelectObject 呼叫（指定蓝色画刷）和一个 Ellipse 呼叫。座标没有指定任何映射方式或转换，它们只是作为数值资料被储存在 metafile 中。

在 WM_PAINT 讯息处理期间，METAFILE 设定一种映射方式并呼叫

PlayMetaFile 在视窗中绘制物件 100 次。Metafile 中函式呼叫的座标按照目的装置内容的目前变换方式加以解释。在呼叫 PlayMetaFile 时，事实上是在重复地呼叫最初在 WM_CREATE 讯息处理期间建立 metafile 时，在 CreateMetaFile 和 CloseMetaFile 之间所做的所有呼叫。

和任何 GDI 物件一样，metafile 物件也应该在程式终止前被删除。这是在 WM_DESTROY 讯息处理期间用 DeleteMetaFile 函式处理的工作。

METAFILE 程式的结果如图 18-1 所示。

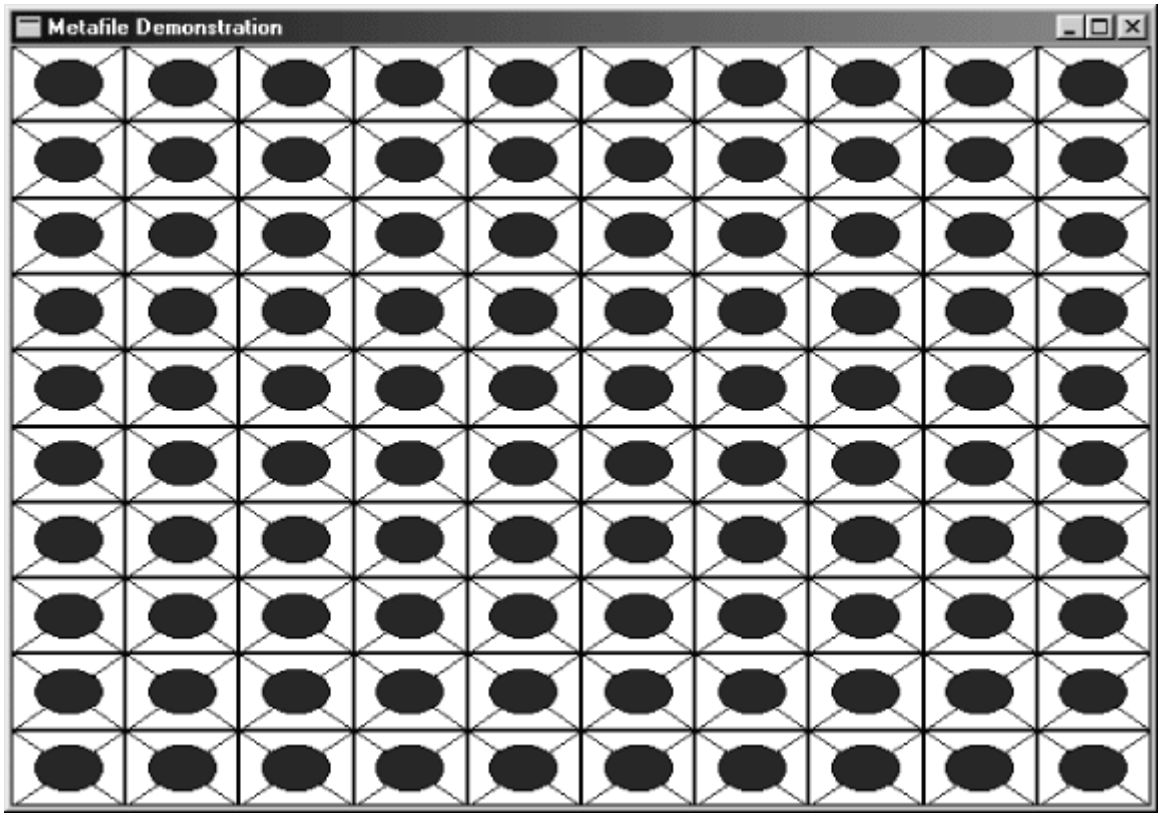

图 18-1 METAFILE 程式执行结果显示

将 metafile 储存在磁碟上

在上面的例子中，CreateMetaFile 的 NULL 参数表示要建立储存在记忆体中的 metafile。我们也可以建立作为档案储存在磁碟上的 metafile，这种方法对於大的 metafile 比较合适，因为可以节省记忆体空间。而另一方面，每次使用磁片上的 metafile 时，就需要存取磁片。

要把 METAFILE 转换为使用 metafile 磁片档案的程式，必须把 CreateMetaFile 的 NULL 参数替换为档案名称。在 WM_CREATE 处理结束时，可以用 metafile 代号来呼叫 DeleteMetaFile，这样代号被删除，但是磁片档案仍然被储存著。

在处理 WM_PAINT 讯息处理期间，可以通过呼叫 GetMetaFile 来取得此磁碟档案的 metafile 代号：

```
hmf = GetMetaFile (szFileName) ;
```

现在就可以像前面那样显示这个 metafile。在 WM_PAINT 讯息处理结束时，可以用下面的叙述删除该 metafile 代号：

```
DeleteMetaFile (hmf) ;
```

在开始处理 WM_DESTROY 讯息时，不必删除 metafile，因为它已经在 WM_CREATE 讯息和每个 WM_PAINT 讯息结束时被删除了，但是仍然需要删除磁碟档案：

```
DeleteFile (szFileName) ;
```

当然，除非您想储存该档案。

正如在第十章讨论过的，metafile 也可以作为使用者自订资源。您可以简单地把它当作资料块载入。如果您有一块包含 metafile 内容的资料，那么您可以使用

```
hmf = SetMetaFileBitsEx (iSize, pData) ;
```

来建立 metafile。SetMetaFileBitsEx 有一个对应的函式——GetMetaFileBitsEx，此函式将 metafile 的内容复制到记忆体块中。

老式 metafile 与剪贴簿

老式 metafile 有个讨厌的缺陷。如果您具有老式 metafile 的代号，那么，当您在显示 metafile 时如何确定它的大小呢？除非您深入分析 metafile 的内部结构，否则无法得知。

此外，当程式从剪贴簿取得老式 metafile 时，如果 metafile 被定义为在 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式下显示,则此程式在使用该 metafile 时具有最大程度的灵活性。程式收到该 metafile 後，就可以在显示它之前简单地通过设定视埠的范围来缩放图像。然而，如果 metafile 内的映射方式被设定为 MM_ISOTROPIC 或 MM_ANISOTROPIC，则收到该 metafile 的程式将无法继续执行。程式仅能在显示 metafile 之前或之後进行 GDI 呼叫,不允许在显示 metafile 当中进行 GDI 呼叫。

为了解决这些问题,老式 metafile 代号不直接放入剪贴簿供其他程式取得，而是作为「metafile 图片」（METAFILEPICT 结构型态）的一部分。此结构使得从剪贴簿上取得 metafile 图片的程式能够在显示 metafile 之前设定映射方式和视埠范围。

METAFILEPICT 结构的长度为 16 个位元组，定义如下：

```
typedef struct tagMETAFILEPICT
{
      LONG mm ;                                  // mapping mode
      LONG xExt ;                                // width of the metafile image
      LONG yExt ;                                // height of the metafile image
```

```
     LONG hMF ;                                  // handle to the metafile
}
METAFILEPICT ;
```

对於 MM_ISOTROPIC 和 MM_ANISOTROPIC 以外的所有映射方式，图像大小用 xExt 和 yExt 值表示，其单位是由 mm 给出的映射方式的单位。利用这些资讯，从剪贴簿复制 metafile 图片结构的程式就能够确定在显示 metafile 时所需的显示空间。建立该 metafile 的程式可以将这些值设定为输入 metafile 的 GDI 绘制函式中所使用的最大的 x 座标和 y 座标值。

在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式下，xExt 和 yExt 栏位有不同的功能。我们在第五章中曾介绍过一个程式，该程式为了在 GDI 函式中使用与图像实际尺寸无关的逻辑单位而采用 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式。当程式只想保持纵横比而可以忽略图形显示平面的大小时，采用 MM_ISOTROPIC 模式；反之，当不需要考虑纵横比时采用 MM_ANISOTROPIC 模式。您也许还记得，第五章中在程式将映射方式设定为 MM_ISOTROPIC 或 MM_ANISOTROPIC 後，通常会呼叫 SetWindowExtEx 和 SetViewportExtEx。SetWindowExtEx 呼叫使用逻辑单位来指定程式在绘制时使用的单位，而 SetViewportExtEx 呼叫使用的装置单位大小则取决於图形显示平面（例如，视窗显示区域的大小）。

如果程式为剪贴簿建立了 MM_ISOTROPIC 或 MM_ANISOTROPIC 方式的 metafile，则该 metafile 本身不应包含对 SetViewportExtEx 的呼叫，因为该呼叫中的装置单位应该依据建立 metafile 的程式的显示平面，而不是依据从剪贴簿读取并显示 metafile 的程式的显示平面。从剪贴簿取得 metafile 的程式可以利用 xExt 和 yExt 值来设定合适的视埠范围以便显示 metafile。但是当映射方式是 MM_ISOTROPIC 或 MM_ANISOTROPIC 时，metafile 本身包含设定视窗范围的呼叫。Metafile 内的 GDI 绘图函式的座标依据这些视窗的范围。

建立 metafile 和 metafile 图片遵循以下规则：

- 设定 METAFILEPICT 结构的 mm 栏位来指定映射方式。
- 对於 MM_ISOTROPIC 和 MM_ANISOTROPIC 以外的映射方式，xExt 与 yExt 栏位设定为图像的宽和高，单位与 mm 栏位相对应。对於在 MM_ISOTROPIC 或 MM_ANISOTROPIC 方式下显示的 metafile，工作要复杂一些。在 MM_ANISOTROPIC 模式下，当程式既不对图片大小跟纵横比给出任何建议资讯时，xExt 和 yExt 的值均为零。在这两种模式下，如果 xExt 和 yExt 的值为正数，它们就是以 0.01mm 单位（MM_HIMETRIC 单位）表示该图像的宽度和高度。在 MM_ISOTROPIC 方式下，如果 xExt 和 yExt 为负值，它们就指出了图像的纵横比而不是大小。

- 在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式下，metafile 本身含有对 SetWindowExtEx 的呼叫，也可能有对 SetWindowOrgEx 的呼叫。亦即，建立 metafile 的程式在 metafile 装置内容中呼叫这些函式。Metafile 一般不会包含对 SetMapMode、SetViewportExtEx 或 SetViewportOrgEx 的呼叫。
- metafile 应该是记忆体 metafile，而不是 metafile 档案。

这里有一段范例程式码，它建立 metafile 并将其复制到剪贴簿。如果 metafile 使用 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式，则该 metafile 的第一个呼叫应该设定视窗范围（在其他模式中，视窗的大小是固定的）。无论在哪种模式下，视窗的位置应如下设定：

```
hdcMeta = CreateMetaFile (NULL) ;
SetWindowExtEx (hdcMeta, ...) ;
SetWindowOrgEx (hdcMeta, ...) ;
```

Metafile 绘图函式中的座标决定於这些视窗范围和视窗原点。当程式使用 GDI 呼叫在 metafile 装置内容中绘制完成後，关闭 metafile 以得到 metafile 代号：

```
hmf = CloseMetaFile (hdcMeta) ;
```

该程式还需要定义指向 METAFILEPICT 型态结构的指标，并为此结构配置一块整体记忆体：

```
GLOBALHANDLE           hGlobal ;
LPMETAFILEPICT         pMFP ;
其他行程式
hGlobal= GlobalAlloc (GHND | GMEM_SHARE, sizeof (METAFILEPICT)) ;
pMFP = (LPMETAFILEPICT) GlobalLock (hGlobal) ;
接著，程式设定该结构的 4 个栏位：
pMFP->mm    = MM_... ;
pMFP->xExt  = ...  ;
pMFP->yExt  = ...  ;
pMFP->hMF   = hmf  ;

GlobalUnlock (hGlobal) ;
```

然後，程式将包含有 metafile 图片的整体记忆体块传送给剪贴簿：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (CF_METAFILEPICT, hGlobal) ;
CloseClipboard () ;
```

完成这些呼叫後，hGlobal 代号（包含 metafile 图片结构的记忆体块）和 hmf 代号（metafile 本身）就对建立它们的程式失效了。

现在来看一看难的部分。当程式从剪贴簿取得 metafile 并显示它时，必须完成下列步骤：

1. 程式利用 metafile 图片结构的 mm 栏位设定映射方式。

2. 对於 MM_ISOTROPIC 或 MM_ANISOTROPIC 以外的映射方式，程式用 xExt 和 yExt 值设定剪贴矩形或简单地设定图像大小。而在 MM_ISOTROPIC 和 MM_ANISOTROPIC 映射方式，程式使用 xExt 和 yExt 来设定视埠范围。

3. 然後，程式显示 metafile。

下面程式码，首先打开剪贴簿，得到 metafile 图片结构代号并将其锁定：

```
OpenClipboard (hwnd) ;
hGlobal = GetClipboardData (CF_METAFILEPICT) ;
pMFP = (LPMETAFILEPICT) GlobalLock (hGlobal) ;
```

现在可以储存目前装置内容的属性，并将映射方式设定为结构中的 mm 值：

```
SaveDC (hdc) ;
SetMappingMode (pMFP->mm) ;
```

如果映射方式不是 MM_ISOTROPIC 或 MM_ANISOTROPIC，则可以用 xExt 和 yExt 的值设定剪贴矩形。由於这两个值是逻辑单位，必须用 LPtoDP 将其转换为用於剪贴矩形的装置单位的座标。也可以简单地储存这些值以掌握图像的大小。

对於 MM_ISOTROPIC 或 MM_ANISOTROPIC 映射方式，xExt 和 yExt 用来设定视埠范围。下面有一个用来完成此项任务的函式，如果 xExt 和 yExt 没有建议的大小，则该函式假定 cxClient 和 cyClient 分别表示 metafile 显示区域的图素高度和宽度。

```
void PrepareMetaFile ( HDC hdc, LPMETAFILEPICT pmfp,
                              int cxClient, int cyClient)
{
      int xScale, yScale, iScale ;
      SetMapMode (hdc, pmfp->mm) ;
      if (pmfp->mm == MM_ISOTROPIC || pmfp->mm == MM_ANISOTROPIC)
      {
                  if (pmfp->xExt == 0)
                              SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
                  else if (pmfp->xExt > 0)
                                    SetViewportExtEx (hdc,
                              pmfp->xExt *     GetDeviceCaps (hdc, HORZRES) /
                            GetDeviceCaps (hdc, HORZSIZE) / 100),
                              pmfp->yExt *     GetDeviceCaps (hdc, VERTRES) /
                            GetDeviceCaps (hdc, VERTSIZE) / 100), NULL) ;
                                    else if (pmfp->xExt < 0)
                              {
                  xScale = 100 *   cxClient * GetDeviceCaps (hdc, HORZSIZE) /
                                 GetDeviceCaps (hdc, HORZRES) / -pmfp->xExt ;
                  lScale = 100 *   cyClient * GetDeviceCaps (hdc, VERTSIZE) /
                     GetDeviceCaps (hdc, VERTRES) / -pmfp->yExt ;
                              iScale = min (xScale, yScale) ;
```

```
   SetViewportExtEx (hdc, -pmfp->xExt * iScale * GetDeviceCaps (hdc, HORZRES)
/
                    GetDeviceCaps (hdc, HORZSIZE) / 100, -pmfp->yExt * iScale
* GetDeviceCaps (hdc, VERTRES) / GetDeviceCaps (hdc, VERTSIZE) / 100, NULL) ;
                    }
               }
}
```

上面的程式码假设 xExt 和 yExt 同时都为零、大於零或小於零，这三种状态之一。如果范围为零，表示没有建议大小或纵横比，视埠范围设定为显示 metafile 的区域。如果大於零，则 xExt 和 yExt 的值代表图像的建议大小，单位是 0.01mm。GetDeviceCaps 函式用来确定每 0.01mm 中包含的图素数，并且该值与 metafile 图片结构的范围值相乘。如果小於零，则 xExt 和 yExt 的值表示建议的纵横比而不是建议的大小。iScale 的值首先根据对应 cxClient 和 cyClient 的毫米表示的纵横比计算出来，该缩放因数用於设定图素单位的视埠范围。

完成了上述工作後，可以设定视埠原点，显示 metafile，并恢复装置内容：

```
PlayMetaFile (pMFP->hMF) ;
RestoreDC (hdc, -1) ;
```

然後，对记忆体块解锁并关闭剪贴簿：

```
GlobalUnlock (hGlobal) ;
CloseClipboard () ;
```

如果程式使用增强型 metafile 就可以省去这项工作。当某个应用程式将这些格式放入剪贴簿而另一个程式却要求从剪贴簿中获得其他格式时，Windows 剪贴簿会自动在老式 metafile 和增强型 metafile 之间进行格式转换。

增强型 metafile

「增强型 metafile」 格式是在 32 位元 Windows 版本中发表的。它包含一组新的函式呼叫、一对新的资料结构、新的剪贴簿格式和新的档案副档名.EMF。

这种新的 metafile 格式最重要的改进是加入可通过函式呼叫取得的更丰富的表头资讯，这种表头资讯可用来帮助应用程式显示 metafile 图像。

有些增强型 metafile 函式使您能够在增强型 metafile(EMF)格式和老式 metafile 格式（也称作 Windows metafile(WMF)格式）之间来回转换。当然，这种转换很可能遇到麻烦，因为老式 metafile 格式并不支援某些，例如 GDI 绘图路径等，新的 32 位元图形功能。

基本程序

程式 18-2 所示的 EMF1 建立并显示增强型 metafile。

程式 18-2 EMF1

```
EMF1.C
/*------------------------------------------------------------------------
--
      EMF1.C --  Enhanced Metafile Demo #1
                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    LPSTR  lpszCmdLine,  int
nCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("EMF1") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;

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

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

      hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #1"),
                             WS_OVERLAPPEDWINDOW,
                             CW_USEDEFAULT, CW_USEDEFAULT,
                             CW_USEDEFAULT, CW_USEDEFAULT,
                             NULL, NULL, hInstance, NULL) ;

      ShowWindow (hwnd, nCmdShow) ;
      UpdateWindow (hwnd) ;

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

```
     {
                    TranslateMessage (&msg) ;
                    DispatchMessage (&msg) ;
    }
     return msg.wParam ;
}

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

     switch (message)
    {
     case  WM_CREATE:
                    hdcEMF = CreateEnhMetaFile (NULL, NULL, NULL, NULL) ;

                    Rectangle   (hdcEMF, 100, 100, 200, 200) ;

                    MoveToEx          (hdcEMF, 100, 100, NULL) ;
                    LineTo            (hdcEMF, 200, 200) ;

                    MoveToEx          (hdcEMF, 200, 100, NULL) ;
                    LineTo            (hdcEMF, 100, 200) ;

                    hemf = CloseEnhMetaFile (hdcEMF) ;
                    return 0 ;

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

                GetClientRect (hwnd, &rect) ;

                rect.left        =                rect.right       / 4 ;
                rect.right       = 3 *      rect.right       / 4 ;
                rect.top         =                rect.bottom      / 4 ;
                rect.bottom      = 3 *      rect.bottom      / 4 ;

                     PlayEnhMetaFile (hdc, hemf, &rect) ;

                     EndPaint (hwnd, &ps) ;
                     return 0 ;

     case  WM_DESTROY:
                    DeleteEnhMetaFile (hemf) ;
```

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

在 EMF1 的视窗讯息处理程式处理 WM_CREATE 讯息处理期间，程式首先通过呼叫 CreateEnhMetaFile 来建立增强型 metafile。该函式有 4 个参数，但可以把它们都设为 NULL。稍候我将说明这 4 个参数在非 NULL 情况下的使用方法。

和 CreateMetaFile 一样，CreateEnhMetaFile 函式传回特定的装置内容代号。该程式利用这个代号绘制一个矩形和该矩形的两条对角线。这些函式呼叫及其参数被转换为二进位元的形式并储存在 metafile 中。

最後通过对 CloseEnhMetaFile 函式的呼叫结束了增强型 metafile 的建立并传回指向它的代号。该档案代号储存在 HENHMETAFILE 型态的静态变数中。

在 WM_PAINT 讯息处理期间，EMF1 以 RECT 结构取得程式的显示区域视窗大小。通过调整结构中的 4 个栏位，使该矩形的长和宽为显示区域视窗长和宽的一半并位於视窗的中央。然後 EMF1 呼叫 PlayEnhMetaFile，该函式的第一个参数是视窗的装置内容代号，第二个参数是该增强型 metafile 的代号，第三个参数是指向 RECT 结构的指标。

在 metafile 的建立程序中,GDI 得出整个 metafile 图像的尺寸。在本例中，图像的长和宽均为 100 个单位。在 metafile 的显示程序中，GDI 将图像拉伸以适应 PlayEnhMetaFile 函式指定的矩形大小。EMF1 在 Windows 下执行的三个执行实体如图 18-2 所示。

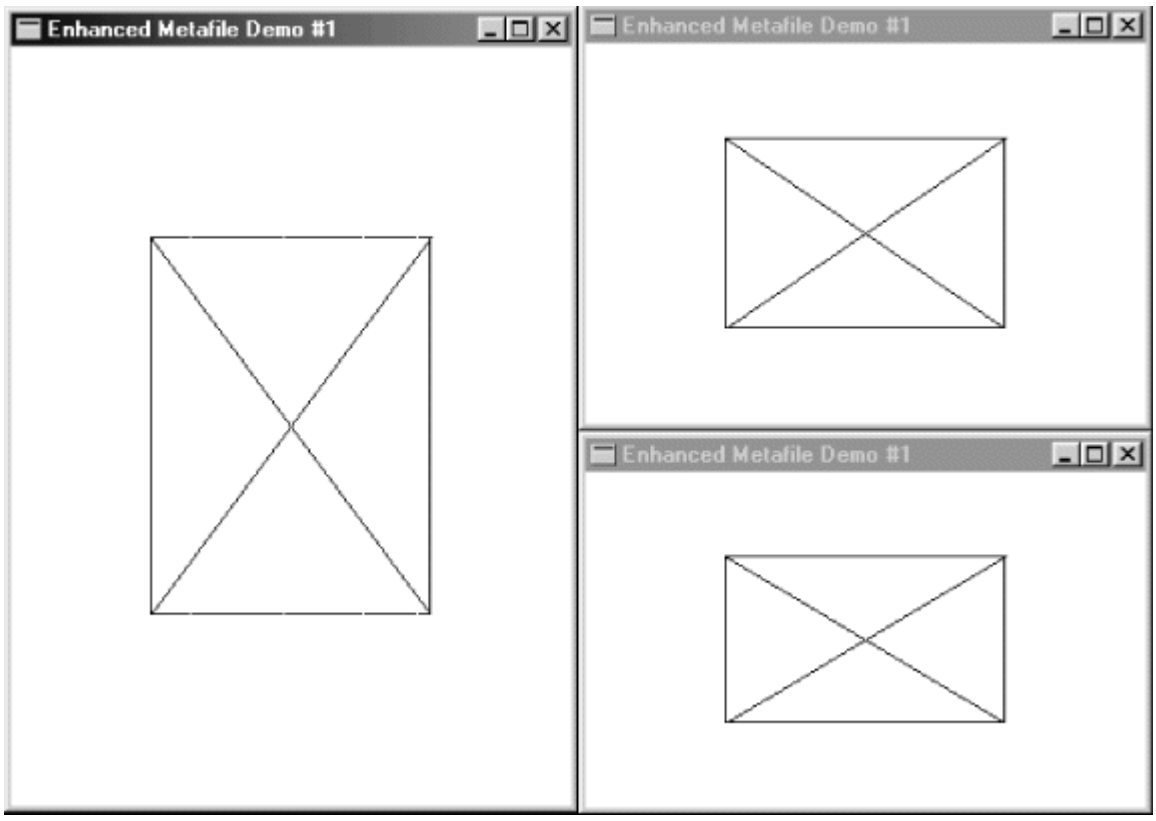

图 18-2 EMF1 得萤幕显示

最後，在 WM_DESTROY 讯息处理期间，EMF1 呼叫 DeleteEnhMetaFile 删除 metafile。

让我们总结一下从 EMF1 程式学到的一些东西。

首先，该程式在建立增强型 metafile 时，画矩形和直线的函式所使用的座标并不是实际意义上的座标。您可以将它们同时加倍或都减去某个常数，而其结果不会改变。这些座标只是在定义图像时说明彼此间的对应关系。

其次，为了适於在传递给 PlayEnhMetaFile 函式的矩形中显示，图像大小会被缩放。因此，如图 18-2 所示，图像可能会变形。尽管 metafile 座标指出该图像是正方形的，但一般情况下我们却得不到这样的图像。而在某些时候，这又正是我们想要得到的图像。例如，将图像嵌入一段文书处理格式的文字中时，可能会要求使用者为图像指定矩形，并且确保整个图像恰好位於矩形中而不浪费空间。这样，使用者可通过适当调整矩形的大小来得到正确的纵横比。

然而有时候，您也许希望保留图像最初的纵横比，因为这一点对於表现视觉资讯尤为重要。例如，警察的嫌疑犯草图既不能比原型胖也不能比原型瘦。或者您希望保留原来图像的度量尺寸，图像必须是两英寸高，否则就不能正常显示。在这种情况下，保留图像的原来尺寸就非常重要了。

同时也要注意 metafile 中画出的那些对角线似乎没有与矩形顶点相交。这是由於 Windows 在 metafile 中储存矩形座标的方式造成的。稍後，会说明解决这个问题的方法。

揭开内幕

如果看一看 metafile 的内容会对 metafile 工作的方式有一个更好的理解。如果您有一个 metafile 档案，这将很容易做到，程式 18-3 中的 EMF2 程式建立了一个 metafile。

程式 18-3 EMF2

```
EMF2.C
/*-------------------------------------------------------------------------
---
      EMF2.C --   Enhanced Metafile Demo #2
                                  (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                LPSTR   lpszCmdLine,   int
nCmdShow)
{
      static TCHAR           szAppName[] = TEXT ("EMF2") ;
      HWND                                    hwnd ;
      MSG                                msg ;
      WNDCLASS                           wndclass ;
      wndclass.style                                 = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc               = WndProc ;
      wndclass.cbClsExtra                = 0 ;
      wndclass.cbWndExtra                = 0 ;
      wndclass.hInstance                 = hInstance ;
      wndclass.hIcon                     = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor             = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground       = GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName        = NULL ;
      wndclass.lpszClassName       = szAppName ;

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

      hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #2"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
```

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

      ShowWindow (hwnd, nCmdShow) ;
      UpdateWindow (hwnd) ;

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

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

      switch (message)
      {
      case  WM_CREATE:
            hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf2.emf"), NULL,
          TEXT ("EMF2\0EMF Demo #2\0")) ;

                        if (!hdcEMF)
                              return 0 ;

                        Rectangle   (hdcEMF, 100, 100, 200, 200) ;

                        MoveToEx          (hdcEMF, 100, 100, NULL) ;
                        LineTo            (hdcEMF, 200, 200) ;

                        MoveToEx          (hdcEMF, 200, 100, NULL) ;
                        LineTo            (hdcEMF, 100, 200) ;

                        hemf = CloseEnhMetaFile (hdcEMF) ;

                        DeleteEnhMetaFile (hemf) ;
                        return 0 ;

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

                        GetClientRect (hwnd, &rect) ;
```

```
                    rect.left        =        rect.right     / 4 ;
                    rect.right       = 3 *       rect.right       / 4 ;
                    rect.top    =      rect.bottom           / 4 ;
                    rect.bottom      = 3 * rect.bottom      / 4 ;

                    if (hemf = GetEnhMetaFile (TEXT ("emf2.emf")))
                    {
                              PlayEnhMetaFile (hdc, hemf, &rect) ;
                              DeleteEnhMetaFile (hemf) ;
                    }
                    EndPaint (hwnd, &ps) ;
                    return 0 ;

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

在 EMF1 程式中，CreateEnhMetaFile 函式的所有参数均被设定为 NULL。在 EMF2 中，第一个参数仍旧设定为 NULL，该参数还可以是装置内容代号。GDI 使用该参数在 metafile 表头中插入度量资讯，很快我会讨论它。如果该参数为 NULL，则 GDI 认为度量资讯是由视讯装置内容决定的。

CreateEnhMetaFile 函式的第二个参数是档案名称。如果该参数为 NULL（在 EMF1 中为 NULL，但在 EMF2 中不为 NULL），则该函式建立记忆体 metafile。EMF2 建立名为 EMF2.EMF 的 metafile 档案。

函式的第三个参数是 RECT 结构的位址，它指出了以 0.01mm 为单位的 metafile 的总大小。这是 metafile 表头资料中极其重要的资讯（这是早期的 Windows metafile 格式的缺陷之一）。如果该参数为 NULL，GDI 会计算出尺寸。我比较喜欢让作业系统替我做这些事，所以将该参数设定为 NULL。当应用程式对性能要求比较严格时，就需要使用该参数以避免让 GDI 处理太多东西。

最後的参数是描述该 metafile 的字串。该字串分为两部分：第一部分是以 NULL 字元结尾的应用程式名称（不一定是程式的档案名称），第二部分是描述视觉图像内容的说明，以两个 NULL 字元结尾。例如用 C 中的符号「\0」作为 NULL 字元，则该描述字串可以是「LoonyCad V6.4\0Flying Frogs\0\0」。由於在 C 中通常会在使用的字串末尾放入一个 NULL 字元，所以如 EMF2 所示，在末尾仅需一个「\0」。

建立完 metafile 後，与 EMF1 一样，EMF2 也透过利用由 CreateEnhMetaFile 函式传回的装置内容代号进行一些 GDI 函式呼叫。然後程式呼叫 CloseEnhMetaFile 删除装置内容代号并取得完成的 metafile 的代号。

然後，在 WM_CREATE 讯息还没处理完毕时，EMF2 做了一些 EMF1 没有做的事情：在获得 metafile 代号之後，程式呼叫 DeleteEnhMetaFile。该操作释放了用於储存 metafile 的所有记忆体资源。然而，metafile 档案仍然保留在磁碟机中（如果愿意，您可以使用如 DeleteFile 的档案删除函式来删除该档案）。注意 metafile 代号并不像 EMF1 中那样储存在静态变数中，这意味著在讯息之间不需要储存它。

现在，为了使用该 metafile，EMF2 需要存取磁片档案。这是在 WM_PAINT 讯息处理期间透过呼叫 GetEnhMetaFile 进行的。Metafile 的档案名称是该函式的唯一参数，该函式传回 metafile 代号。和 EMF1 一样，EMF2 将这个档案代号传递给 PlayEnhMetaFile 函式。该 metafile 图像在 PlayEnhMetaFile 函式的最後一个参数所指定的矩形中显示。与 EMF1 不同的是，EMF2 在 WM_PAINT 讯息结束之前就删除该 metafile。此後每次处理 WM_PAINT 讯息时，EMF2 都会再次读取 metafile，显示并删除它。

要记住，对 metafile 的删除操作仅是释放了用以储存 metafile 的记忆体资源而已，磁片 metafile 甚至在程式执行结束後还保留在磁片上。

由於 EMF2 留下了 metafile 档案，您可以看一看它的内容。图 18-3 显示了该程式建立的 EMF2.EMF 档案的一堆十六进位代码。

```
0000  01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00        ........d...d...
0010  C8 00 00 00 C8 00 00 00 35 0C 00 00 35 0C 00 00     ........5...5...
0020  6A 18 00 00 6A 18 00 00 20 45 4D 46 00 00 01 00     j...j...EMF....
0030  F4 00 00 00 07 00 00 00 01 00 00 00 12 00 00 00     ................
0040  64 00 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00     d...............
0050  40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00     @...............
0060  00 00 00 00 45 00 4D 00 46 00 32 00 00 00 45 00     ....E.M.F.2...E.
0070  4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00     M.F..D.e.m.o..
0080  23 00 32 00 00 00 00 00 2B 00 00 00 18 00 00 00     #.2.....+.......
0090  63 00 00 00 63 00 00 00 C6 00 00 00 C6 00 00 00     c...c...........
00A0  1B 00 00 00 10 00 00 00 64 00 00 00 64 00 00 00     ........d...d...
00B0  36 00 00 00 10 00 00 00 C8 00 00 00 C8 00 00 00     6...............
00C0  1B 00 00 00 10 00 00 00 C8 00 00 00 64 00 00 00     ............d...
00D0  36 00 00 00 10 00 00 00 64 00 00 00 C8 00 00 00     6.......d.......
00E0  0E 00 00 00 14 00 00 00 00 00 00 00 10 00 00 00     ................
00F0  14 00 00 00                                             ....
```

图 18-3 EMF2.EMF 的十六进位代码

图 18-3 所示的 metafile 是 EMF2 在 Microsoft Windows NT 4 下，视讯显示器的解析度为 1024 768 时建立的。同一程式在 Windows 98 下建立的 metafile 会比前者少 12 个位元组，这一点将在稍後讨论。同样地，视讯显示器的解析度也影响 metafile 表头的某些资讯。

增强型 metafile 格式使我们对 metafile 的工作方式有更深刻的理解。增

强型 metafile 由可变长度的记录组成，这些记录的一般格式由 ENHMETARECORD 结构说明，它在 WINGDI.H 表头档案中定义如下：

```
typedef struct tagENHMETARECORD
{
      DWORD iType ;                              // record type
      DWORD nSize ;                              // record size
      DWORD dParm [1] ;                    // parameters
}
ENHMETARECORD ;
```

当然，那个只有一个元素的阵列指出了阵列元素的变数。参数的数量取决於记录型态。iType 栏位可以是定义在 WINGDI.H 档案中以字首 EMR_开始的近百个常数之一。nSize 栏位是总记录的大小，包括 iType 和 nSize 栏位以及一个或多个 dParm 栏位。

有了这些知识後，让我们看一下图 18-3。第一个栏位型态为 0x00000001，大小为 0x00000088，所以它占据档案的前 136 个位元组。记录型态为 1 表示常数 EMR_HEADER。我们不妨把对表头纪录的讨论往後搁，先跳到位於第一个记录末尾的偏移量 0x0088 处。

後面的 5 个记录与 EMF2 建立 metafile 之後的 5 个 GDI 函式呼叫有关。该记录在偏移量 0x0088 处有一个值为 0x0000002B 的型态代码，这代表 EMR_RECTANGLE，很明显是用於 Rectangle 呼叫的 metafile 记录。它的长度为 0x00000018（十进位 24）位元组，用以容纳 4 个 32 位元参数。实际上 Rectangle 函式有 5 个参数，但是第一个参数，也就是装置内容代号并未储存在 metafile 中，因为它没有实际意义。尽管在 EMF2 的函式呼叫中指定了矩形的顶点座标分别是(100，100)和(200，200)，但 4 个参数中的 2 个是 0x00000063 (99)，另外 2 个是 0x000000C6 (198)。EMF2 程式在 Windows 98 下建立的 metafile 显示出前两个参数是 0x00000064 (100)，後 2 个参数是 0x000000C7(199)。显然，在 Rectangle 参数储存到 metafile 之前，Windows 对它们作了调整，但没有保持一致。这就是对角线端点与矩形顶点不能重合的原因。

其次，有 4 个 16 位元记录与 2 个 MoveToEx (0x0000001B 或 EMR_MOVETOEX) 和 LineTo (0x00000036 或 EMR_LINETO)呼叫有关。位於 metafile 中的参数与传递给函式的参数相同。

Metafile 以 20 个位元组长的型态代码为 0x0000000E 或 EMR_EOF（「end of file」）的记录结尾。

增强型 metafile 总是以表头纪录开始。它对应於 ENHMETAHEADER 型态的结构，定义如下：

```
typedef struct tagENHMETAHEADER
{
```

```
      DWORD iType ;       // EMR_HEADER = 1
      DWORD nSize ;       // structure size
      RECTL rclBounds ;    // bounding rectangle in pixels
      RECTL rclFrame ;     // size of image in 0.01 millimeters
      DWORD dSignature ;  // ENHMETA_SIGNATURE = " EMF"
      DWORD nVersion ;    // 0x00010000
      DWORD nBytes ;     // file size in bytes
      DWORD nRecords ;   // total number of records
      WORD  nHandles ;   // number of handles in handle table
      WORD  sReserved ;
      DWORD nDescription ;    // character length of description string
      DWORD offDescription ;   // offset of description string in file
      DWORD nPalEntries ;     // number of entries in palette
      SIZEL szlDevice ;       // device resolution in pixels
      SIZEL szlMillimeters ;    // device resolution in millimeters
      DWORD cbPixelFormat ;  // size of pixel format
      DWORD offPixelFormat ;  // offset of pixel format
      DWORD bOpenGL ;     // FALSE if no OpenGL records
}
ENHMETAHEADER ;
```

这种表头纪录的存在可能是增强型 metafile 格式对早期 Windows metafile 所做的最为重要的改进。不需要对 metafile 档案使用档案 I/O 函式来取得这些表头资讯。如果具有 metafile 代号，就可以使用 GetEnhMetaFileHeader 函式：

```
GetEnhMetaFileHeader (hemf, cbSize, &emh) ;
```

第一个参数是 metafile 代号。最後一个参数是指向 ENHMETAHEADER 结构的指标。第二个参数是该结构的大小。可以使用类似的 GetEnh-MetaFileDescription 函式取得描述字串。

如上面所定义的，ENHMETAHEADER 结构有 100 位元组长，但在 MF2.EMFmetafile 中，记录的大小包括描述字串，所以大小为 0x88，即 136 位元组。而 Windows 98metafile 的表头纪录不包含 ENHMETAHEADER 结构的最後 3 个栏位，这一点解释了 12 个位元组的差别。

rclBounds 栏位是指出图像大小的 RECT 结构，单位是图素。将其从十六进位转换过来，我们看到该图像正如我们希望的那样，其左上角位於(100,100)，右下角位於(200,200)。

rclFrame 栏位是提供相同资讯的另一个矩形结构，但它是以 0.01 毫米为单位。在这种情况下，该档案显示两对角顶点分别位於(0x0C35,0x0C35)和(0x186A,0x186A)，用十进位表示为(3125,3125)和(6250,6250)的矩形。这些数字是怎么来的？我们很快就会明白。

dSignature 栏位始终为值 ENHMETA_SIGNATURE 或 0x464D4520。这看上去是一个奇怪的数字，但如果将位元组的排列顺序倒过来（就像 Intel 处理器在记

忆体中储存多位元组数那样）并转换成 ASCII 码，就变成字串″EMF″。dVersion 栏位的值始终是 0x00010000。

其後是 nBytes 栏位，该栏位在本例中是 0x000000F4，这是该 metafile 的总位元组数。nRecords 栏位（在本例中是 0x00000007）指出了记录数——包括表头纪录、5 个 GDI 函式呼叫和档案结束记录。

下面是两个十六位元的栏位。nHandles 栏位为 0x0001。该栏位一般指出 metafile 所使用的图形物件（如画笔、画刷和字体）的非内定代号的数量。由於没有使用这些图形物件，您可能会认为该栏位为零，但实际上 GDI 自己保留了第一个栏位。我们将很快见到代号储存在 metafile 中的方式。

下两个栏位指出描述字串的字元个数，以及描述字串在档案中的偏移量，这里它们分别为 0x00000012（十进位数字 18）和 0x00000064。如果 metafile 没有描述字串，则这两个栏位均为零。

nPalEntries 栏位指出在 metafile 的调色盘表中条目的个数，本例中没有这种情况。

接著表头纪录包括两个 SIZEL 结构，它们包含两个 32 位栏位，cx 和 cy。szlDevice 栏位 （在 metafile 中的偏移量为 0x0040）指出了以图素为单位的输出设备大小，szlMillimeters 栏位（偏移量为 0x0050）指出了以毫米为单位的输出设备大小。在增强型 metafile 文件中，这个输出设备被称作「参考设备（reference device）」。它是依据作为第一个参数传递给 CreateEnhMetaFile 呼叫的代号所指出的装置内容。如果该参数设为 NULL，则 GDI 使用视讯显示器。当 EMF2 建立上面所示的 metafile 时，正巧是在 Windows NT 上以 1024 768 显示模式工作，因此这就是 GDI 使用的参考设备。

GDI 通过呼叫 GetDeviceCaps 取得此资讯。EMF2.EMF 中的 szlDevice 栏位是 0x0400 0x0300（即 1024 768），它是以 HORZRES 和 VERTRES 作为参数呼叫 GetDeviceCaps 得到的。szlMillimeters 栏位是 0x140 0xF0，或 320 240，是以 HORZSIZE 和 VERTSIZE 作为参数呼叫 GetDeviceCaps 得到的。

通过简单的除法就可以得出图素为 0.3125mm 高和 0.3125mm 宽，这就是前面描述的 GDI 计算 rclFrame 矩形尺寸的方法。

在 metafile 中，ENHMETAHEADER 结构後跟一个描述字串，该字串是 CreateEnhMetaFile 函式的最後一个参数。在本例中，该字串由後跟一个 NULL 字元的「EMF2」字串和後跟两个 NULL 字元的「EMF Demo #2」字串组成。总共 18 个字元，要是以 Unicode 方式储存则为 36 个字元。无论建立 metafile 的程式执行在 Windows NT 还是 Windows 98 下，该字串始终以 Unicode 方式储存。

metafile 与 GDI 物件

我们已经知道了 GDI 绘图命令储存在 metafile 中方式，现在看一下 GDI 物件的储存方式。程式 18-4 EMF3 除了建立用於绘制矩形和直线的非内定画笔和画刷以外，与前面介绍的 EMF2 程式很相似。该程式也对 Rectangle 座标的问题提出了一点修改。EMF3 程式使用 GetVersion 来确定执行环境是 Windows 98 还是 Windows NT，并适当地调整参数。

程式 18-4 EMF3

```
EMF3.C
/*------------------------------------------------------------------------
--
      EMF3.C --   Enhanced Metafile Demo #3
                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      static TCHAR szAppName[] = TEXT ("EMF3") ;
      HWND                                                hwnd ;
      MSG                                               msg ;
      WNDCLASS                                          wndclass ;

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

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

```
     hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #3"),
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, NULL) ;

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

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

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     LOGBRUSH                              lb ;
     HDC                                   hdc, hdcEMF ;
     HENHMETAFILE           hemf ;
     PAINTSTRUCT                    ps ;
     RECT                                        rect ;
     switch (message)
    {
     case  WM_CREATE:
           hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf3.emf"), NULL,
          TEXT ("EMF3\0EMF Demo #3\0")) ;

                SelectObject (hdcEMF, CreateSolidBrush (RGB (0, 0, 255))) ;

                      lb.lbStyle = BS_SOLID ;
                      lb.lbColor = RGB (255, 0, 0) ;
                      lb.lbHatch = 0 ;

                      SelectObject (hdcEMF,
                      ExtCreatePen (PS_SOLID | PS_GEOMETRIC, 5, &lb, 0, NULL)) ;

                      if (GetVersion () & 0x80000000) // Windows 98
                      Rectangle (hdcEMF, 100, 100, 201, 201) ;
                      else
                        // Windows NT
                                    Rectangle (hdcEMF, 101, 101, 202, 202) ;

                      MoveToEx          (hdcEMF, 100, 100, NULL) ;
```

```
                        LineTo           (hdcEMF, 200, 200) ;

                        MoveToEx         (hdcEMF, 200, 100, NULL) ;
                        LineTo           (hdcEMF, 100, 200) ;

                        DeleteObject  (SelectObject  (hdcEMF,  GetStockObject
(BLACK_PEN))) ;
                        DeleteObject  (SelectObject  (hdcEMF,  GetStockObject
(WHITE_BRUSH))) ;

                        hemf = CloseEnhMetaFile (hdcEMF) ;

                                  DeleteEnhMetaFile (hemf) ;
                                  return 0 ;

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

                                  GetClientRect (hwnd, &rect) ;

                  rect.left   =     rect.right            / 4 ;
                  rect.right  = 3 *      rect.right            / 4 ;
                  rect.top    =     rect.bottom           / 4 ;
                  rect.bottom = 3 *      rect.bottom           / 4 ;

                                  hemf = GetEnhMetaFile (TEXT ("emf3.emf")) ;

                                  PlayEnhMetaFile (hdc, hemf, &rect) ;
                                  DeleteEnhMetaFile (hemf) ;
                                  EndPaint (hwnd, &ps) ;
                                  return 0 ;

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

如我们所看到的，当利用 CreateEnhMetaFile 传回的装置内容代号来呼叫 GDI 函式时，这些函式呼叫被储存在 metafile 中而不是直接输出到萤幕或印表机上。然而，一些 GDI 函式根本不涉及特定的装置内容。其中有关建立画笔和画刷等图形物件的 GDI 函式十分重要。虽然逻辑画笔和画刷的定义储存在由 GDI 保留的记忆体中，但是在建立这些物件时，这些抽象的定义并未与任何特定的装置内容相关。

EMF3 呼叫 CreateSolidBrush 和 ExtCreatePen 函式。因为这些函式不需要

装置内容代号，`所以 GDI 不会把这些呼叫储存在 metafile 里。当呼叫它们时，GDI 函式只是简单地建立图形绘制物件而不会影响 metafile。

然而，当程式呼叫 SelectObject 函式将 GDI 物件选入 metafile 装置内容时，GDI 既为物件建立函式编码（源自用於储存物件的内部 GDI 资料）也为 metafile 中的 SelectObject 呼叫进行编码。为了解其工作方式，我们来看一下 EMF3.EMF 档案的十六进位代码，如图 18-4 所示：

```
0000  01 00 00 00 88 00 00 00 60 00 00 00 60 00 00 00        ........`...`...
0010  CC 00 00 00 CC 00 00 00 B8 0B 00 00 B8 0B 00 00    ................
0020  E7 18 00 00 E7 18 00 00 20 45 4D 46 00 00 01 00    ........EMF.....
0030  88 01 00 00 0F 00 00 00 03 00 00 00 12 00 00 00    ................
0040  64 00 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00    d...............
0050  40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00    @...............
0060  00 00 00 00 45 00 4D 00 46 00 33 00 00 00 45 00    ....E.M.F.3...E.
0070  4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00    M.F....D.e.m.o..
0080  23 00 33 00 00 00 00 00 27 00 00 00 18 00 00 00    #.3.....'.......
0090  01 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00    ................
00A0  25 00 00 00 0C 00 00 00 01 00 00 00 5F 00 00 00    %..........._...
00B0  34 00 00 00 02 00 00 00 34 00 00 00 00 00 00 00    4.......4.......
00C0  34 00 00 00 00 00 00 00 00 00 01 00 05 00 00 00    4...............
00D0  00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00    ................
00E0  25 00 00 00 0C 00 00 00 02 00 00 00 2B 00 00 00    %...........+...
00F0  18 00 00 00 63 00 00 00 63 00 00 00 C6 00 00 00    ....c...c.......
0100  C6 00 00 00 1B 00 00 00 10 00 00 00 64 00 00 00    ............d...
0110  64 00 00 00 36 00 00 00 10 00 00 00 C8 00 00 00    d...6...........
0120  C8 00 00 00 1B 00 00 00 10 00 00 00 C8 00 00 00    ................
0130  64 00 00 00 36 00 00 00 10 00 00 00 64 00 00 00    d...6.......d...
0140  C8 00 00 00 25 00 00 00 0C 00 00 00 07 00 00 80    ....%...........
0150  28 00 00 00 0C 00 00 00 02 00 00 00 25 00 00 00    (...........%...
0160  0C 00 00 00 00 00 00 80 28 00 00 00 0C 00 00 00    ........(.......
0170  01 00 00 00 0E 00 00 00 14 00 00 00 00 00 00 00    ................
0180  10 00 00 00 14 00 00 00
        ........
```

图 18-4 EMF3.EMF 的十六进位代码

如果把这个 metafile 跟前面的 EMF2.EMF 档案进行比较，第一个不同点就是 EMF3.EMF 表头部分中的 rclBounds 栏位。在 EMF2.EMF 中，它指出图像限定在座标(0x64,0x64)和(0xC8,0xC8)区域内。而在 EMF3.EMF 中，座标是(0x60,0x60)和(0xCC,0xCC)。这表示使用了较粗的笔。rclFrame 栏位（以 0.01mm 为单位指出图像大小）也受到影响。

EMF2.EMF 中的 nBytes 栏位(偏移量为 0x0030)显示该 metafile 长度为 0xFA 位元组，EMF3.EMF 中长度为 0x0188 位元组。EMF2.EMFmetafile 包含 7 个记录（一个表头纪录，5 个 GDI 函式呼叫和一个档案结束记录），但是 EMF3.EMF 档案包含 15 个记录。多出的 8 个记录是两个物件建立函式、4 个对 SelectObject

函式的呼叫和两个对 DeleteObject 函式的呼叫。

nHandles 栏位（在档案中偏移量为 0x0038）指出 GDI 物件的代号个数。该栏位的值总是比 metafile 使用的非内定物件数多一。（Platform SDK 文件解释这个多出来的一是「此表中保留的零索引」）。该栏位在 EMF2.EMF 的值为 1，而在 EMF3.EMF 中的值为 3，多出的数指出了画笔和画刷。

让我们跳到档案中偏移量为 0x0088 的地方，即第二个记录（表头纪录之後的第一个记录）。记录型态为 0x27，对应常数为 EMR_CREATE-BRUSHINDIRECT。该 metafile 记录用於 CreateBrushIndirect 函式，此函式需要指向 LOGBRUSH 结构的指标作为参数。该记录的长度为 0x18（或 24）位元组。

每个被选入 metafile 装置内容的非备用 GDI 物件得到一个号码，该号码从 1 开始编号。这在此记录的下 4 个位元组中指出，在 metafile 中的偏移量是 0x0090。此记录下面的 3 个 4 位元组栏位分别对应 LOGBRUSH 结构的 3 个栏位：0x00000000（BS_SOLID 的 lbStyle 栏位）、0x00FF0000（lbColor 栏位）和 0x00000000（lbHatch 栏位）。

下一个记录在 EMF3.EMF 中的偏移量为 0x00A0，记录型态为 0x25，或 EMR_SELECTOBJECT，是用於 SelectObject 呼叫的 metafile 记录。该记录的长度为 0x0C（或 12）位元组，下一个栏位是数值 0x01，指出它是选中的第一个 GDI 物件，这就是逻辑画刷。

EMF3.EMF 中的偏移量 0x00AC 是下一个记录，它的记录型态为 0x5F 或 EMR_EXTCREATEPEN。该记录有 0x34（或 52）个位元组。下一个 4 位元组栏位是 0x02，它表示这是在 metafile 内使用的第二个非备用 GDI 物件。

EMR_EXTCREATEPEN 记录的下 4 个栏位重复记录大小两次，之间用 0 栏位隔开：0x34、0x00、0x34 和 0x00。下一个栏位是 0x00010000，它是 PS_SOLID (0x00000000)与 PS_GEOMETRIC (0x00010000)组合的画笔样式。接下来是 5 个单元的宽度，紧接著是 ExtCreatePen 中使用的逻辑画刷结构的 3 个栏位，後接 0 栏位。

如果建立了自订的扩展画笔样式，EMR_EXTCREATEPEN 记录会超过 52 个位元组，这样会影响记录的第二栏位及两个重复的大小栏位。在描述 LOGBRUSH 结构的 3 个栏位後面不会是 0（像在 EMF3.EMF 中那样），而是指出了虚线和空格的数量。这後面接著用於虚线和空格长度的许多栏位。

EMF3.EMF 的下一个 12 位元组的栏位是指出第二个物件（画笔）的另一个 SelectObject 呼叫。接下来的 5 个记录与 EMF2.EMF 中的一样——一个 0x2B(EMR_RECTANGLE)的记录型态和两组 0x1B (EMR_MOVETOEX)和 0x36 (EMR_LINETO)记录。

这些绘图函式後面跟著两组 0x25(EMR_SELECTOBJECT)和 0x28(EMR_DELETEOBJECT)的 12 位元组记录。选择物件记录具有 0x80000007 和 0x80000000 的参数。在设定高位元时，它指出一个备用物件，在此例中是 0x07（对应 BLACK_PEN）和 0x00 (WHITE_BRUSH)。

DeleteObject 呼叫有 2 和 1 两个参数，用於在 metafile 中使用的两个非内定物件。虽然 DeleteObject 函式并不需要装置内容代号作为它的第一个参数，但 GDI 显然保留了 metafile 中使用的被程式删除的物件。

最後，metafile 以 0x0E（EMF_EOF）记录结束。

总结一下，每当非内定的 GDI 物件首次被选入 metafile 装置内容时，GDI 都会为该物件建立函式的记录编码（此例中，为 EMR_CREATEBRUSHINDIRECT 和 EMR_EXTCREATEPEN）。每个物件有一个依序从 1 开始的唯一数值，此数值由记录的第三个栏位表示。跟在此记录後的是引用该数值的 EMR_SELECTOBJECT 记录。以後，将物件选入 metafile 装置内容时（在中间时期没有被删除），就只需要 EMR_SELECTOBJECT 记录了。

metafile 和点阵图

现在，让我们做点稍微复杂的事，在 metafile 装置内容中绘制一幅点阵图，如程式 18-5 EMF4 所示。

程式 18-5 EMF4

```
EMF4.C
/*-------------------------------------------------------------------------
--
      EMF4.C --   Enhanced Metafile Demo #4
                                  (c) Charles Petzold, 1998
-------------------------------------------------------------------------
*/

#define OEMRESOURCE
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("EMF4") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;
```

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

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

     hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #4"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

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

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     BITMAP                        bm ;
     HBITMAP                       hbm ;
     HDC                      hdc, hdcEMF, hdcMem ;
     HENHMETAFILE                  hemf ;
     PAINTSTRUCT                   ps ;
     RECT                          rect ;

     switch (message)
     {
     case  WM_CREATE:
```

```
            hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf4.emf"), NULL,
          TEXT ("EMF4\0EMF Demo #4\0")) ;

                        hbm = LoadBitmap (NULL, MAKEINTRESOURCE (OBM_CLOSE)) ;

                        GetObject (hbm, sizeof (BITMAP), &bm) ;

                        hdcMem = CreateCompatibleDC (hdcEMF) ;

                        SelectObject (hdcMem, hbm) ;

                        StretchBlt (hdcEMF,100,100,100,100,
                  hdcMem,     0,0,bm.bmWidth, bm.bmHeight, SRCCOPY) ;

                        DeleteDC (hdcMem) ;
                  DeleteObject (hbm) ;

                        hemf = CloseEnhMetaFile (hdcEMF) ;

                        DeleteEnhMetaFile (hemf) ;
                        return 0 ;

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

                        GetClientRect (hwnd, &rect) ;
                        rect.left              =    rect.right        / 4 ;
                        rect.right             = 3 * rect.right       / 4 ;
                        rect.top               =    rect.bottom / 4 ;
                        rect.bottom            = 3 * rect.bottom      / 4 ;

                        hemf = GetEnhMetaFile (TEXT ("emf4.emf")) ;

                        PlayEnhMetaFile (hdc, hemf, &rect) ;
                        DeleteEnhMetaFile (hemf) ;
                        EndPaint (hwnd, &ps) ;
                        return 0 ;

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

为了方便，EMF4 载入由常数 OEM_CLOSE 指出的系统点阵图。在装置内容中显示点阵图的惯用方法是通过使用 CreateCompatibleDC 建立与目的装置内容（此例为 metafile 装置内容）相容的记忆体装置内容。然後，通过使用

SelectObject 将点阵图选入该记忆体装置内容并且从该记忆体装置内容呼叫 BitBlt 或 StretchBlt 把点阵图画到目的装置内容。结束後，删除记忆体装置内容和点阵图。

您会注意到 EMF4 也呼叫 GetObject 来确定点阵图的大小。这对 SelectObject 呼叫是很必要的。

首先，这份程式码储存 metafile 的空间对 GDI 来说就是个挑战。在 StretchBlt 呼叫前根本没有别的 GDI 函式去处理 metafile 的装置内容。因此，让我们来看一看 EMF4.EMF 里头是如何做的，图 18-5 只显示了一部分。

```
0000   01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00            ........d...d...
0010   C7 00 00 00 C7 00 00 00 35 0C 00 00 35 0C 00 00               ........5...5...
0020   4B 18 00 00 4B 18 00 00 20 45 4D 46 00 00 01 00            K...K....EMF....
0030   F0 0E 00 00 03 00 00 00 01 00 00 00 12 00 00 00            ................
0040   64 00 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00         d...............
0050   40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00         @...............
0060   00 00 00 00 45 00 4D 00 46 00 34 00 00 00 45 00            ....E.M.F.4...E.
0070   4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00            M.F...D.e.m.o...
0080   23 00 34 00 00 00 00 00 4D 00 00 00 54 0E 00 00            #.4.....M...T...
0090   64 00 00 00 64 00 00 00 C7 00 00 00 C7 00 00 00            d...d...........
00A0   64 00 00 00 64 00 00 00 64 00 00 00 64 00 00 00            d...d...d...d...
00B0   20 00 CC 00 00 00 00 00 00 00 00 00 00 00 80 3F            ...............?
00C0   00 00 00 00 00 00 00 00 00 00 80 3F 00 00 00 00            ...........?....
00D0   00 00 00 00 FF FF FF 00 00 00 00 00 6C 00 00 00            ............l...
00E0   28 00 00 00 94 00 00 00 C0 0D 00 00 28 00 00 00            (...........(...
00F0   16 00 00 00 28 00 00 00 28 00 00 00 16 00 00 00            ....(...(.......
0100   01 00 20 00 00 00 00 00 C0 0D 00 00 00 00 00 00            .. ..............
0110   00 00 00 00 00 00 00 00 00 00 00 00 C0 C0 C0 00            ................
0120   C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00         ................
. . . .
0ED0   C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 0E 00 00 00         ................
0EE0   14 00 00 00 00 00 00 00 10 00 00 00 14 00 00 00            ................
```

图 18-5 EMF4.EMF 的部分十六进位代码

此 metafile 只包含 3 个记录——表头纪录、0x0E54 位元组长度的 0x4D（或 EMR_STRETCHBLT）和档案结束记录。

我不解释该记录每个栏位的含义，但我会指出关键部分，以便理解 GDI 把 EMF4.C 中的一系列函式呼叫转化为单个 metafile 记录的方法。

GDI 已经把原始的与设备相关的点阵图转化为与装置无关的点阵图（DIB）。整个 DIB 储存在记录著自身大小的记录中。我想，在显示 metafile 和点阵图时，GDI 实际上使用 StretchDIBits 函式而不是 StretchBlt。或者，GDI 使用 CreateDIBitmap 把 DIB 转变回与设备相关的点阵图，然後使用记忆体装置内容及 StretchBlt 来显示点阵图。

EMR_STRETCHBLT 记录开始於 metafile 的偏移量 0x0088 处。DIB 储存在

metafile 中，以偏移量 0x00F4 开始，到 0x0EDC 处的记录结尾结束。DIB 以 BITMAPINFOHEADER 型态的 40 位元组的结构开始。在偏移量 0x011C 处接有 22 个图素行，每行 40 个图素。这是每图素 32 位元的 DIB，所以每个图素需要 4 个位元组。

列举 metafile 内容

当您希望存取 metafile 内的个别记录时，可以使用称作 metafile 列举的程序。如程式 18-6 EMF5 所示。此程式使用 metafile 来显示与 EMF3 相同的图像，但它是通过 metafile 列举来进行的。

程式 18-6 EMF5

```
EMF5.C
/*------------------------------------------------------------------------
--
      EMF5.C --   Enhanced Metafile Demo #5
                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                PSTR    szCmdLine,    int
iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("EMF5") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;

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

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

```
                return 0 ;
    }

     hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Demo #5"),
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL) ;

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

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

int CALLBACK EnhMetaFileProc (    HDC hdc, HANDLETABLE * pHandleTable,
                           CONST ENHMETARECORD * pEmfRecord,
                             int iHandles, LPARAM pData)
{
     PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfRecord, iHandles) ;
     return TRUE ;
}

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

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

                        GetClientRect (hwnd, &rect) ;

                        rect.left        =    rect.right        / 4 ;
                        rect.right       = 3 * rect.right       / 4 ;
                        rect.top    =    rect.bottom       / 4 ;
                        rect.bottom      = 3 * rect.bottom      / 4 ;
```

```
                hemf = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;

                EnumEnhMetaFile (hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
                      DeleteEnhMetaFile (hemf) ;
                      EndPaint (hwnd, &ps) ;
                      return 0 ;

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

此程式使用 EMF3 程式建立的 EMF3.EMF 档案，所以确定在执行此程式前先执行 EMF3 程式。同时，需要在 Visual C++环境中执行两个程式，以确保路径的正确。在处理 WM_PAINT 时，两个程式的主要区别是 EMF3 呼叫 PlayEnhMetaFile，而 EMF5 呼叫 EnumEnhMetaFile。PlayEnhMetaFile 函式有下面的语法：

```
PlayEnhMetaFile (hdc, hemf, &rect) ;
```

第一个参数是要显示的 metafile 的装置内容代号。第二个参数是增强型 metafile 代号。第三个参数是指向描述装置内容平面上矩形的 RECT 结构的指标。Metafile 图像大小被缩放过，以便刚好能够显示在不超过该矩形的区域内。

EnumEnhMetaFile 有 5 个参数，其中 3 个与 PlayEnhMetaFile 一样（虽然 RECT 结构的指标已经移到参数表的末尾）。

EnumEnhMetaFile 的第三个参数是列举函式的名称，它用於呼叫 EnhMetaFileProc。第四个参数是希望传递给列举函式的任意资料的指标，这里将该参数简单地设定为 NULL。

现在看一看列举函式。当呼叫 EnumEnhMetaFile 时，对於 metafile 中的每一个记录，GDI 都将呼叫 EnhMetaFileProc 一次，包括表头纪录和档案结束记录。通常列举函式传回 TRUE，但它可能传回 FALSE 以略过剩下的列举程序。

该列举函式有 5 个参数，稍後会描述它们。在这个程式中，我仅把前 4 个参数传递给 PlayEnhMetaFileRecord，它使 GDI 执行由该记录代表的函式呼叫，好像您明确地呼叫它一样。

EMF5 使用 EnumEnhMetaFile 和 PlayEnhMetaFileRecord 得到的结果与 EMF3 呼叫 PlayEnhMetaFile 得到的结果一样。区别在於 EMF5 现在直接介入了 metafile 的显示程序，并能够存取各个 metafile 记录。这是很有用的。

列举函式的第一个参数是装置内容代号。GDI 从 EnumEnhMetaFile 的第一个参数中简单地取得此代号。列举函式把该代号传递给 PlayEnhMetaFileRecord 来标识图像显示的目的装置内容。

我们先跳到列举函式的第三个参数，它是指向 ENHMETARECORD 型态结构的指标，前面已经提到过。这个结构描述实际的 metafile 记录，就像它亲自在 metafile 中编码一样。

您可以写一些程式码来检查这些记录。您也许不想把某些记录传送到 PlayEnhMetaFileRecord 函式。例如，在 EMF5.C 中，把下行插入到 PlayEnhMetaFileRecord 呼叫的前面：

```
if (pEmfRecord->iType != EMR_LINETO)
```

重新编译程序，执行它，将只看到矩形，而没有两条线。或使用下面的叙述：

```
if (pEmfRecord->iType != EMR_SELECTOBJECT)
```

这个小改变会让 GDI 用内定物件显示图像，而不是用 metafile 所建立的画笔和画刷。

程式中不应该修改 metafile 记录，不过先不要担心这一点。先来看一看程式 18-7 EMF6。

程式 18-7 EMF6

```
EMF6.C
/*-------------------------------------------------------------------------
--
      EMF6.C --   Enhanced Metafile Demo #6
                                  (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR   lpszCmdLine,   int
iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("EMF6") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;

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

```
     wndclass.lpszClassName       = szAppName ;

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

     hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #6"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

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

int CALLBACK EnhMetaFileProc (    HDC hdc, HANDLETABLE * pHandleTable,
                              CONST ENHMETARECORD * pEmfRecord,
                              int iHandles, LPARAM pData)
{
     ENHMETARECORD * pEmfr ;
     pEmfr = (ENHMETARECORD *) malloc (pEmfRecord->nSize) ;
     CopyMemory (pEmfr, pEmfRecord, pEmfRecord->nSize) ;
     if (pEmfr->iType == EMR_RECTANGLE)
                    pEmfr->iType = EMR_ELLIPSE ;
     PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, iHandles) ;
     free (pEmfr) ;
     return TRUE ;
}

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

     switch (message)
```

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

                    GetClientRect (hwnd, &rect) ;

                    rect.left                    =    rect.right  / 4 ;
                    rect.right                   = 3 * rect.right / 4 ;
                    rect.top               =    rect.bottom / 4 ;
                    rect.bottom                  = 3 * rect.bottom/ 4 ;

          hemf = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;
          EnumEnhMetaFile (hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
                                    DeleteEnhMetaFile (hemf) ;
                                    EndPaint (hwnd, &ps) ;
                                    return 0 ;

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

与 EMF5 一样，EMF6 使用 EMF3 程式建立的 EMF3.EMFmetafile，因此要在 Visual C++中执行这个程式之前先执行过 EMF3 程式。

EMF6 展示了如果在显示 metafile 之前要修改它们，解决方法是非常简单的：做个被修改过的副本出来就好了。您可以看到，列举程序一开始使用 malloc 配置一块 metafile 记录大小的记忆体，它是由传递给该函式的 pEmfRecord 结构的 nSize 栏位表示的。这个记忆体块的指标储存在变数 pEmfr 中，pEmfr 本身是指向 ENHMETARECORD 结构的指标。

程式使用 CopyMemory 把 pEmfRecord 指向的结构内容复制到 pEmfr 指向的结构中。现在我们就可以做些修改了。程式检查记录是否为 EMR_RECTANGLE 型态，如果是，则用 EMR_ELLIPSE 取代 iType 栏位。PEmfr 指标被传递到 PlayEnhMetaFileRecord 然後被释放。结果是程式画出一个椭圆而不是矩形。其他的内容的修改方式都是相同的。

当然，我们的小改变很容易起作用，因为 Rectangle 和 Ellipse 函式有同样的参数，这些参数都定义同一件事——图画的边界框。要进行范围更广的修改需要一些不同 metafile 记录格式的相关知识。

另一个可能性是插入一、两个额外的记录。例如，用下面的叙述代替 EMF6.C 中的 if 叙述：

```
if (pEmfr->iType == EMR_RECTANGLE)
```

```
{
      PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, nObjects) ;
      pEmfr->iType = EMR_ELLIPSE ;
}
```

无论何时出现Rectangle记录,程式都会处理此记录并把它更改为Ellipse,然後再显示。现在程式将画出矩形和椭圆。

现在讨论一下在列举 metafile 时 GDI 物件处理的方式。

在 metafile 表头中,ENHMETAHEADER 结构的 nHandles 栏位是比在 metafile 中建立的 GDI 物件数还要大的值。因此，对於 EMF5 和 EMF6 中的 metafile，此栏位是 3，表示画笔、画刷和其他东西。「其他东西」的具体内容，稍後我会说明。

您会注意到 EMF5 和 EMF6 中列举函式的倒数第二个参数，也称作 nHandles，它是同一个数，3。

列举函式的第二个参数是指向 HANDLETABLE 结构的指标，在 WINGDI.H 中定义如下：

```
typedef struct tagHANDLETABLE
{
      HGDIOBJ objectHandle [1] ;
}
HANDLETABLE ;
```

HGDIOBJ 资料型态是 GDI 物件的代号，被定义为 32 位元的指标，类似於所有其他 GDI 物件。这是那些带有一个元素的阵列栏位的结构之一。这意味著此栏位具有可变的长度。objectHandle 阵列中的元素数等於 nHandles，在此程式中是 3。

在列举函式中，可以使用以下运算式取得这些 GDI 物件代号：

```
pHandleTable->objectHandle[i]
```

对於 3 个代号，i 是 0、1 和 2。

每次呼叫列举函式时,阵列的第一个元素都将包含所列举的 metafile 代号。这就是前面提到的「其他东西」。

在第一次呼叫列举函式时，表的第二、第三个元素将是 0。它们是画笔和画刷代号的保留位置。

以下是列举函式运作的方式：metafile 中的第一个物件建立函式具有 EMR_CREATEBRUSHINDIRECT 的记录型态，此记录指出了物件编号 1。当把该记录传递给 PlayEnhMetaFileRecord 时，GDI 建立画刷并取得它的代号。此代号储存在 objectHandle 阵列的元素 1(第二个元素)中。当把第一个 EMR_SELECTOBJECT 记录传递给 PlayEnhMetaFileRecord 时，GDI 发现此物件编号为 1，并能够从表中找到该物件实际的代号，而把它用来呼叫 SelectObject。当 metafile 最後删

除画刷时，GDI 将 objectHandle 阵列的元素 1 设定回 0。

通过存取 objectHandle 阵列，可以使用例如 GetObjectType 和 GetObject 等呼叫取得在 metafile 中使用的物件资讯。

嵌入图像

列举 metafile 的最重要应用也许是在现有的 metafile 中嵌入其他图像(甚至是整个 metafile）。事实上，现有的 metafile 保持不变；真正进行的是建立包含现有 metafile 和新嵌入图像的新 metafile。基本的技巧是把 metafile 装置内容代号传递给 EnumEnhMetaFile，作为它的第一个参数。这使您能够在 metafile 装置内容上显示 metafile 记录和 GDI 函式呼叫。

在 metafile 命令序列的开头或结尾嵌入新图像是极简单的——就在 EMR_HEADER 记录之後或在 EMF_EOF 记录之前。然而，如果您熟悉现有的 metafile 结构，就可以把新的绘图命令嵌入所需的任何地方。如程式 18-8 EMF7 所示。

程式 18-8 EMF7

```
EMF7.C
/*------------------------------------------------------------------------
-
      EMF7.C -- Enhanced Metafile Demo #7
                                (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (   HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                        PSTR    lpszCmdLine,
int iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("EMF7") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;

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

```
     wndclass.lpszClassName      = szAppName ;

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

     hwnd = CreateWindow (szAppName, TEXT ("Enhanced Metafile Demo #7"),
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

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

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

int CALLBACK EnhMetaFileProc (    HDC hdc, HANDLETABLE * pHandleTable,
                            CONST ENHMETARECORD * pEmfRecord,
                          int iHandles, LPARAM pData)
{
     HBRUSH                 hBrush ;
     HPEN                        hPen ;
     LOGBRUSH               lb ;

     if (pEmfRecord->iType != EMR_HEADER && pEmfRecord->iType != EMR_EOF)
               PlayEnhMetaFileRecord   (hdc,   pHandleTable,   pEmfRecord,
iHandles) ;
     if (pEmfRecord->iType == EMR_RECTANGLE)
     {
               hBrush = SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
               lb.lbStyle = BS_SOLID ;
               lb.lbColor = RGB (0, 255, 0) ;
               lb.lbHatch = 0 ;

               hPen = SelectObject (hdc,
                                    ExtCreatePen (PS_SOLID | PS_GEOMETRIC,
5, &lb, 0, NULL)) ;
          Ellipse (hdc, 100, 100, 200, 200) ;
```

```
                DeleteObject (SelectObject (hdc, hPen)) ;
                SelectObject (hdc, hBrush) ;
     }
     return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     ENHMETAHEADER          emh ;
     HDC                                    hdc, hdcEMF ;
     HENHMETAFILE           hemfOld, hemf ;
     PAINTSTRUCT                 ps ;
     RECT                                   rect ;

     switch (message)
     {
     case  WM_CREATE:

                                 // Retrieve existing metafile and header

                     hemfOld = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;

                GetEnhMetaFileHeader (hemfOld, sizeof (ENHMETAHEADER), &emh) ;

                                                 // Create a new metafile DC

                hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf7.emf"), NULL,
                                 TEXT ("EMF7\0EMF Demo #7\0")) ;

                                      // Enumerate the existing metafile

                     EnumEnhMetaFile (hdcEMF, hemfOld, EnhMetaFileProc, NULL,
                                 (RECT *) & emh.rclBounds) ;

                                                 // Clean up

                hemf = CloseEnhMetaFile (hdcEMF) ;

                DeleteEnhMetaFile (hemfOld) ;
                     DeleteEnhMetaFile (hemf) ;
                     return 0 ;

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

                     GetClientRect (hwnd, &rect) ;
                     rect.left                  =    rect.right  / 4 ;
```

```
                    rect.right                  = 3 * rect.right / 4 ;
                    rect.top              =     rect.bottom   / 4 ;
                    rect.bottom            = 3 * rect.bottom   / 4 ;

                    hemf = GetEnhMetaFile (TEXT ("emf7.emf")) ;

                    PlayEnhMetaFile (hdc, hemf, &rect) ;
                    DeleteEnhMetaFile (hemf) ;
                    EndPaint (hwnd, &ps) ;
                    return 0 ;

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

EMF7 使用 EMF3 程式建立的 EMF3.EMF，所以在执行 EMF7 之前要执行 EMF3 程式建立 metafile。

EMF7 中的 WM_PAINT 处理使用 PlayEnhMetaFile 而不是 EnumEnhMetaFile，而且 WM_CREATE 处理有很大的差别。

首先,程式通过呼叫 GetEnhMetaFile 取得 EMF3.EMF 档案的 metafile 代号，还呼叫 GetEnhMetaFileHeader 得到增强型 metafile 表头记录，目的是在後面的 EnumEnhMetaFile 呼叫中使用 rclBounds 栏位。

接下来，程式建立新的 metafile 档案，名为 EMF7.EMF。CreateEnhMetaFile 函式为 metafile 传回装置内容代号。然後，使用 EMF7.EMF 的 metafile 装置内容代号和 EMF3.EMF 的 metafile 代号呼叫 EnumEnhMetaFile。

现在来看一看 EnhMetaFileProc。如果被列举的记录不是表头纪录或档案结束记录，函式就呼叫 PlayEnhMetaFileRecord 把记录转换为新的 metafile 装置内容（并不一定排除表头纪录或档案结束记录，但它们会使 metafile 变大）。

如果刚转换的记录是 Rectangle 呼叫，则函式建立画笔用绿色的轮廓线和透明的内部来绘制椭圆。注意程式中经由储存先前的画笔和画刷代号来恢复装置内容状态的方法。在此期间，所有这些函式都被插入到 metafile 中（记住，也可以使用 PlayEnhMetaFile 在现有的 metafile 中插入整个 metafile）。

回到 WM_CREATE 处理，程式呼叫 CloseEnhMetaFile 取得新 metafile 的代号。然後，它删除两个 metafile 代号，将 EMF3.EMF 和 EMF7.EMF 档案留在磁片上。

从程式显示输出中可以很明显地看到，椭圆是在矩形之後两条交叉线之前绘制的。

增强型 metafile 阅览器和印表机

使用剪贴簿转换增强型metafile非常简单,剪贴簿型态是CF_ENHMETAFILE。GetClipboardData 函式传回增强型 metafile 代号，SetClipboardData 也使用该 metafile 代号。复制 metafile 时可以使用 CopyEnhMetaFile 函式。如果把增强型 metafile 放在剪贴簿中，Windows 会让需要旧格式的那些程式也可以使用它。如果在剪贴簿中放置旧格式的 metafile，Windows 将也会自动视需要把内容转换为增强型 metafile 的格式。

程式 18-9 EMFVIEW 所示为在剪贴簿中传送 metafile 的程式码，它也允许载入、储存和列印 metafile。

程式 18-9 EMFVIEW

```
EMFVIEW.C
/*-------------------------------------------------------------------------
-
      EMFVIEW.C --     View Enhanced Metafiles
                                              (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

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

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

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

     hwnd = CreateWindow (  szAppName, TEXT ("Enhanced Metafile Viewer"),
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            NULL, NULL, hInstance, NULL) ;

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

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

HPALETTE CreatePaletteFromMetaFile (HENHMETAFILE hemf)
{
     HPALETTE                    hPalette ;
     int                         iNum ;
     LOGPALETTE *     plp ;

     if (!hemf)
                       return NULL ;
     if (0 == (iNum = GetEnhMetaFilePaletteEntries (hemf, 0, NULL)))
                       return NULL ;
     plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
     plp->palVersion    = 0x0300 ;
     plp->palNumEntries = iNum ;

     GetEnhMetaFilePaletteEntries (hemf, iNum, plp->palPalEntry) ;
     hPalette = CreatePalette (plp) ;
     free (plp) ;
     return hPalette ;
}
```

```
LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     static DOCINFO   di = { sizeof (DOCINFO), TEXT ("EmfView: Printing") } ;
     static HENHMETAFILE hemf ;
     static OPENFILENAME ofn ;
     static PRINTDLG             printdlg = { sizeof (PRINTDLG) } ;
     static TCHAR                      szFileName  [MAX_PATH],  szTitleName
[MAX_PATH] ;
     static TCHAR                      szFilter[] =
                                                              TEXT
("Enhanced Metafiles (*.EMF)\0*.emf\0")
                                                              TEXT
("All Files (*.*)\0*.*\0\0") ;
     BOOL                                             bSuccess ;
     ENHMETAHEADER                    header ;
     HDC                                              hdc, hdcPrn ;
     HENHMETAFILE                     hemfCopy ;
     HMENU                                            hMenu ;
     HPALETTE                                    hPalette ;
     int                                              i, iLength, iEnable ;
     PAINTSTRUCT                           ps ;
     RECT                                             rect ;
     PTSTR                                            pBuffer ;

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

```
                    ofn.lpfnHook                        = NULL ;
                    ofn.lpTemplateName                  = NULL ;
                    return 0 ;

     case  WM_INITMENUPOPUP:
                    hMenu = GetMenu (hwnd) ;

                    iEnable = hemf ? MF_ENABLED : MF_GRAYED ;

          EnableMenuItem (hMenu, IDM_FILE_SAVE_AS,            iEnable) ;
          EnableMenuItem (hMenu, IDM_FILE_PRINT,              iEnable) ;
          EnableMenuItem (hMenu, IDM_FILE_PROPERTIES,         iEnable) ;
          EnableMenuItem (hMenu, IDM_EDIT_CUT,                iEnable) ;
               EnableMenuItem (hMenu, IDM_EDIT_COPY,
     iEnable) ;
               EnableMenuItem (hMenu, IDM_EDIT_DELETE,
     iEnable) ;
               EnableMenuItem (hMenu, IDM_EDIT_PASTE,
               IsClipboardFormatAvailable (CF_ENHMETAFILE) ?
               MF_ENABLED : MF_GRAYED) ;
               return 0 ;

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

                                   ofn.Flags = 0 ;

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

                         // If there's an existing EMF, get rid of it.

                                   if (hemf)
                                   {
                                             DeleteEnhMetaFile (hemf) ;
                                             hemf = NULL ;
                                   }
                                             // Load the EMF into memory

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

                              hemf = GetEnhMetaFile (szFileName) ;

                              ShowCursor (FALSE) ;
```

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

               // Invalidate the client area for later update

                    InvalidateRect (hwnd, NULL, TRUE) ;

                    if (hemf == NULL)
                    {
               MessageBox (      hwnd, TEXT ("Cannot load metafile"),
                           szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    }
                    return 0 ;

               case  IDM_FILE_SAVE_AS:
                    if (!hemf)
                              return 0 ;

                              // Show the File Save dialog box

                    ofn.Flags = OFN_OVERWRITEPROMPT ;

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

                              // Save the EMF to disk file

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

                    hemfCopy = CopyEnhMetaFile (hemf, szFileName) ;

                    ShowCursor (FALSE) ;
                    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
                    if (hemfCopy)
                    {
                              DeleteEnhMetaFile (hemf) ;
                              hemf = hemfCopy ;
                    }
                    else
               MessageBox (      hwnd, TEXT ("Cannot save metafile"),
                           szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    return 0 ;

               case  IDM_FILE_PRINT:
                    // Show the Print dialog box and get printer DC

               printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;
```

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

                    if (NULL == (hdcPrn = printdlg.hDC))
                    {
          MessageBox ( hwnd, TEXT ("Cannot obtain printer DC"),
                     szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                              return 0 ;
                    }
                    // Get size of printable area of page

                rect.left   = 0 ;
                rect.right = GetDeviceCaps (hdcPrn, HORZRES) ;
                rect.top          = 0 ;
                rect.bottom = GetDeviceCaps (hdcPrn, VERTRES) ;

                    bSuccess = FALSE ;

                              // Play the EMF to the printer

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

          if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
                {
                         PlayEnhMetaFile (hdcPrn, hemf, &rect) ;

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

                    DeleteDC (hdcPrn) ;

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

          case  IDM_FILE_PROPERTIES:
                    if (!hemf)
                              return 0 ;

                iLength = GetEnhMetaFileDescription (hemf, 0, NULL) ;
```

```
                        pBuffer = malloc ((iLength + 256) * sizeof (TCHAR)) ;

                                    GetEnhMetaFileHeader     (hemf,     sizeof
(ENHMETAHEADER), &header) ;

                                          // Format header file information

                                    i  = wsprintf (pBuffer,
                                          TEXT ("Bounds = (%i, %i) to (%i, %i)
pixels\n"),

      header.rclBounds.left, header.rclBounds.top,

      header.rclBounds.right, header.rclBounds.bottom) ;

                                    i += wsprintf (pBuffer + i,
                                                      TEXT ("Frame =
(%i, %i) to (%i, %i) mms\n"),

      header.rclFrame.left, header.rclFrame.top,

      header.rclFrame.right, header.rclFrame.bottom) ;

                                    i += wsprintf (pBuffer + i,
                                                      TEXT
("Resolution = (%i, %i) pixels")
                                                      TEXT (" = (%i,
%i) mms\n"),

      header.szlDevice.cx, header.szlDevice.cy,

      header.szlMillimeters.cx,

      header.szlMillimeters.cy) ;

                        i += wsprintf (pBuffer + i,
                              TEXT ("Size = %i, Records = %i, ")
                              TEXT ("Handles = %i, Palette entries = %i\n"),
                              header.nBytes, header.nRecords,
                              header.nHandles, header.nPalEntries) ;
                        // Include the metafile description, if present

                  if (iLength)
                  {
                        i += wsprintf (pBuffer + i, TEXT ("Description = ")) ;
                        GetEnhMetaFileDescription (hemf, iLength, pBuffer + i) ;
                                    pBuffer [lstrlen (pBuffer)] = '\t' ;
                  }
```

```
          MessageBox (hwnd, pBuffer, TEXT ("Metafile Properties"), MB_OK) ;
                              free (pBuffer) ;
                              return 0 ;

               case  IDM_EDIT_COPY:
               case  IDM_EDIT_CUT:
                              if (!hemf)
                                        return 0 ;

                         // Transfer metafile copy to the clipboard

                         hemfCopy = CopyEnhMetaFile (hemf, NULL) ;

                         OpenClipboard (hwnd) ;
                         EmptyClipboard () ;
                         SetClipboardData (CF_ENHMETAFILE, hemfCopy) ;
                         CloseClipboard () ;

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

          case  IDM_EDIT_PASTE:
                         OpenClipboard (hwnd) ;
                         hemfCopy = GetClipboardData (CF_ENHMETAFILE) ;

                              CloseClipboard () ;
                              if (hemfCopy && hemf)
                              {
                                        DeleteEnhMetaFile (hemf) ;
                                        hemf = NULL ;
                              }

                              hemf = CopyEnhMetaFile (hemfCopy, NULL) ;
                              InvalidateRect (hwnd, NULL, TRUE) ;
                              return 0 ;

          case  IDM_APP_ABOUT:
                              MessageBox (     hwnd,       TEXT
```

```
("Enhanced Metafile Viewer\n")
                      TEXT ("(c) Charles Petzold, 1998"),

     szAppName, MB_OK) ;
                                return 0 ;

          case  IDM_APP_EXIT:
                                SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
                                return 0 ;
               }
               break ;

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

                      if (hemf)
                      {
                            if (  hPalette = CreatePaletteFromMetaFile (hemf))
                                  {
                                        SelectPalette (hdc, hPalette, FALSE) ;
                                        RealizePalette (hdc) ;
                                  }
                                  GetClientRect (hwnd, &rect) ;
                                  PlayEnhMetaFile (hdc, hemf, &rect) ;

                                  if (hPalette)
                                        DeleteObject (hPalette) ;
                      }
                      EndPaint (hwnd, &ps) ;
                      return 0 ;

     case  WM_QUERYNEWPALETTE:
               if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
                                return FALSE ;

                      hdc = GetDC (hwnd) ;
                      SelectPalette (hdc, hPalette, FALSE) ;
                      RealizePalette (hdc) ;
                      InvalidateRect (hwnd, NULL, FALSE) ;

                      DeleteObject (hPalette) ;
                      ReleaseDC (hwnd, hdc) ;
                      return TRUE ;

     case  WM_PALETTECHANGED:
                      if ((HWND) wParam == hwnd)
                                        break ;
```

```
                if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
                                        break ;

                        hdc = GetDC (hwnd) ;
                        SelectPalette (hdc, hPalette, FALSE) ;
                        RealizePalette (hdc) ;
                        UpdateColors (hdc) ;

                        DeleteObject (hPalette) ;
                        ReleaseDC (hwnd, hdc) ;
                        break ;

      case  WM_DESTROY:
                        if (hemf)
                              DeleteEnhMetaFile (hemf) ;

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

EMFVIEW.RC （摘录）

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

//////////////////////////////////////////////////////////////////////////////
/
// Menu
EMFVIEW MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
       MENUITEM "&Open\tCtrl+O",  IDM_FILE_OPEN
       MENUITEM "Save &As...",    IDM_FILE_SAVE_AS
            MENUITEM SEPARATOR
            MENUITEM "&Print...\tCtrl+P",IDM_FILE_PRINT
       MENUITEM SEPARATOR
            MENUITEM "&Properties",       IDM_FILE_PROPERTIES
            MENUITEM SEPARATOR
            MENUITEM "E&xit",             IDM_APP_EXIT
END
      POPUP "&Edit"
      BEGIN
       MENUITEM "Cu&t\tCtrl+X",                    IDM_EDIT_CUT
       MENUITEM "&Copy\tCtrl+C",              IDM_EDIT_COPY
            MENUITEM "&Paste\tCtrl+V",              IDM_EDIT_PASTE
       MENUITEM "&Delete\tDel",               IDM_EDIT_DELETE
```

```
      END
      POPUP "Help"
      BEGIN
       MENUITEM "&About EmfView...",          IDM_APP_ABOUT
      END
END

//////////////////////////////////////////////////////////////////////////
/
// Accelerator
EMFVIEW ACCELERATORS DISCARDABLE
BEGIN
"C",IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
"O",IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
"P",IDM_FILE_PRINT,VIRTKEY, CONTROL, NOINVERT
"V",IDM_EDIT_PASTE,VIRTKEY, CONTROL, NOINVERT
VK_DELETE,IDM_EDIT_DELETE,VIRTKEY, NOINVERT
"X",IDM_EDIT_CUT,VIRTKEY, CONTROL, NOINVERT
END
 RESOURCE.H （摘录）
// Microsoft Developer Studio generated include file.
// Used by EmfView.rc

#define          IDM_FILE_OPEN           40001
#define          IDM_FILE_SAVE_AS         40002
#define          IDM_FILE_PRINT          40003
#define          IDM_FILE_PROPERTIES      40004
#define          IDM_APP_EXIT            40005
#define          IDM_EDIT_CUT            40006
#define          IDM_EDIT_COPY           40007
#define          IDM_EDIT_PASTE          40008
#define          IDM_EDIT_DELETE          40009
#define          IDM_APP_ABOUT           40010
```

EMFVIEW 也支援完整的调色盘处理，以便支援有调色盘编码资讯的 metafile 。（透过呼叫 Selectpalette 来进行）。该程式在 CreatePaletteFromMetaFile 函式中处理调色盘，在处理 WM_PAINT 显示 metafile 以及处理 WM_QUERYNEWPALETTE 和 WM_PALETTECHANGED 讯息时，呼叫这个函式。

在回应功能表中的「Print」命令时，EMFVIEW 显示普通的印表机对话方块，然後取得页面中可列印区域的大小。Metafile 被缩放成适当尺寸以填入整个区域。EMFVIEW 在视窗中以类似方 式显示 metafile。

「File」功能表中的「Properties」项使 EMFVIEW 显示包含 metafile 表头资讯的讯息方块。

如果列印本章前面建立的 EMF2.EMFmetafile 图像，您将会发现用高解析度

的印表机列印出的线条非常细，几乎看不清楚线条的锯齿。列印向量图像时应该使用较宽的画笔（例如，一点宽）。本章後面所示的直尺图像就是这样做的。

显示精确的 metafile 图像

Metafile 图像的好处在於它能够以任意大小缩放并且仍能保持一定的逼真度。这是因为 metafile 通常由一系列向量图形的基本图形组成，基本图形是指线条、填入的区域以及轮廓字体等等。扩大或缩小图像只是简单地缩放定义这些基本图形的所有座标点。另一方面，对点阵图来说，压缩图像会遗漏整行列的图素，因而失去重要的显示资讯。

当然，metafile 的压缩并不是完美无缺的。我们所使用的图形输出设备的图素大小是有限的。当 metafile 图像压缩到一定大小时，组成 metafile 的大量线条会变成模糊的斑点，同时区域填入图案和混色看起来也很奇怪。如果 metafile 中包含嵌入的点阵图或旧的点阵字体，同样会引起类似的问题。

尽管如此，大多数情况下 metafile 可以任意地缩放。这在把 metafile 放入文书处理或桌上印刷文件内时非常有用。一般来说，在上述的应用程式中选择 metafile 图像时，会出现围绕图像的矩形，您可以用滑鼠拖动该矩形，将它缩放为任意大小。图像送到印表机时，它也具有同样对应的大小。

然而，有时任意缩放 metafile 并不是个好主意。例如：假设您有一个储存著存款客户签名样本的银行系统，这些签名以一系列折线的方式储存在 metafile 中。将 metafile 变宽或变高会使签名变形，因此应该保持图像的纵横比一致。

在前面的范例程式中，是以显示区域的大小来确定 PlayEnhMetaFile 呼叫使用的围绕矩形范围。所以，如果改变程式表单的大小，也就改变了图像的大小。这与在文书处理文件中改变 metafile 图像大小的概念相似。

正确地显示 metafile 图像（以特定的度量单位或用适当的纵横比），需要使用 metafile 表头中的大小资讯并根据此资讯设定矩形结构。

在本章剩下的范例程式中将使用名为 EMF.C 的程式架构，它包括列印处理的程式码、资源描述档 EMF.RC 和表头档案 RESOURCE.H。程式 18-10 显示了这些档案以及 EMF8.C 程式，该程式使用这些档案显示一把 6 英寸的直尺。

程式 18-10　EMF8

```
EMF8.C
/*-------------------------------------------------------------------------
-
      EMF8.C -- Enhanced Metafile Demo #8
                                  (c) Charles Petzold, 1998
-------------------------------------------------------------------------
```

```
-*/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF8") ;
TCHAR szTitle          [] = TEXT ("EMF8: Enhanced Metafile Demo #8") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
     int                   iAdj, i, iHeight ;
     LOGFONT          lf ;
     TCHAR                 ch ;

     iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
                           // Black pen with 1-point width
     SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
                           //  Rectangle  surrounding  entire  pen  (with
adjustment)
     Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
                           // Tick marks
     for (i = 1 ; i < 96 ; i++)
     {
               if (i % 16 == 0) iHeight = cy /  2 ;          // inches
          else  if (i %  8 == 0) iHeight = cy /  3 ;     // half inches
          else  if (i %  4 == 0) iHeight = cy /  5 ;     // quarter inches
          else  if (i %  2 == 0) iHeight = cy /  8 ;     // eighths
          else                   iHeight = cy / 12 ;          // sixteenths

               MoveToEx (hdc, i * cx / 96, cy, NULL) ;
               LineTo   (hdc, i * cx / 96, cy - iHeight) ;
     }
                           // Create logical font
     FillMemory (&lf, sizeof (lf), 0) ;
     lf.lfHeight = cy / 2 ;
     lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

     SelectObject     (hdc, CreateFontIndirect (&lf)) ;
     SetTextAlign     (hdc, TA_BOTTOM | TA_CENTER) ;
     SetBkMode        (hdc, TRANSPARENT) ;

                           // Display numbers

     for (i = 1 ; i <= 5 ; i++)
     {
                    ch = (TCHAR) (i + '0') ;
                    TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
     }
                           // Clean up
     DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
```

```
     DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
     HDC                         hdcEMF ;
     HENHMETAFILE hemf ;
     int                         cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

     hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf8.emf"), NULL,
                          TEXT ("EMF8\0EMF Demo #8\0")) ;
     if (hdcEMF == NULL)
                    return ;
     cxMms             = GetDeviceCaps (hdcEMF, HORZSIZE) ;
     cyMms             = GetDeviceCaps (hdcEMF, VERTSIZE) ;
     cxPix             = GetDeviceCaps (hdcEMF, HORZRES) ;
     cyPix             = GetDeviceCaps (hdcEMF, VERTRES) ;

     xDpi        = cxPix * 254 / cxMms / 10 ;
     yDpi        = cyPix * 254 / cyMms / 10 ;

     DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
     hemf = CloseEnhMetaFile (hdcEMF) ;
     DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
     ENHMETAHEADER          emh ;
     HENHMETAFILE           hemf ;
     int                                   cxImage, cyImage ;
     RECT                                  rect ;

     hemf = GetEnhMetaFile (TEXT ("emf8.emf")) ;
     GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
     cxImage = emh.rclBounds.right - emh.rclBounds.left ;
     cyImage = emh.rclBounds.bottom - emh.rclBounds.top ;

     rect.left              = (cxArea - cxImage) / 2 ;
     rect.right             = (cxArea + cxImage) / 2 ;
     rect.top          = (cyArea - cyImage) / 2 ;
     rect.bottom            = (cyArea + cyImage) / 2 ;

     PlayEnhMetaFile (hdc, hemf, &rect) ;
     DeleteEnhMetaFile (hemf) ;
}
 EMF.C
/*------------------------------------------------------------------------
```

```
--
      EMF.C --    Enhanced Metafile Demonstration Shell Program
                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
#include <commdlg.h>
#include "..\\emf8\\resource.h"

extern void CreateRoutine (HWND) ;
extern void PaintRoutine  (HWND, HDC, int, int) ;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HANDLE hInst ;
extern TCHAR szClass [] ;
extern TCHAR szTitle [] ;

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

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

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

      hwnd = CreateWindow (  szClass, szTitle,
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, CW_USEDEFAULT,
```

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

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

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

BOOL PrintRoutine (HWND hwnd)
{
     static DOCINFO         di ;
     static PRINTDLG        printdlg = { sizeof (PRINTDLG) } ;
     static TCHAR                szMessage [32] ;
     BOOL                                       bSuccess = FALSE ;
     HDC                                        hdcPrn ;
     int                                        cxPage, cyPage ;

     printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;
     if (!PrintDlg (&printdlg))
                    return TRUE ;
     if (NULL == (hdcPrn = printdlg.hDC))
                    return FALSE ;
     cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
     cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

     lstrcpy (szMessage, szClass) ;
     lstrcat (szMessage, TEXT (": Printing")) ;

     di.cbSize                  = sizeof (DOCINFO) ;
     di.lpszDocName         = szMessage ;

     if (StartDoc (hdcPrn, &di) > 0)
     {
               if (StartPage (hdcPrn) > 0)
               {
                         PaintRoutine (hwnd, hdcPrn, cxPage, cyPage) ;
                         if (EndPage (hdcPrn) > 0)
                         {
                                   EndDoc (hdcPrn) ;
                                   bSuccess = TRUE ;
                         }
               }
```

```
     }
     DeleteDC (hdcPrn) ;
     return bSuccess ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     BOOL                         bSuccess ;
     static int             cxClient, cyClient ;
     HDC                          hdc ;
     PAINTSTRUCT            ps ;

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

     case  WM_COMMAND:
                       switch (wParam)
                       {
                       case  IDM_PRINT:
                                  SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                                  ShowCursor (TRUE) ;

                                  bSuccess = PrintRoutine (hwnd) ;

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

                                       if (!bSuccess)
          MessageBox (hwnd, TEXT ("Error encountered during printing"),
                         szClass, MB_ICONASTERISK | MB_OK) ;
                              return 0 ;

                       case  IDM_EXIT:
                                       SendMessage (hwnd, WM_CLOSE, 0, 0) ;
                                       return 0 ;

                       case  IDM_ABOUT:
          MessageBox (hwnd, TEXT ("Enhanced Metafile Demo Program\n")
                         TEXT ("Copyright (c) Charles Petzold, 1998"),
                         szClass, MB_ICONINFORMATION | MB_OK) ;
                              return 0 ;
                       }
                       break ;
```

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

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

                        PaintRoutine (hwnd, hdc, cxClient, cyClient) ;

                        EndPaint (hwnd, &ps) ;
                        return 0 ;

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

EMF.RC （摘录）

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

//////////////////////////////////////////////////////////////////////////////
/
// Menu
EMF MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
                        MENUITEM "&Print...",
      IDM_PRINT
                        MENUITEM SEPARATOR
                        MENUITEM "E&xit",
      IDM_EXIT
      END
      POPUP "&Help"
      BEGIN
                        MENUITEM "&About...",
      IDM_ABOUT
      END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Emf.rc
//
#define IDM_PRINT             40001
```

```
#define IDM_EXIT              40002
#define IDM_ABOUT             40003
```

在处理 WM_CREATE 讯息处理期间，EMF.C 呼叫名为 CreateRoutine 的外部函式，该函式建立 metafile。EMF.C 在两个地方呼叫 PaintRoutine 函式：一处在 WM_PAINT 讯息处理期间，另一处在函式 PrintRoutine 中以回应功能表命令列印图像。

因为现代的印表机通常比视讯显示器有更高的解析度，列印 metafile 的能力是测试以特定大小处理图像能力的重要工具。当 EMF8 建立的 metafile 图像以特定大小显示时，最有意义。该图像是一把 6 英寸长 1 英寸宽的直尺，每英寸分为十六格，数字从 1 到 5 为 TrueType 字体。

要绘制一把 6 英寸的直尺，需要知道一些设备解析度的知识。EMF8.C 中的 CreateRoutine 函式首先建立 metafile，然後使用从 CreateEnhMetaFile 传回的装置内容代号呼叫 GetDeviceCaps 四次。这些呼叫取得单位分别为毫米和图素的显示平面的高度与宽度。

这听起来有点怪。Metafile 装置内容通常是作为 GDI 绘制命令的储存媒介，它不是像视讯显示器或印表机的真正设备，那么它的宽度和高度从何而来？

您可能已经想起来了，CreateEnhMetaFile 的第一个参数被称作「参考装置内容」。GDI 用这为 metafile 建立设备特徵。如果参数设定为 NULL（如 EMF8 中），GDI 就把显示器作为参考装置内容。因而，当 EMF8 使用装置内容呼叫 GetDeviceCaps 时，它实际上取得有关显示器的资讯。

EMF8.C 以图素大小除以毫米大小并乘以 25.4（1 英寸为 25.4 毫米）计算以每英寸的点数为单位的解析度。

即使我们非常认真地以 metafile 直尺的正确大小绘制它，但是这样子作还是不够的。PlayEnhMetaFile 函式在显示图像时，使用作为最後一个参数传递给它的矩形来缩放图像大小，因此该矩形必须设定为直尺的大小。

由於此原因，EMF8 中的 PaintRoutine 函式呼叫 GetEnhMetaFileHeader 函式来取得 metafile 的表头资讯。ENHMETAHEADER 结构的 rclBounds 栏位指出以图素为单位的 metafile 图像的围绕矩形。程式使用此资讯使直尺位於显示区域中央，如图 18-6 所示。

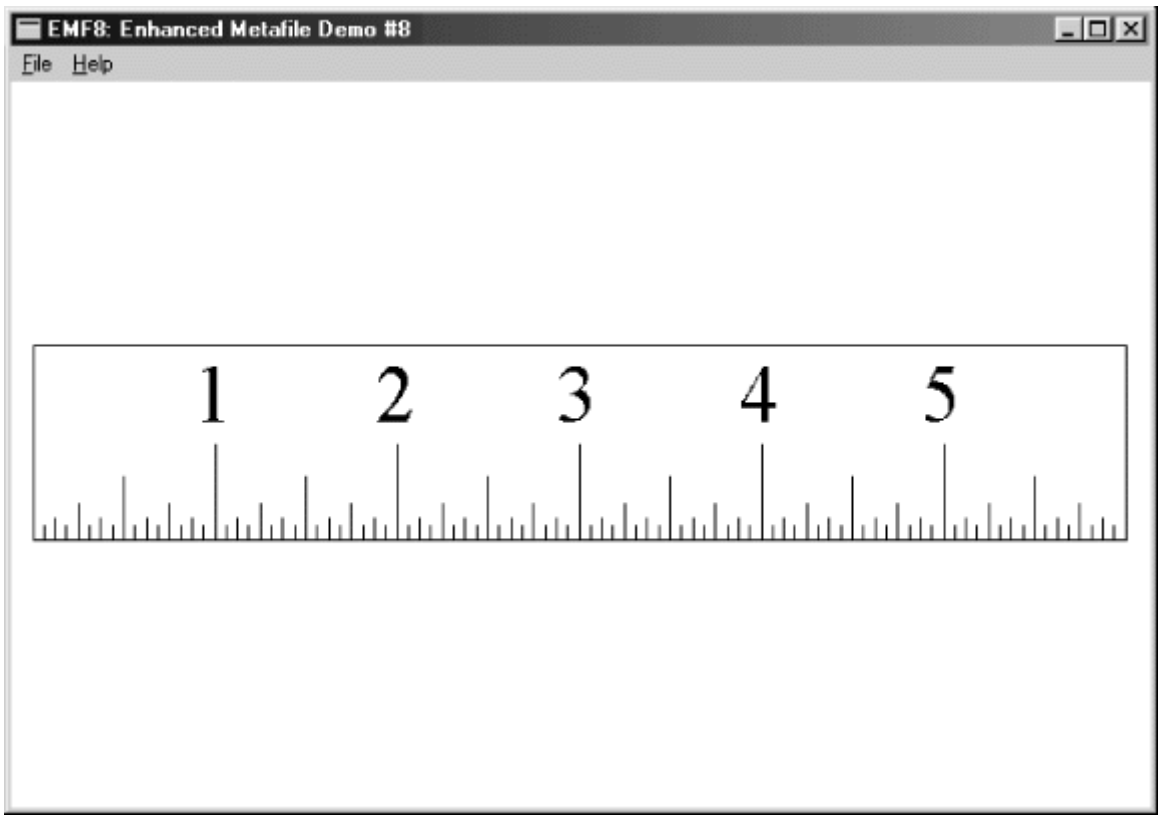

图 18-6 EMF8 得萤幕显示

记住，如果您拿直尺与萤幕中的直尺比较时，两者并不一定非常吻合。如同第五章中所论述的，显示器只能近似地实际显示尺寸。

既然这样做好像有用了，现在就来试著列印图像。哇！如果您有一台 300dpi 的雷射印表机，那么列印出的直尺的宽将会是 11/3 英寸。这是由於我们依据视讯显示器的图素尺寸来列印。虽然您可能认为这把小尺很可爱，但它不是我们所需要的。让我们再试一试。

ENHMETAHEADER 结构包括两个描述图像大小的矩形结构。第一个是 rclBounds，EMF8 使用这个，它以图素为单位给出图像的大小。第二为 rclFrame，它以 0.01 毫米为单位给出图像的大小。这两个栏位之间的关系是由最初建立 metafile 时使用的参考装置内容决定的，在此情况下为显示器（metafile 表头也包括两个名为 szlDevice 和 szlMillimeters 的栏位，它们是 SIZEL 结构，分别以图素单位和毫米单位指出了参考设备的大小，这与从 GetDeviceCaps 得到的资讯一样）。

EMF9 使用图像的毫米大小资讯，如程式 18-11 所示。

程式 18-11 EMF9

```
EMF9.C
/*------------------------------------------------------------------------
-
      EMF9.C -- Enhanced Metafile Demo #9
                                    (c) Charles Petzold, 1998
------------------------------------------------------------------------
```

```
*/

#include <windows.h>
#include <string.h>

TCHAR szClass          [] = TEXT ("EMF9") ;
TCHAR szTitle          [] = TEXT ("EMF9: Enhanced Metafile Demo #9") ;

void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
      ENHMETAHEADER          emh ;
      HENHMETAFILE           hemf ;
      int                                    cxMms,   cyMms,   cxPix,   cyPix,
cxImage, cyImage ;
      RECT                                   rect ;

      cxMms            = GetDeviceCaps (hdc, HORZSIZE) ;
      cyMms            = GetDeviceCaps (hdc, VERTSIZE) ;
      cxPix            = GetDeviceCaps (hdc, HORZRES) ;
      cyPix            = GetDeviceCaps (hdc, VERTRES) ;

      hemf = GetEnhMetaFile (TEXT ("..\\emf8\\emf8.emf")) ;
      GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
      cxImage                = emh.rclFrame.right - emh.rclFrame.left ;
      cyImage                = emh.rclFrame.bottom - emh.rclFrame.top ;

      cxImage                = cxImage * cxPix / cxMms / 100 ;
      cyImage                = cyImage * cyPix / cyMms / 100 ;

      rect.left              = (cxArea - cxImage) / 2 ;
      rect.right             = (cxArea + cxImage) / 2 ;
      rect.top         = (cyArea - cyImage) / 2 ;
      rect.bottom            = (cyArea + cyImage) / 2 ;

      PlayEnhMetaFile (hdc, hemf, &rect) ;
      DeleteEnhMetaFile (hemf) ;
}
```

EMF9 使用 EMF8 建立的 metafile，因此确定执行 EMF8。

EMF9 中的 PaintRoutine 函式首先使用目的装置内容呼叫 GetDeviceCaps 四次。像在 EMF8 中的 CreateRoutine 函式一样，这些呼叫提供有关设备解析度的资讯。在得到 metafile 代号之後，它取得表头结构并使用 rclFrame 栏位来计算以 0.01 毫米为单位的 metafile 图像大小。这是第一步。

然後，函式通过乘以输出设备的图素大小、除以毫米大小再除以 100（因为度量尺寸以 0.01 毫米为单位）将此大小转换为图素大小。现在，PaintRoutine 函式具有以图素为单位的直尺大小——与显示器无关。这是适合目的设备的图素大小，而且很容易使图像居中对齐。

就显示器而言，EMF9 的显示与 EMF8 显示的一样。但是如果从 EMF9 列印直尺，您会看到更正常的直尺——6 英寸长、1 英寸宽。

缩放比例和纵横比

您也可能想要使用 EMF8 建立的直尺 metafile，而不必显示 6 英寸的图像。保持图像正确的 6 比 1 的纵横比是重要的。如前所述，在文书处理程式或别的应用程式中使用围绕方框来改变 metafile 的大小是很方便的，但是这样会导致某种程度的失真。在这种应用程式中，应该给使用者一个选项来保持原先的纵横比，而不用管围绕方框的大小如何变化。这就是说，传递给 PlayEnhMetaFile 的矩形结构不能直接由使用者选择的围绕方框定义。传递给该函式的矩形结构只是围绕方框的一部分。

让我们看一看程式 18-12 EMF10 是如何做的。

程式 18-12 EMF10

```
EMF10.C
/*------------------------------------------------------------------------
      EMF10.C --  Enhanced Metafile Demo #10
                                          (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF10") ;
TCHAR szTitle          [] = TEXT ("EMF10: Enhanced Metafile Demo #10") ;
void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
      ENHMETAHEADER          emh ;
      float                                fScale ;
      HENHMETAFILE           hemf ;
      int                                  cxMms,   cyMms,   cxPix,   cyPix,
cxImage, cyImage ;
      RECT                                 rect ;

      cxMms            = GetDeviceCaps (hdc, HORZSIZE) ;
```

```
     cyMms            = GetDeviceCaps (hdc, VERTSIZE) ;
     cxPix            = GetDeviceCaps (hdc, HORZRES) ;
     cyPix            = GetDeviceCaps (hdc, VERTRES) ;

     hemf  = GetEnhMetaFile (TEXT ("..\\emf8\\emf8.emf")) ;

     GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

     cxImage     = emh.rclFrame.right - emh.rclFrame.left ;
     cyImage     = emh.rclFrame.bottom - emh.rclFrame.top ;

     cxImage     = cxImage * cxPix / cxMms / 100 ;
     cyImage     = cyImage * cyPix / cyMms / 100 ;

     fScale      = min ((float) cxArea / cxImage, (float) cyArea / cyImage) ;

     cxImage               = (int) (fScale * cxImage) ;
     cyImage               = (int) (fScale * cyImage) ;

     rect.left             = (cxArea - cxImage) / 2 ;
     rect.right       = (cxArea + cxImage) / 2 ;
     rect.top              = (cyArea - cyImage) / 2 ;
     rect.bottom = (cyArea + cyImage) / 2 ;
     PlayEnhMetaFile (hdc, hemf, &rect) ;
     DeleteEnhMetaFile (hemf) ;
}
```

EMF10 伸展直尺图像以适应显示区域（或列印页面的可列印部分），但不会失真。通常直尺会伸展到显示区域的整个宽度，但是会上下居中对齐。如果您把视窗拉得太小，则直尺会与显示区域一般高，但是会水平居中对齐。

可能有许多种方法来计算合适的显示矩形，但是我们只根据 EMF9 的方式完成该项工作。EMF10.C 中的 PaintRoutine 函式开始部分与 EMF9.C 相同，为目的装置内容计算 6 英寸长的直尺图像适当的图素大小。

然後，程式计算名为 fScale 的浮点值，它是显示区域宽度与图像宽度的比值以及显示区域高度与图像高度比值两者的最小值。这个因数在计算围绕矩形前用於增加图像的图素大小。

metafile 中的映射方式

前面绘制的直尺单位有英寸，也有毫米。这种工作使用 GDI 提供的各种映射方式似乎非常适合。但是我坚持使用图素，并「手工」完成所有必要的计算。为什么呢？

答案很简单，就是将映射方式与 metafile 一起使用会十分混乱。我们不妨

实验一下。

当使用 metafile 装置内容呼叫 SetMapMode 时，该函式在 metafile 中像其他 GDI 函式一样被编码。如程式 18-13 EMF11 显示的那样。

程式 18-13 EMF11

```
EMF11.C
/*-------------------------------------------------------------------------
-
      EMF11.C -- Enhanced Metafile Demo #11
                                                (c) Charles Petzold, 1998
  ------------------------------------------------------------------------*/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF11") ;
TCHAR szTitle          [] = TEXT ("EMF11: Enhanced Metafile Demo #11") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
      int                    i, iHeight ;
      LOGFONT          lf ;
      TCHAR                  ch ;

                       // Black pen with 1-point width

      SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
                       // Rectangle surrounding entire pen (with adjustment)
      if (GetVersion () & 0x80000000)    `                // Windows 98
                 Rectangle (hdc, 0, -2, cx + 2, cy) ;
      else
// Windows NT
                       Rectangle (hdc, 0, -1, cx + 1, cy) ;

                       // Tick marks

      for (i = 1 ; i < 96 ; i++)
      {
        if  (i %16== 0) iHeight = cy /   2     ;     // inches
        else if(i % 8  == 0)iHeight = cy /    3 ;   // half inches
        else if(i % 4  == 0)iHeight = cy /    5 ;   // quarter inches
        else if(i % 2  == 0)iHeight = cy /    8 ;   // eighths
        else iHeight = cy /12;     // sixteenths

            MoveToEx (hdc, i * cx / 96, 0, NULL) ;
                 LineTo   (hdc, i * cx / 96, iHeight) ;
      }
                 // Create logical font
      FillMemory (&lf, sizeof (lf), 0) ;
      lf.lfHeight = cy / 2 ;
```

```
      lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

      SelectObject      (hdc, CreateFontIndirect (&lf)) ;
      SetTextAlign      (hdc, TA_BOTTOM | TA_CENTER) ;
      SetBkMode               (hdc, TRANSPARENT) ;

                  // Display numbers

      for (i = 1 ; i <= 5 ; i++)
      {
                  ch = (TCHAR) (i + '0') ;
                  TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
      }
                  // Clean up
      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
         DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
      HDC         hdcEMF ;
      HENHMETAFILE      hemf ;

      hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf11.emf"), NULL,
                              TEXT ("EMF11\0EMF Demo #11\0")) ;
      SetMapMode (hdcEMF, MM_LOENGLISH) ;
      DrawRuler (hdcEMF, 600, 100) ;
      hemf = CloseEnhMetaFile (hdcEMF) ;
      DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
      ENHMETAHEADER           emh ;
      HENHMETAFILE            hemf ;
      int                     cxMms, cyMms, cxPix, cyPix, cxImage, cyImage ;
      RECT                    rect ;

      cxMms             = GetDeviceCaps (hdc, HORZSIZE) ;
      cyMms             = GetDeviceCaps (hdc, VERTSIZE) ;
      cxPix             = GetDeviceCaps (hdc, HORZRES) ;
      cyPix             = GetDeviceCaps (hdc, VERTRES) ;

      hemf = GetEnhMetaFile (TEXT ("emf11.emf")) ;
      GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
      cxImage     = emh.rclFrame.right - emh.rclFrame.left ;
      cyImage     = emh.rclFrame.bottom - emh.rclFrame.top ;
```

```
     cxImage     = cxImage * cxPix / cxMms / 100 ;
     cyImage     = cyImage * cyPix / cyMms / 100 ;

     rect.left               = (cxArea - cxImage) / 2 ;
     rect.top          = (cyArea - cyImage) / 2 ;
     rect.right              = (cxArea + cxImage) / 2 ;
     rect.bottom             = (cyArea + cyImage) / 2 ;

     PlayEnhMetaFile (hdc, hemf, &rect) ;
     DeleteEnhMetaFile (hemf) ;
}
```

EMF11 中的 CreateRoutine 函式比 EMF8（最初的直尺 metafile 程式）中的那个简单，因为它不需要呼叫 GetDeviceCaps 来确定以每英寸点数为单位的显示器解析度。相反，EMF11 呼叫 SetMapMode 将映射方式设定为 MM_LOENGLISH，其逻辑单位等於 0.01 英寸。因而，直尺的大小为 600 100 个单位，并将这些数值传递给 DrawRuler。

除了 MoveToEx 和 LineTo 呼叫绘制直尺的刻度外，EMF11 中的 DrawRuler 函式与 EMF9 中的一样。当以图素单位绘制时（内定的 MM_TEXT 映射方式），垂直轴上的单位沿著萤幕向下增长。对於 MM_LOENGLISH 映射方式（以及其他度量映射方式），则向上增长。这就需要修改程式码。同时，也需要更改 Rectangle 函式中的调节因数。

EMF11 中的 PaintRoutine 函式基本上与 EMF9 中的相同，那个版本的程式能在显示器和印表机上以正确尺寸显示直尺。唯一不同之处在於 EMF11 使用 EMF11.EMF 档案，而 EMF9 使用 EMF8 建立的 EMF8.EMF 档案。

EMF11 显示的图像基本上与 EMF9 所显示的相同。因此，在这里可以看到将 SetMapMode 呼叫嵌入 metafile 能够简化 metafile 的建立，而且不影响以其正确大小显示 metafile 的机制。

映射与显示

在 EMF11 中计算目的矩形包括对 GetDeviceCaps 的几个呼叫。我们的第二个目的是使用映射方式代替这些呼叫。GDI 将目的矩形的座标视为逻辑座标。为这些座标使用 MM_HIMETRIC 似乎是个好方案，因为它使用 0.01 毫米作为逻辑单位，与增强型 metafile 表头中用於围绕矩形的单位相同。

程式 18-14 中所示的 EMF12 程式，保留了 EMF8 中使用的 DrawRuler 处理方式，但是使用 MM_HIMETRIC 映射方式显示 metafile。

程式 18-14　EMF12

```
EMF12.C
/*------------------------------------------------------------------------
```

```
-
      EMF12.C -- Enhanced Metafile Demo #12
                                          (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF12") ;
TCHAR szTitle          [] = TEXT ("EMF12: Enhanced Metafile Demo #12") ;
void DrawRuler (HDC hdc, int cx, int cy)
{
      int                    iAdj, i, iHeight ;
      LOGFONT          lf ;
      TCHAR                  ch ;

      iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
                        // Black pen with 1-point width
      SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
                        // Rectangle surrounding entire pen (with adjustment)
      Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
                        // Tick marks
      for (i = 1 ; i < 96 ; i++)
      {
        if (i % 16 == 0)     iHeight = cy /  2 ;          // inches
        else if (i %  8 == 0)      iHeight = cy /  3 ;    // half inches
        else if (i %  4 == 0)      iHeight = cy /  5 ;    // quarter inches
        else if (i %  2 == 0)      iHeight = cy /  8 ;    // eighths
        else iHeight = cy / 12 ;         // sixteenths

                  MoveToEx (hdc, i * cx / 96, cy, NULL) ;
                  LineTo   (hdc, i * cx / 96, cy - iHeight) ;
      }
                        // Create logical font
      FillMemory (&lf, sizeof (lf), 0) ;
      lf.lfHeight = cy / 2 ;
      lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

      SelectObject (hdc, CreateFontIndirect (&lf)) ;
      SetTextAlign (hdc, TA_BOTTOM | TA_CENTER) ;
      SetBkMode    (hdc, TRANSPARENT) ;

                        // Display numbers

      for (i = 1 ; i <= 5 ; i++)
    {
                  ch = (TCHAR) (i + '0') ;
                  TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
      }
```

```
                // Clean up
     DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
     DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
     HDC                         hdcEMF ;
     HENHMETAFILE     hemf ;
     int                         cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

     hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf12.emf"), NULL,
                          TEXT ("EMF13\0EMF Demo #12\0")) ;

     cxMms = GetDeviceCaps (hdcEMF, HORZSIZE) ;
     cyMms = GetDeviceCaps (hdcEMF, VERTSIZE) ;
     cxPix = GetDeviceCaps (hdcEMF, HORZRES) ;
     cyPix = GetDeviceCaps (hdcEMF, VERTRES) ;

     xDpi = cxPix * 254 / cxMms / 10 ;
     yDpi = cyPix * 254 / cyMms / 10 ;

     DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
     hemf = CloseEnhMetaFile (hdcEMF) ;
     DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
     ENHMETAHEADER          emh ;
     HENHMETAFILE           hemf ;
     POINT                                  pt ;
     int                                    cxImage, cyImage ;
     RECT                                   rect ;

     SetMapMode (hdc, MM_HIMETRIC) ;
     SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
     pt.x = cxArea ;
     pt.y = 0 ;

     DPtoLP (hdc, &pt, 1) ;
     hemf = GetEnhMetaFile (TEXT ("emf12.emf")) ;
     GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
     cxImage     = emh.rclFrame.right - emh.rclFrame.left ;
     cyImage     = emh.rclFrame.bottom - emh.rclFrame.top ;

     rect.left              = (pt.x - cxImage) / 2 ;
     rect.top          = (pt.y + cyImage) / 2 ;
```

```
      rect.right             = (pt.x + cxImage) / 2 ;
      rect.bottom            = (pt.y - cyImage) / 2 ;

      PlayEnhMetaFile (hdc, hemf, &rect) ;
      DeleteEnhMetaFile (hemf) ;
}
```

EMF12 中的 PaintRoutine 函式首先将映射方式设定为 MM_HIMETRIC。像其他度量映射方式一样，y 值沿著萤幕向上增长。然而，原点座标仍在萤幕的左上角，这就意味显示区域内的 y 座标值是负数。为了纠正这个问题，程式呼叫 SetViewportOrgEx 将原点座标设定在左下角。

装置座标(cxArea,0)位於萤幕的右上角。把该座标点传递给 DPtoLP（「装置座标点到逻辑座标点」）函式，得到以 0.01 毫米为单位的显示区域大小。

然後，程式载入 metafile，取得档案表头，并找到以 0.01 毫米为单位的 metafile 大小。这样计算目的矩形在显示区域居中对齐的位置就变得十分简单。

现在我们看到了在建立 metafile 时能够使用映射方式，显示它时也能使用映射方式。我们能一起完成它们吗？

如程式 18-15 EMF13 展示的那样，这是可以的。

程式 18-15 EMF13

```
EMF13.C
/*-----------------------------------------------------------------------------
-
      EMF13.C -- Enhanced Metafile Demo #13
                                          (c) Charles Petzold, 1998
-----------------------------------------------------------------------------
*/

#include <windows.h>
TCHAR szClass          [] = TEXT ("EMF13") ;
TCHAR szTitle          [] = TEXT ("EMF13: Enhanced Metafile Demo #13") ;

void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
      ENHMETAHEADER          emh ;
      HENHMETAFILE           hemf ;
      POINT                                  pt ;
      int                                    cxImage, cyImage ;
      RECT                                   rect ;

      SetMapMode (hdc, MM_HIMETRIC) ;
```

```
     SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
     pt.x  = cxArea ;
     pt.y  = 0 ;

     DPtoLP (hdc, &pt, 1) ;

     hemf = GetEnhMetaFile (TEXT ("..\\emf11\\emf11.emf")) ;

     GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

     cxImage     = emh.rclFrame.right - emh.rclFrame.left ;
     cyImage     = emh.rclFrame.bottom - emh.rclFrame.top ;

     rect.left               = (pt.x - cxImage) / 2 ;
     rect.top          = (pt.y + cyImage) / 2 ;
     rect.right              = (pt.x + cxImage) / 2 ;
     rect.bottom             = (pt.y - cyImage) / 2 ;

     PlayEnhMetaFile (hdc, hemf, &rect) ;
     DeleteEnhMetaFile (hemf) ;
}
```

在 EMF13 中，由於直尺 metafile 已由 EMF11 建立，所以它没有使用映射方式建立 metafile。EMF13 只是简单地载入 metafile，然後像 EMF11 一样使用映射方式计算目的矩形。

现在，我们可以建立一些规则。在建立 metafile 时，GDI 使用对映射方式的任意嵌入修改，来计算以图素和毫米为单位的 metafile 图像的大小。图像的大小储存在 metafile 表头内。在显示 metafile 时,GDI 在呼叫 PlayEnhMetaFile 时根据有效的映射方式建立目的矩形的实际位置，而本来的 metafile 中并没有任何记录去更改这个位置。