第十二章　剪贴簿

Microsoft Windows 剪贴簿允许把资料从一个程式传送到另一个程式中。它的原理相对而言比较简单，把资料存放到剪贴簿上的程式或从剪贴簿上取出资料的程式都无须太多的负担。Windows 98 和 Microsoft Windows NT 都提供了剪贴簿浏览程式，该程式可以显示剪贴簿的目前内容。

许多处理档案或者其他资料的程式都包含一个「Edit」功能表，其中包括「Cut」、「Copy」和「Paste」选项。当使用者选择「Cut」或者「Copy」时，程式将资料传送给剪贴簿。这个资料使用某种格式，如文字、点阵图（一种按位元排列的矩形阵列，其中的位元与平面显示的图素相对应）或者 metafile（用二进位元数值内容表示的绘图命令集）等。当使用者从功能表中选择「Paste」时，程式检查剪贴簿中包含的资料，看看使用的是否是程式可以接受的一种格式。如果是，那么资料将从剪贴簿传送到程式中。

如果使用者不发出明确的指令，程式就不能把资料送入或移出剪贴簿。例如，在某个程式中执行剪下或复制（或者按 Ctrl-X 及 Ctrl-C）操作的使用者，应该能够假定资料将储存在剪贴簿上，直到下次剪下或复制操作为止。

回忆一下第十和第十一章所示的 POPPAD 程式的修订版中，我们加上了「Edit」功能表，但是在那边这功能表的作用只是发送讯息给编辑控制项而已。多数情况下，处理剪贴簿并不方便，您必须自己呼叫剪贴簿传输函式。

本章集中讨论将文字传入和移出剪贴簿。在後面的章节里，我将向您展示如何用剪贴簿处理点阵图（第十四、十五和十六章）和 metafile（第十八章）。

剪贴簿的简单使用

我们由分析把资料传送到剪贴簿（剪下或复制）和存取剪贴簿资料（粘贴）的程式码开始。

标准剪贴簿资料格式

Windows 支援不同的预先定义剪贴簿格式，这些格式在 WINUSER.H 定义成以 CF 为字首的识别字。

首先介绍三种能够储存在剪贴簿上的文字资料型态，以及一个与剪贴簿格式相关的资料型态：

CF_TEXT 以 NULL 结尾的 ANSI 字元集字串。它在每行末尾包含一个 carriage return 和 linefeed 字元，这是最简单的剪贴簿资料格式。传送到剪

贴簿的资料存放在整体记忆体块中，并且是利用记忆体块代号进行传送的（我将简短地讨论此项概念）。这个记忆体块专供剪贴簿使用，建立它的程式不应该继续使用它。

CF_OEMTEXT 含有文字资料（与 CF_TEXT 类似）的记忆体块。但是它使用的是 OEM 字元集。通常 Windows 程式不必关心这一点；它只有与在视窗中执行 MS-DOS 程式一起使用剪贴簿时才会使用。

CF_UNICODETEXT 含有 Unicode 文字的记忆体块。与 CF_TEXT 类似，它在每一行的末尾包含一个 carriage return 和 linefeed 字元，以及一个 NULL 字元（两个 0 位元组）以表示资料结束。CF_UNICODETEXT 只支援 Windows NT。

CF_LOCALE 一个国家地区识别字的代号。表示剪贴簿文字使用的国别地区设定。

下面是两种附加的剪贴簿格式，它们在概念上与 CF_TEXT 格式相似（也就是说，它们都是文字资料），但是它们不需要以 NULL 结尾，因为格式已经定义了资料的结尾。现在已经很少使用这些格式了：

CF_SYLK 包含 Microsoft 「符号连结」资料格式的整体记忆体块。这种格式用在 Microsoft 的 Multiplan、Chart 和 Excel 程式之间交换资料，它是一种 ASCII 码格式，每一行都用 carriage return 和 linefeed 结尾。

CF_DIF 包含资料交换格式(DIF)之资料的整体记忆体块。这种格式是由 Software Arts 公司提出的，用於把资料送到 VisiCalc 试算表程式中。这也是一种 ASCII 码格式，每一行都使用 carriage return 和 linefeed 结尾。

下面三种剪贴簿格式与点阵图有关。所谓点阵图就是资料位元的矩形阵列，其中的资料位元与输出设备的图素相对应。第十四和第十五章将详细讨论点阵图以及这些点阵图剪贴簿的格式：

CF_BITMAP 与装置相关的点阵图格式。点阵图是通过点阵图代号传送给剪贴簿的。同样，在把这个点阵图传送给剪贴簿之後，程式不应该再继续使用这个点阵图。

CF_DIB 定义一个装置无关点阵图（在第十五章中描述）的记忆体块。这种记忆体块是以点阵图资讯结构开始的，後面跟著可用的颜色表和点阵图资料位元。

CF_PALETTE 调色盘代号。它通常与 CF_DIB 配合使用，以定义与装置相关的点阵图所使用的颜色调色盘。

在剪贴簿中，还有可能以工业标准的 TIFF 格式储存的点阵图资料：

CF_TIFF 含有标号图像档案格式(TIFF)资料的整体记忆体块。这种格式由 Microsoft、Aldus 公司和 Hewlett-Packard 公司以及一些硬体厂商推荐使用。

这一格式可从 Hewlett-Packard 的网站上获得。

下面是两个 metafile 格式，我将在第十八章详细讨论。一个 metafile 就是一个以二进位格式储存的画图命令集：

CF_METAFILEPICT 以旧的 metafile 格式存放的「图片」。

CF_ENHMETAFILE 增强型 metafile（32 位元 Windows 支援的）代号。

最後介绍几个混合型的剪贴簿格式：

CF_PENDATA 与 Windows 的笔式输入扩充功能联合使用。

CF_WAVE 声音（波形）档案。

CF_RIFF 使用资源交换档案格式（Resource Interchange File Format）的多媒体资料。

CF_HDROP 与拖放服务相关的档案列表。

记忆体配置

程式向剪贴簿传输一些资料的时候，必须配置一个记忆体块，并且将这块记忆体交给剪贴簿处理。在本书早期的程式中需要配置记忆体时，我们只需使用标准 C 执行时期程式库所支援的 malloc 函式。但是，由於在 Windows 中执行的应用程式之间必须要共用剪贴簿所储存的记忆体块，这时 malloc 函式就有些不适任这项任务了。

实际上，我们必须把早期 Windows 所开发的记忆体配置函式再拿出来使用，那时的作业系统在 16 位元的实际模式记忆体结构中执行。现在的 Windows 仍然支援这些函式，您还可以使用它们，但不是必须使用这些函式就是了。

要用 Windows API 来配置一个记忆体块，可以呼叫：

```
hGlobal = GlobalAlloc (uiFlags, dwSize) ;
```

此函式有两个参数：一系列可能的旗标和记忆体块的位元组大小。函式传回一个 HGLOBAL 型态的代号，称为「整体记忆体块代号」或「整体代号」。传回值为 NULL 表示不能配置足够的记忆体。

虽然 GlobalAlloc 的两个参数略有不同，但它们都是 32 位元的无正负号整数。如果将第一个参数设定为 0，那么您就可以更有效地使用旗标 GMEM_FIXED。在这种情况下，GlobalAlloc 传回的整体代号实际是指向所配置记忆体块的指标。

如果不喜欢将记忆体块中的每一位元都初始化为 0，那么您也能够使用旗标 GMEM,_ZEROINIT。在 Windows 表头档案中，简洁的 GPTR 旗标定义为 GMEM_FIXED 和 GMEM_ZEROINIT 旗标的组合：

```
#define GPTR (GMEM_FIXED | GMEM_ZEROINIT)
```

下面是一个重新配置函式：

```
hGlobal = GlobalReAlloc (hGlobal, dwSize, uiFlags) ;
```

如果记忆体块扩大了，您可以用 GMEM_ZEROINIT 旗标将新的位元组设为 0。

下面是获得记忆体块大小的函式：

```
dwSize = GlobalSize (hGlobal) ;
```

释放记忆体块的函式：

```
GlobalFree (hGlobal) ;
```

在早期 16 位元的 Windows 中，因为 Windows 不能在实体记忆体中移动记忆体块，所以禁止使用 GMEM_FIXED 旗标。在 32 位元的 Windows 中，GMEM_FIXED 旗标很常见。这是因为它将传回一个虚拟位址，并且作业系统也能够通过改变记忆体页映射表在实体记忆体中移动记忆体块。因此为 16 位元的 Windows 写程式时，GlobalAlloc 推荐使用 GMEM_MOVEABLE 旗标。在 Windows 的表头档案中还定义了一个简写识别字，用此识别字可以在可移动的记忆体之外填 0：

```
#define GHND (GMEM_MOVEABLE | GMEM_ZEROINIT)
```

GMEM_MOVEABLE 旗标允许 Windows 在虚拟记忆体中移动一个记忆体块。这不是说将在实体记忆体中移动记忆体块，只是应用程式用於读写这块记忆体的位址可以被变动。

尽管 GMEM_MOVEABLE 是 16 位元 Windows 的通则，但是它的作用现在已经少得多了。如果您的应用程式频繁地配置、重新配置以及释放不同大小的记忆体块，应用程式的虚拟位址空间将会变得支离破碎。可以想像得到，最後虚拟记忆体位址空间就会被用完。如果这是个可能会发生的问题，那么您将希望记忆体是可移动的。下面就介绍如何让记忆体块成为可搬移位置的。

首先定义一个指标（例如，一个 int 型态的）和一个 GLOBALHANDLE 型态的变数：

```
int * p ;
            GLOBALHANDLE hGlobal ;
```

然後配置记忆体。例如：

```
hGlobal = GlobalAlloc (GHND, 1024) ;
```

与处理其他 Windows 代号一样，您不必担心数字的实际意义，只要照著作就好了。需要存取记忆体块时，可以呼叫：

```
p = (int *) GlobalLock (hGlobal) ;
```

此函式将代号转换为指标。在记忆体块被锁定期间，Windows 将固定虚拟记忆体中的位址，不再移动那块记忆体。存取结束後呼叫：

```
GlobalUnlock (hGlobal) ;
```

这将使 Windows 可以在虚拟记忆体中移动记忆体块。要真正确保此程序正常运作（体验早期 Windows 程式写作者的痛苦经历），您应该在单一个讯息处理期间锁定和解锁记忆体块。

在释放记忆体时，呼叫 GlobalFree 应使用代号而不是指标。如果您现在不

能存取代号，可以使用下面的函式：

```
hGlobal = GlobalHandle (p) ;
```

在解锁之前，您能够多次锁定一个记忆体块。Windows 保留一个锁定次数，而且在记忆体块可被自由移动之前，每次锁定都需要相对应的解锁。当 Windows 在虚拟记忆体中移动一个记忆体块时，不需要将位元组从一个位置复制到另一个，只需巧妙地处理记忆体页映射表。通常，让 32 位元 Windows 为您的程式配置可移动的记忆体块，其唯一确实的理由只是避免虚拟记忆体的空间碎裂出现。使用剪贴簿时，也应该使用可移动记忆体。

为剪贴簿配置记忆体时，您应该以 GMEM_MOVEABLE 和 GMEM_SHARE 旗标呼叫 GlobalAlloc 函式。GMEM_SHARE 旗标使得其他应用程式也可以使用那块记忆体。

将文字传送到剪贴簿

让我们想像把一个 ANSI 字串传送到剪贴簿上，并且我们已经有了指向这个字串的指标(pString)。现在希望传送这个字串的 iLength 字元，这些字元可能以 NULL 结尾，也可能不以 NULL 结尾。

首先，通过使用 GlobalAlloc 来配置一个足以储存字串的记忆体块，其中还包括一个终止字元 NULL：

```
hGlobal = GlobalAlloc (GHND | GMEM_SHARE, iLength + 1) ;
```

如果未能配置到记忆体块，hGlobal 的值将为 NULL 。如果配置成功，则锁定这块记忆体，并得到指向它的一个指标：

```
pGlobal = GlobalLock (hGlobal) ;
```

将字串复制到记忆体块中：

```
for (i = 0 ; i < wLength ; i++)
    *pGlobal++ = *pString++ ;
```

由於 GlobalAlloc 的 GHND 旗标已使整个记忆体块在配置期间被清除为零，所以不需要增加结尾的 NULL 。以下叙述为记忆体块解锁：

```
GlobalUnlock (hGlobal) ;
```

现在就有了表示以 NULL 结尾的文字所在记忆体块的记忆体代号。为了把它送到剪贴簿中，打开剪贴簿并把它清空：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
```

利用 CF_TEXT 识别字把记忆体代号交给剪贴簿，关闭剪贴簿：

```
SetClipboardData (CF_TEXT, hGlobal) ;
CloseClipboard () ;
```

工作告一段落。

下面是关於此过程的一些规则：

在处理同一个讯息的过程中呼叫 OpenClipboard 和 CloseClipboard。不需

要时，不要打开剪贴簿。

不要把锁定的记忆体代号交给剪贴簿。

当呼叫 SetClipboardData 後，请不要再继续使用该记忆体块。它不再属於使用者程式，必须把代号看成是无效的。如果需要继续存取资料，可以制作资料的副本，或从剪贴簿中读取它（如下节所述）。您也可以在 SetClipboardData 呼叫和 CloseClipboard 呼叫之间继续使用记忆体块，但是不要使用传递给 SetClipboardData 函式的整体代号。事实上，此函式也传回一个整体代号，必需锁定这些代码以存取记忆体。在呼叫 CloseClipboard 之前，应先为此代号解锁。

从剪贴簿上取得文字

从剪贴簿上取得文字只比把文字传送到剪贴簿上稍微复杂一些。您必须首先确定剪贴簿是否含有 CF_TEXT 格式的资料，最简单的方法是呼叫

```
bAvailable = IsClipboardFormatAvailable (CF_TEXT) ;
```

如果剪贴簿上含有 CF_TEXT 资料，这个函式将传回 TRUE（非零）。我们在第十章的 POPPAD2 程式中已使用了这个函式，用它来确定「Edit」功能表中「Paste」项是被启用还是被停用的。IsClipboardFormatAvailable 是少数几个不需先打开剪贴簿就可以使用的剪贴簿函式之一。但是，如果您之後想再打开剪贴簿以取得这个文字，就应该再做一次检查（使用同样的函式或其他方法），以便确定 CF_TEXT 资料是否仍然留在剪贴簿中。

为了传送出文字，首先打开剪贴簿：

```
OpenClipboard (hwnd) ;
```

会得到代表文字的记忆体块代号：

```
hGlobal = GetClipboardData (CF_TEXT) ;
```

如果剪贴簿不包含 CF_TEXT 格式的资料，此代号就为 NULL。这是确定剪贴簿是否含有文字的另一种方法。如果 GetClipboardData 传回 NULL，则关闭剪贴簿，不做其他任何工作。

从 GetClipboardData 得到的代号并不属於使用者程式——它属於剪贴簿。仅在 GetClipboardData 和 CloseClipboard 呼叫之间这个代号才有效。您不能释放这个代号或更改它所引用的资料。如果需要继续存取这些资料，必须制作这个记忆体块的副本。

这里有一种将资料复制到使用者程式中的方法。首先，配置一块与剪贴簿资料块大小相同的记忆体块，并配置一个指向该块的指标：

```
pText = (char *) malloc (GlobalSize (hGlobal)) ;
```

再次呼叫 hGlobal ，而 hGlobal 是从 GetClipboardData 呼叫传回的整体代

号。现在锁定代号，获得一个指向剪贴簿块的指标：

```
pGlobal = GlobalLock (hGlobal) ;
```

现在就可以复制资料了：

```
strcpy (pText, pGlobal) ;
```

或者，您可以使用一些简单的 C 程式码：

```
while (*pText++ = *pGlobal++) ;
```

在关闭剪贴簿之前先解锁记忆体块：

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

现在您有了一个叫做 pText 的指标，以後程式的使用者就可以用它来复制文字了。

打开和关闭剪贴簿

在任何时候，只有一个程式可以打开剪贴簿。呼叫 OpenClipboard 的作用是当一个程式使用剪贴簿时，防止剪贴簿的内容发生变化。OpenClipboard 传回 BOOL 值，它说明是否已经成功地打开了剪贴簿。如果另一个应用程式没有关闭剪贴簿，那么它就不能被打开。如果每个程式在回应使用者的命令时都尽快地、遵守规范地打开然後关闭剪贴簿，那么您将永远不会遇到不能打开剪贴簿的问题。

但是，在不遵守规范程式和优先权式多工环境中，总会发生一些问题。即使在您的程式将某些东西放入剪贴簿和使用者启动一个「Paste」选项期间，您的程式并没有失去输入焦点，但是您也不能假定您放入的东西仍然在那里，一个背景程式有可能已经在这段期间存取过剪贴簿了。

而且，请留意一个与讯息方块有关的更微妙问题：如果不能配置足够的记忆体来将内容复制到剪贴簿，那么您可能希望显示一个讯息方块。但是，如果这个讯息方块不是系统模态的，那么使用者可以在显示讯息方块期间切换到另一个应用程式中。您应该使用系统模态的讯息方块，或者在您显示讯息方块之前关闭剪贴簿。

如果您在显示一个对话方块时将剪贴簿保持为打开状态，那么您还可能遇到其他问题，对话方块中的编辑栏位会使用剪贴簿进行文字的剪贴。

剪贴簿和 Unicode

迄今为止,我只讨论了用剪贴簿处理 ANSI 文字(每个字元对应一个位元组)。我们用 CF_TEXT 识别字时就是这种格式。您可能对 CF_OEMTEXT 和 CF_UNICODETEXT 还不熟悉吧。

我有一些好消息：在处理您所想要的文字格式时，您只需呼叫 SetClipboardData 和 GetClipboardData，Windows 将处理剪贴簿中所有的文字转换。例如，在 Windows NT 中，如果一个程式用 SetClipboardData 来处理 CF_TEXT 剪贴簿资料型态，程式也能用 CF_OEMTEXT 呼叫 GetClipboardData。同样地，剪贴簿也能将 CF_OEMTEXT 资料转换为 CF_TEXT。

在 Windows NT 中，转换发生在 CF_UNICODETEXT、CF_TEXT 和 CF_OEMTEXT 之间。程式应该使用对程式本身而言最方便的一种文字格式来呼叫 SetClipboardData 。同样地，程式应该用程式需要的文字格式来呼叫 GetClipboardData。我们已经知道，本书附上的程式在编写时可以带有或不带 UNICODE 识别字。如果您的程式也依此编写，那么在定义了 UNICODE 识别字之後，程式将执行带有 CF_UNICODETEXT 参数的 SetClipboardData 以及 GetClipboardData 呼叫，而不是 CF_TEXT。

CLIPTEXT 程式，如程式 12-1 所示，展示了一种可行的方法。

程式 12-1 CLIPTEXT

```
CLIPTEXT.C
/*------------------------------------------------------------------------
      CLIPTEXT.C --          The Clipboard and Text
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

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

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
#ifdef UNICODE
#define CF_TCHAR CF_UNICODETEXT
TCHAR szDefaultText[]        = TEXT ("Default Text - Unicode Version") ;
TCHAR szCaption[]            =  TEXT  ("Clipboard  Text  Transfers  -  Unicode
Version") ;
#else
#define CF_TCHAR CF_TEXT
TCHAR szDefaultText[] = TEXT ("Default Text - ANSI Version") ;
TCHAR szCaption[]            = TEXT ("Clipboard Text Transfers - ANSI Version") ;

#endif
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("ClipText") ;
      HACCEL                       hAccel ;
      HWND                         hwnd ;
      MSG                    msg ;
      WNDCLASS                     wndclass ;
```

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

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

     hwnd = CreateWindow (szAppName, szCaption,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInstance, NULL) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;
     hAccel = LoadAccelerators (hInstance, szAppName) ;

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

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
     static PTSTR          pText ;
     BOOL                  bEnable ;
     HGLOBAL               hGlobal ;
```

```
     HDC              hdc ;
     PTSTR                  pGlobal ;
     PAINTSTRUCT            ps ;
     RECT                   rect ;

     switch (message)
     {
     case  WM_CREATE:
                SendMessage (hwnd, WM_COMMAND, IDM_EDIT_RESET, 0) ;
                return 0 ;

     case  WM_INITMENUPOPUP:
                EnableMenuItem         ((HMENU)    wParam,
     IDM_EDIT_PASTE,
                       IsClipboardFormatAvailable   (CF_TCHAR)              ?
     MF_ENABLED  : MF_GRAYED) ;

                bEnable = pText ? MF_ENABLED : MF_GRAYED ;

                EnableMenuItem         ((HMENU)    wParam,     IDM_EDIT_CUT,
           bEnable) ;
                EnableMenuItem         ((HMENU)    wParam,    IDM_EDIT_COPY,
           bEnable) ;
                EnableMenuItem         ((HMENU)    wParam,
     IDM_EDIT_CLEAR,  bEnable) ;
                break ;

     case  WM_COMMAND:
                switch (LOWORD (wParam))
                {
                case  IDM_EDIT_PASTE:
                                  OpenClipboard (hwnd) ;

                                  if    (hGlobal      =      GetClipboardData
(CF_TCHAR))
                                  {
                                        pGlobal = GlobalLock (hGlobal) ;
                                        if (pText)
                                        {
                                                   free (pText) ;
                                                   pText = NULL ;
                                        }
                                        pText = malloc (GlobalSize (hGlobal)) ;
                                        lstrcpy (pText, pGlobal) ;
                                        InvalidateRect (hwnd, NULL, TRUE) ;
                                  }
                                  CloseClipboard () ;
                                  return 0 ;
```

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

                                hGlobal = GlobalAlloc (GHND | GMEM_SHARE,
                      (lstrlen (pText) + 1) * sizeof (TCHAR)) ;
                                pGlobal = GlobalLock (hGlobal) ;
                                lstrcpy (pGlobal, pText) ;
                                GlobalUnlock (hGlobal) ;

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

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

                case  IDM_EDIT_RESET:
                                if (pText)
                                {
                                      free (pText) ;
                                      pText = NULL ;
                                }
                          pText = malloc ((lstrlen (szDefaultText) + 1) * sizeof
(TCHAR)) ;
                                lstrcpy (pText, szDefaultText) ;
                                InvalidateRect (hwnd, NULL, TRUE) ;
                                return 0 ;
                }
                break ;

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

                GetClientRect (hwnd, &rect) ;
```

```
                if (pText != NULL)
                DrawText    (hdc,   pText,   -1,   &rect,   DT_EXPANDTABS   |
DT_WORDBREAK) ;

                EndPaint (hwnd, &ps) ;
                return 0 ;

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

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

CLIPTEXT.RC （摘录）

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

////////////////////////////////////////////////////////////////////////////
/
// Menu
CLIPTEXT MENU DISCARDABLE
BEGIN
      POPUP "&Edit"
      BEGIN
            MENUITEM "Cu&t\tCtrl+X",                IDM_EDIT_CUT
       MENUITEM "&Copy\tCtrl+C",              IDM_EDIT_COPY
       MENUITEM "&Paste\tCtrl+V",             IDM_EDIT_PASTE
MENUITEM "De&lete\tDel",          IDM_EDIT_CLEAR
       MENUITEM SEPARATOR
       MENUITEM "&Reset",                   IDM_EDIT_RESET
      END
END

////////////////////////////////////////////////////////////////////////////
/
// Accelerator
CLIPTEXT ACCELERATORS DISCARDABLE
BEGIN
      "C",          IDM_EDIT_COPY,    VIRTKEY, CONTROL, NOINVERT
      "V",          IDM_EDIT_PASTE,   VIRTKEY, CONTROL, NOINVERT
      VK_DELETE,              IDM_EDIT_CLEAR,   VIRTKEY, NOINVERT
      "X",          IDM_EDIT_CUT,     VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H（摘录）

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

#define IDM_EDIT_CUT          40001
#define IDM_EDIT_COPY         40002
#define IDM_EDIT_PASTE        40003
#define IDM_EDIT_CLEAR        40004
#define IDM_EDIT_RESET        40005
```

这是在 Windows NT 下执行 Unicode 版和 ANSI 版程式的概念，而且可以看到，剪贴簿是如何在两种字元集之间转换的。注意 CLIPTEXT.C 顶部的#ifdef 叙述。如果定义了 UNICODE 识别字，那么 CF_TCHAR（我命名的一种常用的剪贴簿格式）就等於 CF_UNICODETEXT；否则，它就等於 CF_TEXT。程式後面呼叫的 IsClipboardFormatAvailable、GetClipboardData 和 SetClipboardData 函式都使用 CF_TCHAR 来指定资料型态。

在程式的开始部分（以及您从「Edit」功能表中选择「Reset」选项时），静态变数 pText 包含一个指标，在 Unicode 版的程式中，指标指向 Unicode 字串「Default Text -Unicode version」；在非 Unicode 版的程式中，指标指向「Default Text - ANSI version」。您可以用「Cut」或「Copy」命令将字串传递给剪贴簿，用「Cut」或「Delete」命令从程式中删除字串。「Paste」命令将剪贴簿中的文字内容复制到 pText。在 WM_PAINT 讯息处理期间，pText 将字串显示在程式的显示区域。

如果您先在 Unicode 版的 CLIPTEXT 中选择了「Copy」命令，然後在非 Unicode 版中选择「Paste」命令，那么您就能看到文字已经从 Unicode 转换成了 ANSI。类似地，如果您执行相反的操作，那么文字就会从 ANSI 转换成 Unicode。

复杂的剪贴簿用法

我们已经看到，在将资料准备好之後，从剪贴簿传输资料时需要四个呼叫：

```
OpenClipboard                (hwnd) ;
EmptyClipboard               () ;
SetClipboardData       (iFormat, hGlobal) ;
CloseClipboard               () ;
存取这些资料需要三个呼叫
OpenClipboard (hwnd) ;
hGlobal = GetClipboardData (iFormat) ;
其他行程式
CloseClipboard () ;
```

在 GetClipboardData 和 CloseClipboard 呼叫之间，可以复制剪贴簿资料或以其他方式来使用它。很多应用程式都需要采用这种方法，但也可以用更复杂的方式来使用剪贴簿。

利用多个资料项目

当打开剪贴簿并把资料传送给它时，必须先呼叫 EmptyClipboard，通知 Windows 释放或删除剪贴簿上的内容。不能在现有的剪贴簿内容中附加其他东西。所以，从这种意义上说，剪贴簿每次只能保留一个资料项目。

但是，可以在 EmptyClipboard 和 CloseClipboard 呼叫之间多次呼叫 SetClipboardData，每次都使用不同的剪贴簿格式。例如，如果想在剪贴簿中储存一个很短的文字字串，可以把这个文字写入 metafile，也可以把这个文字写入点阵图。把点阵图选进记忆体装置内容中，并把这个字串写进点阵图中。利用这种方法可以使字串不仅能为从剪贴簿上读取文字的程式所使用，也可以为从剪贴簿上读取点阵图和 metafile 的程式所使用。当然，这些程式并不能知道 metafile 或点阵图实际上包含了一个字串。

如果想把一些代号写到剪贴簿上，对每个代号均可以呼叫 SetClipboardData：

```
OpenClipboard          (hwnd) ;
EmptyClipboard         () ;
SetClipboardData (CF_TEXT, hGlobalText) ;
SetClipboardData (CF_BITMAP, hBitmap) ;
SetClipboardData (CF_METAFILEPICT, hGlobalMFP) ;
CloseClipboard         () ;
```

当这三种格式的资料同时位於剪贴簿上时，用 CF_TEXT、CF_BITMAP 或 CF_METAFILEPICT 参数呼叫 IsClipboardFormatAvailable 将传回 TRUE。通过下列呼叫程式可以存取这些代码：

```
hGlobalText = GetClipboardData (CF_TEXT) ;
```

或

```
hBitmap = GetClipboardData (CF_BITMAP) ;
```

或

```
hGlobalMFP = GetClipboardData (CF_METAFILEPICT) ;
```

下一次程式呼叫 EmptyClipboard 时，Windows 将释放或删除剪贴簿上保留的所有三个代号。

在将不同的文字格式、不同的点阵图格式或者不同的 metafile 格式添加到剪贴簿时，不要使用这种技术。只使用一种文字格式、一种点阵图格式以及一种 metafile 格式。就像我所说的那样，Windows 将在 CF_TEXT、CF_OEMTEXT 和 CF_UNICODETEXT 之间转换，也可以在 CF_BITMAP 和 CF_DIB 之间，以及在 CF_METAFILEPICT 和 CF_ENHMETAFILE 之间进行转换。

透过首先打开剪贴簿，然後呼叫 EnumClipboardFormats，程式可以确定剪贴簿储存的所有格式。开始时设定变数 iFormat 为 0：

```
iFormat = 0 ;
OpenClipboard (hwnd) ;
```

现在从 0 值开始逐次进行连续的 EnumClipboardFormats 呼叫。函式将为目前在剪贴簿中的每种格式传回一个正的 iFormat 值。当函式传回 0 时，表示完成：

```
while (iFormat = EnumClipboardFormats (iFormat))
{
    各个 iFormat 值的处理方式
}
CloseClipboard () ;
```

您可以通过下面的呼叫来取得目前在剪贴簿中之不同格式的个数：

```
iCount = CountClipboardFormats () ;
```

延迟提出

当把资料放入剪贴簿中时，一般来说要制作一份资料的副本，并将包含这份副本的记忆体块代号传给剪贴簿。对非常大的资料项目来说，这种方法会浪费记忆体空间。如果使用者不想把资料粘贴到另一个程式里，那么，在被其他内容取代之前，它将一直占据著记忆体空间。

通过使用一种叫做「延迟提出」的技术可以避免这个问题。实际上，直到另一个程式需要资料，程式才提供这份资料。为此，不将资料代号传给 Windows，而是在 SetClipboardData 呼叫中使用 NULL：

```
OpenClipboard          (hwnd) ;
EmptyClipboard         () ;
SetClipboardData (iFormat, NULL) ;
CloseClipboard         () ;
```

可以有多个使用不同 iFormat 值的 SetClipboardData 呼叫，对其中某些呼叫可使用 NULL 值。而对其他一些则使用实际的代号值。

前面的过程比较简单，以下的过程就要稍微复杂一些了。当另一个程式呼叫 GetClipboardData 时，Windows 将检查那种格式的代号是否为 NULL。如果是，Windows 将给「剪贴簿所有者」（您的程式）发送一个讯息，要求取得资料的实际代号，这时您的程式必须提供这个代号。

更具体地说，「剪贴簿所有者」是将资料放入剪贴簿的最後一个视窗。当一个程式呼叫 OpenClipboard 时，Windows 储存呼叫这个函式时所用的视窗代号，这个代号标示打开剪贴簿的视窗。一旦收到一个 EmptyClipboard 呼叫，Windows 就使这个视窗作为新的剪贴簿所有者。

使用延迟提出技术的程式在它的视窗讯息处理程式中必须处理三个讯息：WM_RENDERFORMAT、WM_RENDERALLFORMATS 和 WM_DESTROYCLIPBOARD。当另一个

程式呼叫 GetClipboardData 时，Windows 给视窗讯息处理程式发送一个 WM_RENDERFORMAT 讯息，wParam 的值是所要求的格式。在处理 WM_RENDERFORMAT 讯息时，不要打开或清空剪贴簿。为 wParam 所指定的格式建立一个整体记忆体块，把数据传给它，并用正确的格式和相应代号呼叫 SetClipboardData。很明显地，为了在处理 WM_RENDERFORMAT 时正确地构造出此资料，需要在程式中保留这些资讯。当另一个程式呼叫 EmptyClipboard 时，Windows 给您的程式发送一个 WM_DESTROYCLIPBOARD 讯息，告诉您不再需要构造剪贴簿资料的资讯。您的程式不再是剪贴簿的所有者。

如果程式在它自己仍然是剪贴簿所有者的时候就要终止执行，并且剪贴簿上仍然包含著该程式用 SetClipboardData 设定的 NULL 资料代号，它将收到 WM_RENDERALLFORMATS 讯息。这时，应该打开剪贴簿，清空它，把资料载入记忆体块中，并为每种格式呼叫 SetClipboardData，然後关闭剪贴簿。WM_RENDERALLFORMATS 讯息是视窗讯息处理程式最後收到的讯息之一。它後面跟有 WM_DESTROYCLIPBOARD 讯息（由於已经提出了所有资料），然後是正常的 WM_DESTROY 讯息。

如果您的程式只能向剪贴簿传输一种格式的资料（例如文字），那么您可以把 WM_RENDERALLFORMATS 和 WM_RENDERFORMAT 处理结合在一起。这些程式码应该类似下面这样：

```
case  WM_RENDERALLFORMATS :
      OpenClipboard (hwnd) ;
      EmptyClipboard () ;
                                                  // fall through
case  WM_RENDERFORMAT :
    // 将文字放入整体记忆体块
      SetClipboardData (CF_TEXT, hGlobal) ;
      if (message == WM_RENDERALLFORMATS)
                  CloseClipboard () ;
      return 0 ;
```

如果您的程式使用好几种剪贴簿格式，那么您可能想为 wParam 所要求的格式处理 WM_ RENDERFORMAT。除非程式在存放构造资料所需的资讯时遇到困难，否则不需要处理 WM_DESTROYCLIPBOARD 讯息。

自订资料格式

到目前为止，我们仅处理了 Windows 定义的标准剪贴簿资料格式。但是，您可能想用剪贴簿来储存「自订资料格式」。许多文书处理程式使用这种技术来储存包含著字体和格式化资讯的文字。

初看之下，这个概念似乎是没有意义的。如果剪贴簿的作用是在应用程式

之间传送资料，那么，为什么剪贴簿中要含有只有一个应用程式才能理解的资料呢？答案很简单：剪贴簿允许在同一个程式的内部（或者可能在一个程式中的不同执行实体之间）传送资料。很明显地，这些执行实体能理解它们自己的自订资料格式。

有几种使用自订资料格式的方法。最简单的方法用到一种表面上是标准剪贴簿格式（文字、点阵图或 metafile）的资料，可是该资料实际上只对您的程式有意义。这种情况下，在 SetClipboardData 和 GetClipboardData 呼叫中可使用下列 wFormat 值：CF_DSPTEXT、CF_DSPBITMAP、CF_DSPMETAFILEPICT 或 CF_DSPENHMETAFILE（字母 DSP 代表「显示器」）。这些格式允许 Windows 按文字、点阵图或 metafile 来浏览或显示资料。但是，另一个使用常规的 CF_TEXT、CF_BITMAP 、 CF_DIB 、 CF_METAFILEPICT 或 CF_ENHMETAFILE 格式呼叫 GetClipboardData 的程式将不能取得这个资料。

如果用其中一种格式把资料放入剪贴簿中，则必须使用同样的格式读出资料。但是，如何知道资料是来自程式的另一个执行实体，还是来自使用其中某种资料格式的另一个程式呢？这里有一种方法，可以透过下列呼叫首先获得剪贴簿所有者：

```
hwndClipOwner = GetClipboardOwner () ;
```

然後可以得到此视窗代号的视窗类别名称：

```
TCHAR szClassName [32] ;
//其他行程式
GetClassName (hwndClipOwner, szClassName, 32) ;
```

如果类别名称与程式名称相同，那么资料是由程式的另一个执行实体传送到剪贴簿中的。

使用自订资料格式的第二种方法涉及到 CF_OWNERDISPLAY 旗标。SetClipboardData 的整体记忆体代号是 NULL：

```
SetClipboardData (CF_OWNERDISPLAY, NULL) ;
```

这是某些文书处理程式在 Windows 的剪贴簿浏览器的显示区域中显示格式化文字时所采用的方法。很明显地，剪贴簿浏览器不知道如何显示这种格式化文字。当一个文书处理程式指定 CF_OWNERDISPLAY 格式时，它也就承担起在剪贴簿浏览器的显示区域中绘图的责任。

由於整体记忆体代号为 NULL，所以用 CF_OWNERDISPLAY 格式（剪贴簿所有者）呼叫 SetClipboardData 的程式必须处理由 Windows 发往剪贴簿所有者的延迟提出讯息、以及 5 条附加讯息。这 5 个讯息是由剪贴簿浏览器发送到剪贴簿所有者的：

WM_ASKCBFORMATNAME 剪贴簿浏览器把这个讯息发送到剪贴簿所有者，以得到资料格式名称。lParam 参数是指向缓冲区的指标，wParam 是这个缓冲区能

容纳的最大字元数目。剪贴簿所有者必须把剪贴簿资料格式的名字复制到这个缓冲区中。

WM_SIZECLIPBOARD 这个讯息通知剪贴簿所有者，剪贴簿浏览器的显示区域大小已发生了变化。wParam 参数是剪贴簿浏览器的代号，lParam 是指向包含新尺寸的 RECT 结构的指标。如果 RECT 结构中都是 0，则剪贴簿浏览器退出或最小化。尽管 Windows 的剪贴簿浏览器只允许它自己的一个执行实体执行，但其他剪贴簿浏览器也能把这个讯息发送给剪贴簿所有者。应付多个剪贴簿浏览器并非不可能（假定 wParam 标识特定的浏览器），但剪贴簿所有者处理起来也不容易。

WM_PAINTCLIPBOARD 这个讯息通知剪贴簿所有者修改剪贴簿浏览器的显示区域。同时，wParam 是剪贴簿浏览器视窗的代号，lParam 是指向 PAINTSTRUCT 结构的整体指标。剪贴簿所有者可以从此结构的 hdc 栏中得到剪贴簿浏览器装置内容的代号。

WM_HSCROLLCLIPBOARD 和 WM_VSCROLLCLIPBOARD 这两个讯息通知剪贴簿所有者，使用者已经卷动了剪贴簿浏览器的卷动列。wParam 参数是剪贴簿浏览器视窗的代号，lParam 的低字组是卷动请求，并且，如果低字组是 SB_THUMBPOSITION，那么 lParam 的高字组就是滑块位置。

处理这些讯息比较麻烦，看来并不值得这样做。但是，这种处理对使用者来说是有益的。当从文书处理程式把文字复制到剪贴簿时，使用者在剪贴簿浏览器的显示区域中看见文字还保持著格式时心里会舒坦些。

使用私有剪贴簿资料格式的第三种方法是注册自己的剪贴簿格式名。您向 Windows 提供格式名，Windows 给程式提供一个序号，它可以用作 SetClipboardData 和 GetClipboardData 的格式参数。一般来说，采用这种方法的程式也要以一种标准格式把资料复制到剪贴簿。这种方法允许剪贴簿浏览器在它的显示区域中显示资料（没有与 CF_OWNERDISPLAY 相关的冲突），并且允许其他程式从剪贴簿上复制资料。

例如，假定我们已经编写了一个以点阵图格式、metafile 格式和自己的已注册的剪贴簿格式把资料复制到剪贴簿中的向量绘图程式。剪贴簿浏览器将显示 metafile 或者点阵图，其他从剪贴簿上读取点阵图和 metafile 的程式将获得这几种格式。但是，当我们的向量绘图程式需要从剪贴簿上读数据时，它会按照自己已注册的格式复制资料，这是因为这种格式可能包含著比点阵图档案或者 metafile 更多的资讯。

程式透过下面的呼叫来注册一个新的剪贴簿格式：

```
iFormat = RegisterClipboardFormat (szFormatName) ;
```

iFormat 的值介於 0xC000 和 0xFFFF 之间。剪贴簿浏览器（或一个通过呼叫 EnumClipboardFormats 取得目前所有剪贴簿资料格式的程式）可以取得这种资料格式的 ASCII 名称，这是通过下面呼叫实作的：

```
GetClipboardFormatName (iFormat, psBuffer, iMaxCount) ;
```

Windows 将多达 iMaxCount 个字元复制到 psBuffer 中。

使用这种方法把资料复制到剪贴簿中的程式写作者，可能需要公开资料格式名称和实际的资料格式。如果这个程式流行起来，那么其他程式就会以这种格式从剪贴簿中复制资料。

实作剪贴簿浏览器

监视剪贴簿内容变化的程式称为「剪贴簿浏览器」。您可以在 Windows 中得到一个剪贴簿浏览器，但是您也可以编写自己的剪贴簿浏览器程式。剪贴簿浏览器通过传递到浏览器视窗讯息处理程式的讯息来监视剪贴簿内容的变化。

剪贴簿浏览器链

任意数量的剪贴簿浏览器应用程式都可以同时在 Windows 下执行，它们都可以监视剪贴簿内容的变化。但是，从 Windows 的角度来看，只存在一个剪贴簿浏览器，我们称之为「目前剪贴簿浏览器」。Windows 只保留一个识别目前剪贴簿浏览器的视窗代号，并且当剪贴簿的内容发生变化时只把讯息发送到那个视窗中。

剪贴簿浏览器应用程式有必要加入「剪贴簿浏览器链」，以便执行的所有剪贴簿浏览器都可以收到 Windows 发送给目前剪贴簿浏览器的讯息。当一个程式将自己注册为一个剪贴簿浏览器时，它就成为目前的剪贴簿浏览器。Windows 把先前的目前浏览器视窗代号交给这个程式，并且此程式将储存这个代号。当此程式收到一个剪贴簿浏览器讯息时，它把这个讯息发送给剪贴簿链中下一个程式的视窗讯息处理程式。

剪贴簿浏览器的函式和讯息

程式透过呼叫 SetClipboardViewer 函式可以成为剪贴簿浏览器链的一部分。如果程式的主要作用是作为剪贴簿浏览器，那么这个程式在 WM_CREATE 讯息处理期间可以呼叫这个函式，该函式传回前一个目前剪贴簿浏览器的视窗代号。程式应该把这个代号储存在静态变数中：

```
static HWND hwndNextViewer ;
//其他行程式
case  WM_CREATE :
```

```
      //其他行程式
      hwndNextViewer = SetClipboardViewer (hwnd) ;
```

如果在 Windows 的一次执行期间，您的程式成为剪贴簿浏览器的第一个程式，那么 hwndNextViewer 将为 NULL。

不管剪贴簿中的内容怎样变化，Windows 都将把 WM_DRAWCLIPBOARD 讯息发送给目前的剪贴簿浏览器（最近注册为剪贴簿浏览器的视窗）。剪贴簿浏览器链中的每个程式都应该用 SendMessage 把这个讯息发送到下一个剪贴簿浏览器。浏览器链中的最後一个程式（第一个将自己注册为剪贴簿浏览器的视窗）所储存的 hwndNextViewer 为 NULL。如果 hwndNextViewer 为 NULL，那么程式只简单地将控制项权还给系统而已，而不向其他程式发送任何讯息（不要把 WM_DRAWCLIPBOARD 讯息和 WM_PAINTCLIPBOARD 讯息混淆了。WM_PAINTCLIPBOARD 是由剪贴簿浏览器发送给使用 CF_OWNERDISPLAY 剪贴簿资料格式的程式，而 WM_DRAWCLIPBOARD 讯息是由 Windows 发往目前剪贴簿浏览器的）。

处理 WM_DRAWCLIPBOARD 讯息的最简单方法是将讯息发送给下一个剪贴簿浏览器（除非 hwndNextViewer 为 NULL），并使视窗的显示区域无效：

```
case  WM_DRAWCLIPBOARD :
      if    (     hwndNextViewer)
                        SendMessage (hwndNextViewer, message, wParam, lParam) ;

      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
```

在处理 WM_PAINT 讯息处理期间，通过使用常规的 OpenClipboard、GetClipboardData 和 CloseClipboard 呼叫可以读取剪贴簿的内容。

当某个程式想从剪贴簿浏览器链中删除它自己时，它必须呼叫 ChangeClipboardChain。这个函式接收脱离浏览器链的程式之视窗代号，和下一个剪贴簿浏览器的视窗代号：

```
ChangeClipboardChain (hwnd, hwndNextViewer) ;
```

当程式呼叫 ChangeClipboardChain 时，Windows 发送 WM_CHANGECBCHAIN 讯息给目前的剪贴簿浏览器。wParam 参数是从链中移除它自己的那个浏览器视窗代号（ChangeClipboardChain 的第一个参数），lParam 是从链中移除自己後的下一个剪贴簿浏览器的视窗代号（ChangeClipboardChain 的第二个参数）。

当程式接收到 WM_CHANGECBCHAIN 讯息时，必须检查 wParam 是否等於已经储存的 hwndNextViewer 的值。如果是这样，程式必须设定 hwndNextViewer 为 lParam。这项工作保证将来的 WM_DRAWCLIPBOARD 讯息不会发送给从剪贴簿浏览器链中删除了自己的视窗。如果 wParam 不等於 hwndNextViewer ，并且 hwndNextViewer 不为 NULL，则把讯息送到下一个剪贴簿浏览器。

```
case  WM_CHANGECBCHAIN :
```

```
      if ((HWND) wParam == hwndNextViewer)
                  hwndNextViewer = (HWND) lParam ;

      else if (hwndNextViewer)
                              SendMessage  (hwndNextViewer,  message,  wParam,
lParam) ;
      return 0 ;
```

不一定要使用 else if 叙述，它只用於保证 hwndNextViewer 为非 NULL 的值。hwndNextViewer 的值为 NULL 时，执行这段程式码的程式就是链中最後一个浏览器，而这是不可能的。

当程式快结束时，如果它仍然在剪贴簿浏览器链中，则必须从链中删除它。您可以在处理 WM_DESTROY 讯息时呼叫 ChangeClipboardChain 来完成这项工作。

```
case  WM_DESTROY :
      ChangeClipboardChain (hwnd, hwndNextViewer) ;
      PostQuitMessage (0) ;
      return 0 ;
```

Windows 还有一个允许程式获得第一个剪贴簿浏览器视窗代号的函式：

```
hwndViewer = GetClipboardViewer () ;
```

一般来说不需要这个函式。如果没有目前的剪贴簿浏览器，则传回值为 NULL。

下面是一个说明剪贴簿浏览器链如何工作的例子。当 Windows 刚启动时，目前剪贴簿浏览器是 NULL：

剪贴簿浏览器： NULL

一个具有 hwnd1 视窗代号的程式呼叫 SetClipboardViewer。这个函式传回的 NULL 成为这个程式中的 hwndNextViewer 值：

目前剪贴簿浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

第二个具有 hwnd2 视窗代号的程式呼叫 SetClipboardViewer ，并传回 hwnd1：

目前的剪贴簿浏览器： hwnd2

hwnd2 的下一个浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

每三个程式 (hwnd3)和第四个程式 (hwnd4) 也呼叫 SetClipboardViewer ，并且传回 hwnd2 和 hwnd3：

目前的剪贴簿浏览器： hwnd4

hwnd4 的下一个浏览器： hwnd3

hwnd3 的下一个浏览器： hwnd2

hwnd2 的下一个浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

当剪贴簿的内容发生变化时，Windows 发送一个 WM_DRAWCLIPBOARD 讯息给 hwnd4， hwnd4 发送讯息给 hwnd3，hwnd3 发送讯息给 hwnd2，hwnd2 发送讯息给 hwnd1，hwnd1 传回。

现在 hwnd2 决定通过下列呼叫从链中删除自己：

ChangeClipboardChain (hwnd2, hwnd1) ;

Windows 将 wParam 等於 hwnd2、lParam 等於 hwnd1 的 WM_CHANGECBCHAIN 讯息发送给 hwnd4。由於 hwnd4 的下一个测览器是 hwnd3，所以 hwnd4 把这个讯息传给 hwnd3。现在 hwnd3 注意到 wParam 等於它的下一个测览器(hwnd2)，所以将下一个浏览器设定为 lParam (hwnd1)并且传回。这样工作就完成了。现在剪贴簿浏览器链如下：

目前剪贴簿浏览器： hwnd4

hwnd4 的下一个浏览器： hwnd3

hwnd3 的下一个浏览器： hwnd1

hwnd1 的下一个浏览器： NULL

一个简单的剪贴簿浏览器

剪贴簿浏览器不一定要像 Windows 所提供的那样完善，例如，剪贴簿浏览器可以只显示一种剪贴簿资料格式。程式 12-2 中所示的 CLIPVIEW 程式是一种只能显示 CF_TEXT 格式的剪贴簿浏览器。

程式 12-2 CLIPVIEW

```
CLIPVIEW.C
/*------------------------------------------------------------------------
      CLIPVIEW.C --     Simple Clipboard Viewer
                                              (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

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

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

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

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

      hwnd = CreateWindow (szAppName,
                                 TEXT ("Simple Clipboard Viewer (Text Only)"),
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     NULL, NULL, hInstance, NULL) ;

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

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

LRESULT CALLBACK WndProc     (     HWND hwnd, UINT message, WPARAM wParam,LPARAM
lParam)
{
      static HWND      hwndNextViewer ;
      HGLOBAL                hGlobal ;
      HDC                    hdc ;
      PTSTR                  pGlobal ;
      PAINTSTRUCT               ps ;
      RECT                   rect ;

      switch (message)
      {
      case  WM_CREATE:
```

```
                hwndNextViewer = SetClipboardViewer (hwnd) ;
                return 0 ;

     case  WM_CHANGECBCHAIN:
                if ((HWND) wParam == hwndNextViewer)
                                hwndNextViewer = (HWND) lParam ;

                else if     (hwndNextViewer)
                                        SendMessage (hwndNextViewer, message,
wParam, lParam) ;

        return 0 ;
     case  WM_DRAWCLIPBOARD:
                if (hwndNextViewer)
                                SendMessage (hwndNextViewer, message, wParam,
lParam) ;

                InvalidateRect (hwnd, NULL, TRUE) ;
                return 0 ;

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

#ifdef UNICODE
                hGlobal = GetClipboardData (CF_UNICODETEXT) ;
#else
                hGlobal = GetClipboardData (CF_TEXT) ;
#endif
                if (hGlobal != NULL)
                {
                           pGlobal = (PTSTR) GlobalLock (hGlobal) ;
                           DrawText (hdc, pGlobal, -1, &rect, DT_EXPANDTABS) ;
                           GlobalUnlock (hGlobal) ;
                }

                CloseClipboard () ;
                EndPaint (hwnd, &ps) ;
                return 0 ;

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

CLIPVIEW 依上面所讨论的方法来处理 WM_CREATE、WM_CHANGECBCHAIN、WM_DRAWCLIPBOARD 和 WM_DESTROY 讯息。WM_PAINT 讯息处理打开剪贴簿，并用 CF_TEXT 格式呼叫 GetClipboardData。如果函式传回一个整体记忆体代号，那么 CLIPVIEW 将锁定它，并用 DrawText 在显示区域显示文字。

处理标准格式（如 Windows 提供的那个剪贴簿一样）以外的资料格式的剪贴簿浏览器还需要完成一些其他工作，比如显示剪贴簿中目前所有资料格式的名称。使用者可以通过呼叫 EnumClipboardFormats 并使用 GetClipboardFormatName 得到非标准资料格式名称来完成这项工作。使用 CF_OWNERDISPLAY 资料格式的剪贴簿浏览器必须把下面四个讯息送往剪贴簿资料的拥有者以显示该资料：

- WM_PAINTCLIPBOARD
- WM_SIZECLIPBOARD
- WM_VSCROLLCLIPBOARD
- WM_HSCROLLCLIPBOARD

如果您想编写这样的剪贴簿浏览器，那么必须使用 GetClipboardOwner 获得剪贴簿所有者的视窗代号，并当您需要修改剪贴簿的显示区域时，将这些讯息发送给该视窗。