Windows 95 程式設計指南　Windows 95:A Developer's Guide

光碟片內容

■Voyeur
■AppLog
■Touch
■VerShow
■MagCrack

■DlgTmplt
■Layout
■SuperCls
■AniBtn
■Recorder
■DFSrc
■SendKeys

碁峰資訊股份有限公司
GOTOP INFORMATION INC.
URL: http://www.gotop.com

02847792　NT$ 720

程式設計指南 Windows 95

Jeffrey Richter
Jonathan Locke 著
李書良 翻譯
侯俊傑 總監

碁峰
CK032

GOTOP

◆一本權威的 Windows 95 和 Windows NT 程式設計書籍

◆內含許多困難題目的簡明答案

◆光碟片內含多媒體效果、原始碼及程式展示

高等 WIN 32 程式設計

Windows 95 程式設計指南

Windows 95 : A Developer's Guide

本書所有內容都適用於 Windows 95 和 Windows NT

M&T BOOKS

Jeffrey Richter
Jonathan Locke 著

碁峰資訊

李書良 翻譯　侯俊傑 總監

LET'S GO TOP WITH GOTOP

本中文版，從某種角度來說，或許我有權將之公開。但是說到原書所附之源碼，由於完全沒有我的參與，所以我也完全沒有立場將之放在網上供人下載。請不要有任何人來詢問或要求書附源碼，我不會回應。

本中文版 PDF 免費開放下載，已徵得所有相關人士之口頭應允，感謝這些心存善念、助人爲樂的朋友。但如本電子書免費開放致生任何法律問題，概由本人承擔。

-- 侯捷

2001/06/01. 新竹.臺灣

侯俊傑 序

我常常被要求推薦 Win32 程式設計之優良書籍。在 SDK programming 層面上，從來我只推薦兩本書，一本是 Charles Petzold 的 ***Programming Windows 95***，另一本就是 Jeffrey Richter 的 ***Windows 95 : A Developer's Guide***。我這麼說難免造成遺珠之憾，但是泰山北斗已現，又何必案牘勞形於浩瀚書海之中！這兩本書都是從 Windows 3.x 時代一路改版而來，斬將搴旗，追奔逐北，成就一身榮光。

Petzold 的 ***Programming Windows 95*** 早享大名，旦夕聞之。遺憾的是 Jeffrey 的 ***Windows 95 : A Developer's Guide*** 在臺灣好似乎沒有得到它該有的知名度與尊榮。1995 年末我在偶然機會下於台北購得此書，發現它已從原來的 3.0 版、3.1 版演化為 95 版，精采之處依舊在，甚至猶有過之。我說猶有過之，是因為 Jeffrey 專司高難度技術，常常直指系統核心，做些走鋼絲的危險生，這種行為在 Windows 95 或 Windows NT 中可要比 Windows 3.x 更困難多了。

其後，我觀察電腦書市兩年之久，竟無一家出版公司進行此書之中譯，甚至連原文本也漸漸看不到了。我認為舉薦好書是我的責任之一，於是在 1997 年六月向碁峰力薦此書，並很快獲得回應。我雖然沒有時間親自完成此書之中譯，但此書由我的得意學生李書良先生完成，復由我擔任總監，校正並修潤全文，亦堪慰矣。

此書所選皆為高階罕見之題材，讀者可從章節一目瞭然。其中涵蓋的window structure、window class structure、hook、subclassing、superclassing、processing keystrokes...等題目更與作業系統或物件導向原理相呼應。對於擁有相當功力之 Windows 程式員而言，此書實如至寶。但也就是說，此書絕對不適合 Windows Programming 初學者。

閱讀此書，我有兩個建議，可收旁徵博引左右逢源之效：

1. 先讀過 Charlest Petzold 之 ***Programming Windows 95***，或具備相等功力。
2. 再時閱讀 Jeffrey Richter 之另一本巨著：***Advanced Windows***，以求對 Win32 作業系統有更深入的認識。作業系統的根基不夠，程式設計能力就不可能有多好 – 我希望每一個人都能夠明白這件事情。

這本書的原文名稱，說實在沒有符合其內涵。第一，書中所有技術與實作碼，不僅適用於 Windows 95，在 Windows NT 上也都通過了檢驗；第二，書籍內容的深度不是 "A Developer's Guide" 這幾個字所能披露。我原想依其實際內涵把中文書名改為 **高等 Win32 程式設計**，終因為了忠實原著而放棄。誌之於斯，以為紀念。

此譯本中的每一個句子，都經過我的檢驗與修潤。如果書中仍有誤謬，一切責任應歸於我。

侯俊傑　1997/10/07　于新竹
jjhou@ccca.nctu.edu.tw
FAX : 886-3-5733976

P.S.　拿到書，請儘快安裝書附的 CD-ROM 碟片，享受一下 Jeffrey Richter 的程式設計與包裝功力。你會看到令你目瞪口呆的安裝畫面，以及他的一段 VCR。

譯 序

Windows 95 : A Developer's Guide 書籍作者 Jeffrey Richer 可謂 Windows 界的傳奇人物。能有機會翻譯這樣的大師級作品，真是一個難得的經驗，也是我的一項挑戰。當我欣然接受本書的翻譯工作，心中湧起了許多的理想與抱負，心想我一定要翻譯好此書，讓讀者能輕易一窺 Windows 的奇境。

這是一本 Win32 SDK 程式設計的高階好書，很適合想在 Windows 95 和 Windows NT 環境上精研程式開發樂趣的讀者們閱讀，其內容包含了 Custom Controls、Subclassing and Superclassing、Hook、Drag and Drop、以及 Version Control 等等。不僅介紹了 Win32 程式設計的技巧與函式內部的動作，更重要的是記載了作者多年的寶貴經驗。

從程式設計的角度來看，一個好的程式不應只是功能上的發揮，清楚且容易地讓人瞭解其原始碼的來龍去脈亦是同等重要。本書教你如何利用「訊息剖析器」的技巧來處理訊息，使程式碼顯明易懂，並能以最短時間達到最大成效，節省你的軟體開發時間。本書所涵蓋的知識是每一位程式設計者所必備的，我想此書必將使您受益匪淺。

翻譯此書的過程中，我自己受益良多，相信您也會因為從書中習得許多寶貴的知識與經驗而讚不絕口。期望藉由本書的誕生，使您能完全擺脫原文的掣肘，我的目的就達到了。

原書中有不少的錯別字，可能是作者當初急於趕上 Windows 95 的熱潮，一時疏忽所致，別擔心，我於譯書中已做了更正。

本書得以完成要感謝的人很多，最重要的是要感謝侯俊傑老師不時給我技術上的指導與鼓勵，使我成長許多，並讓我踏上了一個新的境域，有了新的體認。再者感謝我的父母親不時給我精神上的鼓勵與物質上的協助；尋尋不時從我一次次的挫敗中拉起我來，以及我最可愛的妹妹，在這些日子中犧忍著夜晚的搖滾樂，鍵盤的敲擊聲，真的是難為她了。還要感謝好友黃志仁、楊叔倫與陳世雅的協助，不時提供寶貴的建議與經驗。最後，感謝碁峰出版社給我的機會與協助，使此書得以順利完成。

我盡我最大的努力，呈獻最好的翻譯品質給各位讀者。如您對於這本譯書有任何批評與建議，懇請不吝指教，謝謝！

李書良 1997.9.30 于台北
shuliang@ms8.hinet.net

笑看失與落　　模糊對視之
豁達人生路　　歡喜在心頭

目　錄

簡 介

雖然寫書是一件既艱辛又繁複的工作，但我真的很喜歡這份工作，早在 1994 年以前我就已經開始接觸 Windows 95 的程式設計工作了，並且也看了許多既刺激又好玩而且有趣的成果與應用。撰寫這本書的同時，我也分享了 Windows 95 工作小組的辛酸與喜悅。我覺得很幸運，因爲沒有多少人有機會看到一個新的作業系統的誕生及其從頭到尾的發展過程。Windows 95 上市之後，獲得空前的歡迎與成功，每個人都很容易因此忘掉過去那段冗長而艱辛的歷程。但是我們不容易忘記，因爲在撰寫此書的過程中，我們同樣也走過了那條路，並從中獲得了許多心得。我們要向 Windows 95 工作小組的成員們致上我們的恭喜與敬意，他們寫出了一個改變全世界的作業系統。

試著把這本書想像是 Jeff 和 Jon 告訴你的一組一組小故事。每一章告訴你一個我們在 Windows 95 高速公路上令人興奮的冒險故事，由內而外，由淺入深，詳細探究 Windows 95 作業系統的奧秘，並提出適當而正確的解釋與說明，不時則傳達一些寶貴經驗，使你在開發 Windows 程式的過程中避免一些不必要的困擾與錯誤的方向，節省你的開發時間。我誠摯希望，藉由本書的誕生，能帶著你一同揭開 Win32 程式設計的神秘面紗。

讀者背景

本書定位在設計一些較高階的 Win32 程式，是一本中高階書籍，前提是假設讀者已有寫過 Windows 程式的經驗。你應該知道如何建立一個 Win32 程式，和一個 Win32 DLL（動

態聯結函式庫），並具備以下經驗：

- 註冊一個視窗類別
- 建立一個自己的視窗處理程序（視窗函式）
- 建立 model 與 modeless 的對話盒
- 與子視窗（控制元件）通訊

本書的主要目的是讓你瞭解 Windows 95 和 Windows NT 的運作原理，以及 Win32 程式設計的發展程序，並在各章節中深入探討各個主題，舉出適當的例子，使你更加得心應手。

本書的範例程式

學習 Win32 程式設計的過程中，實際撰寫程式碼是相當重要的，你可以因此實現自己的想法，並對 Win32 程式設計觀念的印象更深刻。書中每一章節不但詳細介紹 Win32 程式設計上重要的基本觀念，隨後的範例程式更能幫助你將觀念具體化。每一個應用程式皆是我精心設計出來的，並經過不斷的嘗試與修改。透過此書的介紹與描述，你將能夠習得許多有用的技巧與方法。程式中的大量註解將使你對整個程式更加瞭解。

版權聲明

每一個範例程式中的每一個程式片段，你都可以截取放到你自己所發展的程式中，事實上只要它適合你，我們很高興你採用它。版權聲明的主要目的只是爲了保護這些東西在未來本書新版中的權利。你從本書所截取或衍生的任何程式碼，必須含入以下宣告：

C 語言的使用

撰寫本書的時候，我很苦惱到底要使用 C 語言還是 C++ 語言。對於較大的程式開發計劃，我通常使用 C++ 語言，但大部分的 Windows 程式員事實上還未開始使用 C++ 語言，而我不希望失去廣大的讀者，所以書中每一個程式皆使用 C 語言來開發。

訊息剖析器 (Message Cracker Macro)

如果你未曾使用過 C++ 以及 Windows 類別庫（像是 MFC）來開發 Win32 程式，我建議你使用訊息剖析器（Message Cracker Macro），它被定義在 WindowsX.H 表頭檔中。這些巨集使你能夠很容易地開發出易於閱讀與維護的程式。欲對訊息剖析器有更多瞭解，請看 Win32 SDK 的電子書 "***Programming techniques***"，其中有詳盡的說明。

不相關的程式碼

我試圖去除任何與我所選擇的主題技術不相干的 Windows 程式碼，不幸的是在開發 Windows 程式時這個理想很難實現，但我仍盡最大的努力去做。舉個例子，市面上大部分的 Windows 程式設計書籍，於每一個程式中皆有註冊視窗類別的程式碼（這些碼份量可不少），而本書大部分程式的使用者介面都是使用一個簡單的 DialogBox 函式所建立的 model 對話盒。於是，本書大部分程式都不再需要重新初始化一個 WNDCLASSEX 結構，或是呼叫 RegisterClassEx 函式。

STRICT 編譯

書中所有程式都以 STRICT 方式來編譯，可以因此找出許多常見的程式錯誤。舉個例，傳遞一個不正確的 handle 型態給函式，會在編譯時就被發現而予以修正，不會等到執行時才出錯。如果你想瞭解 STRICT 的更多資訊，請參閱 Win32 SDK 技術文件。

錯誤檢查

軟體開發過程中，大部分程式碼都用在錯誤檢查並做適當處理上面。遺憾的是，錯誤處理雖能使程式發展更加嚴謹，卻會大大增加程式碼的量以及複雜度。爲使本書容易瞭解而不致混亂，我並沒有在程式中放入大量的錯誤檢查碼，但我強烈建議你仔細閱讀本書程式，並適當加入一些錯誤檢查碼，這會使你的程式更嚴謹，而你的程式功力無形中也會大大提升。

零錯誤程式

每一位程式設計者都希望自己撰寫的程式沒有臭蟲，但人非聖賢，有臭蟲實在是難免的。我不斷嘗試各種方法，找出本書各個程式的錯誤，以求其正確性，但難免有疏失。如果你發現臭蟲的話，懇請你告訴我，請寄 v-jeffrr@microsoft.com，謝謝。（也請你告訴侯俊傑：jjhou@ccca.nctu.edu.tw，謝謝）

開發平台

本書大部分的研究與發展，都是在 Intel 機器上完成。我也曾在 MIPS 和 DEC Alpha AXP 機器上重新編譯並測試過所有的範例程式，都沒有問題！我用的是 Visual C++ 2.x 的編譯器和聯結器。

編譯器

本書有些程式使用了微軟編譯器所提供的特殊編譯選項，程式原始碼可以因此少做一些修改。由於將來你可能使用另一套編譯器，所以儘量少更改原始碼是很重要的。

自從 Visual C++ 2.0 問世之後，許多程式開始使用 Windows 95 身上的 Win32 特性。但光使用 Visual C++ 2.0 所附的表頭檔和函式庫是不夠的，你還需要 Win32 SDK 的表頭檔和函式庫。Visual C++ 2.2（及其新版）都已含入適當的檔案，也就是說你可以完全不需要 Win32 SDK 了。

Unicode（萬國字元組）

本書撰寫時已將 Unicode（萬國字元組）考慮進來，所有程式皆能編譯成原生（native）的 Unicode 版本或 ANSI 版本。關於 Unicode，請參考附錄 A。

Perl

本書附錄 B 中的 MsgCrack.BAT 不是一個批次檔，而是一個 Perl 語言描述檔（script）。要執行這個描述檔，你需要一個 "Win32 port of Perl"（可在 Windows95 和 Windows NT 下執行），本書光碟片中的 \MsgCrack.0B\NTPerl 子目錄下有這玩意兒。

以下敘述摘錄自 Perl 技術文件：

> "Perl—Practical Extraction and Report Language ...Perl is an interpreted language optimized for scanning arbitrary text files，extracting information from those test files，and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful(tiny, elegant, minimal). It combines(in the author's option, anyway)some of the best features of C, sed , awk and sh. So people familiar with those languages should have a little difficult with it ..."

如果你使用 Perl 語言來撰寫程式，你可以使用 MsgCrack.BAT 來執行你所寫的程式。Perl 是一個非常好用的語言，我建議你試著去瞭解它，你會發現它很容易上手。但是請注意，MsgCrack.BAT 是一項免費產品，尚未獲得廣泛的支援。

範例程式的安裝

隨書所附的光碟片中包含了本書所有應用程式的原始碼，另外還包含了適合於 x86、MIPS 和 Alpha AXP 機器的執行檔（EXEs）和動態聯結函式庫（DLLs）。所有檔案皆未經過壓縮，可直接從光碟片中讀取並執行之。

光碟片的根目錄中包含了一個安裝程式和一個表頭檔 Win95ADG.H（我將於附錄 A 討論這個檔案）。根目錄下的其他子目錄，像是 x86.BIN、MIPS.BIN 和 Alpha.BIN 中分別包含了各種不同硬體平台上的執行檔和動態聯結函式庫，分別在不同的機器上執行。如果你使用的是 Windows NT 作業系統，但不在 x86、MIPS 或 Alpha 平台上執行，你仍然擁有所有的範例程式原始碼，但必須重新編譯與聯結才可執行。稍後「安裝在未被支援的 CPU 平台上」一節會討論如何安裝這些範例程式。

光碟片中的其它子目錄包含範例程式的原始碼，每一個程式皆有自己的子目錄，每一個子目錄的主名稱就是範例程式名稱，副名稱爲章節編號，例如 Dlgsize.03 表示第 3 章的 Dlgsize 範例程式。

如果你只是想看一看程式碼的執行結果，無需拷貝任何檔案到你的硬碟中。但如果你想對程式做某些修改、編譯或除錯動作，你就必須將檔案拷貝到你的硬碟來。接下來的兩小節將介紹如何在 Windows 95 或 Windows NT 作業系統上使用此光碟片。

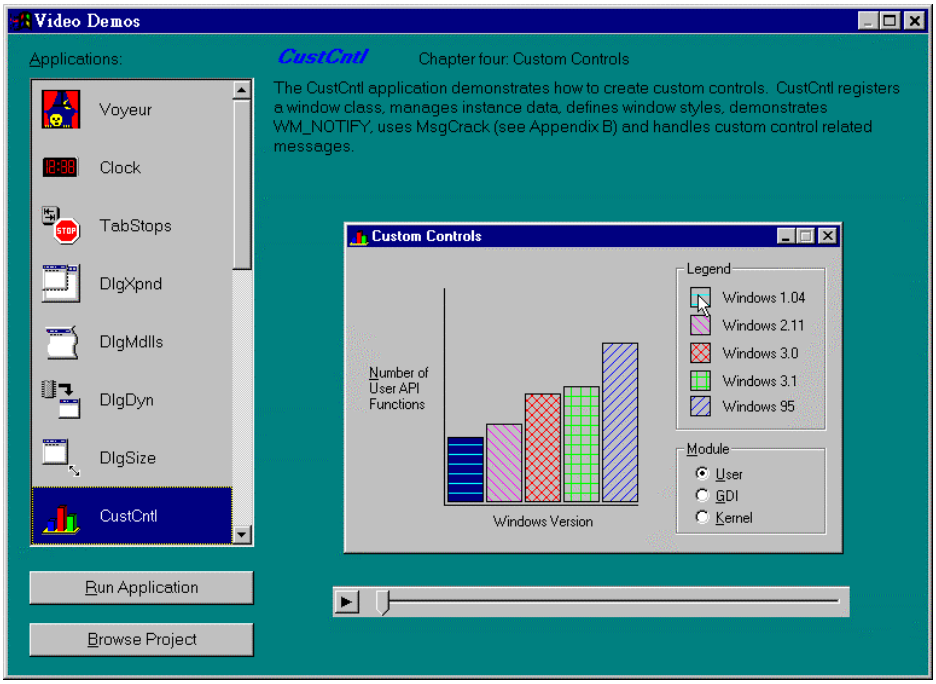
自動播放的 Welcome 程式

當你將光碟片放入光碟機中，Windows 95 會自動偵測並執行 Welcome 程式，介紹本書的特色與綱要。畫面上會出現以下幾個選項：

選項	說明
Introduction	本書簡介。
Video Demos	描述本書各範例程式的重要功能及特色。
Setup	安裝本書範例程式。可選擇是否要安裝可執行檔（EXE）到你的硬碟中。
Explorer	讓你直接從光碟片中瀏覽並執行程式。
Credit	本書幕前幕後工作者介紹。
Training	Jeff 的 Win32 程式設計研討會相關資訊。

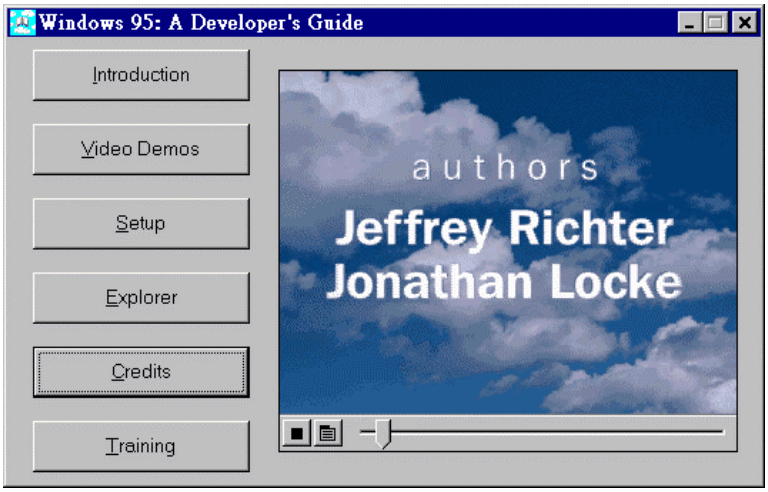
這是【Introduction】畫面：

這是【Video Demos】畫面：



這是【Explorer】畫面：

這是【Credits】畫面：



安裝在未被支援的 CPU 平台上

如果你使用的是一台未被支援的 CPU 硬體平台上，你可能無法執行 Setup.exe 程式讓它將範例程式安裝到你的硬碟中，但你可以執行 Install.bat 安裝本書的範例程式，它位於光碟片的根目錄下。

欲執行 Install.bat，你必須指定安裝路徑。舉個例子，假設你的光碟機是 D 碟，你要安裝範例程式到 C 碟的 Win95ADG 目錄下（如果此目錄目前不存在，將會被自動建立），動作如下（在 DOS 視窗中執行）：

```
C:\> D:\INSTALL  C:\Win95ADG
```

這個 Install.bat 會將範例程式的原始碼拷貝到你的硬碟中，每一個範例程式都會各安其份，被拷貝到特定子目錄下，子目錄的主名稱爲範例程式名稱，副名稱爲章節編號。例如子目錄 Clock.02 表示其內放置的是第 2 章的 Clock 程式。

第 1 章

從生到死
－一個完整的 Win32 程式

看來似乎很簡單！你，使用者，要的就是執行一個程式。首先，你必須熟悉如何使用 Windows 作業系統去執行一個程式，程式執行之後，一般情況下你將會在螢幕上看到一個視窗，這個視窗會提供一些選項供你選擇，以便執行各種不同的功能。當你想要結束程式，只須選取視窗左上方的系統選單中的「關閉」項目即可。

從使用者的觀點來看，一個視窗就是一個程式。但對程式設計者來說，程式和視窗是兩種完全不同的概念，絕對不可以混淆。事實上，有些程式並不產生任何視窗。早期，瞭解程式與視窗的不同，是程式設計者在撰寫一個好的 Win32 程式時的重要關鍵。當我開始撰寫第一個 Windows 程式時，我並沒有完全瞭解它們的不同，並因而面臨了一些難以解決的問題。

這一章主要的目的在介紹 Win32 行程（process）、視窗類別（window class）與視窗實體（window instance）的基本觀念。我將一一探討它們彼此之間的關係。

一個 Win32 程式

你可以使用 C 或 C++ 語言來開發一個 Win32 程式。然後，當你編譯你的程式碼，可能需要和一些 LIB 檔、OBJ 檔聯結，以產生 EXE 可執行檔。這個執行檔平時存放於硬碟中，直到使用者去執行它（通常是藉由工作列 Taskbar 去執行），它才會動作。

當系統執行這個程式時，會爲它產生一個新的行程（process），這個行程不多不少地擁有 4GB 虛擬位址空間。一開始，這些位址空間並沒有任何資料存在，而後系統開始將一些程式碼和資料放入其中。首先，系統會尋找這個程式在硬碟中的存放位置，並將其全部內容載入到這個 4GB 虛擬位址空間中。此時，系統也會將此程式所用到的所有動態聯結函式庫（DLLs）的內容全部載入到這個 4GB 虛擬位址空間中[1]。

之後，行程位址空間中便擁有了這個行程的執行檔和動態聯結函式庫所要用到的程式碼和資料。系統會爲這個行程產生一個主執行緒（primary thread）。執行緒是一個很抽象的觀念，對於一個 Win32 程式設計新手而言，一開始很難接受這個觀念。執行緒是由作業系統負責維護，它僅承擔程式碼的執行責任。當一個新的行程誕生，系統同時也會爲它產生一個新執行緒（稱爲行程的主執行緒）。系統首先會初始化這個執行緒，主要目的是告知程式碼在行程位址空間中開始執行的位址。通常，主執行緒會被告知以 C-Runtime startup code 做爲開始。

C-Runtime startup code 承擔著 C-Runtime 系統元件初始化的工作，使你在程式中使用的任何函式（像是 malloc 或 signal）得以正確運作，。C-Runtime startup code 被初始化後，程式便可開始執行。程式進入點是 WinMain 函式，型式如下：

---

[1] 事實上，系統並沒有將全部的 EXE 檔載入到記憶體中，而是使用了一個較有效率的方法去善用記憶體資源。基本上，身爲程式員的我希望能將所有的程式碼和資料都包含在 EXE 檔中，也希望所有的程式碼和資料包含在每一個動態聯結函式庫（DLL）裡，並且真的將其全部內容載入到行程位址空間中。系統利用一塊塊的 4KB 記憶體，稱爲頁（page），將資料在記憶體與硬碟之間交換。

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstExePrev,
                    LPSTR lpszCmdLine, int nCmdshow)；
```

C-Runtime startup code 會呼叫你的 WinMain 函式，並傳入上述四個參數。從我們（程式員）的觀點來看，這就是程式開始執行的地方。如果你的 WinMain 函式如下：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstExePrev,
                    LPSTR lpszCmdLine, int nCmdshow) {

   return(0);
}
```

此函式沒有做任何動作，也沒有產生任何視窗，一開始執行就結束了。當 WinMain 函式結束，行程的主執行緒便會返回到 C-Runtime startup code，開始做善後清理的工作（也就是釋放一些資料及記憶體），接著 C-Runtime startup code 會呼叫 Win32 的 ExitProcess 函式，終止主執行緒。於是行程結束，而此行程的 4GB 位址空間內的程式碼和資料將完全被破壞。

這個行程完成了它的生命週期。當然，所有事情的發生僅在轉眼之間，沒有產生任何視窗，使用者也看不到任何視窗。這是一個毫無用處的程式。

在這一節即將結束之前，我想要說的是，一個行程就是一個執行個體（instance）。舉個例，你可以寫一個程式，檔名叫做 MyApp.EXE。假如你執行它七次，系統會爲它建立七個行程，每一個行程有自己獨立的 4GB 虛擬位址空間，其程式碼、資料、以及它所用到的動態聯結函式庫（DLLs），都將被載入到每一行程的虛擬位址空間中[2]。

系統會爲每一個行程指定一個 32 位元值。這個值稱爲行程識別碼（process ID），有些 Win32 API 會使用這個值做爲參數。爲了操控另一行程，執行緒必須獲得對方行程的識別碼，才能爲之。此外，每一執行緒亦擁有一個獨一無二的識別碼（thread ID）。

[2] 再一次，系統使用了一個較有效率的記憶體管理方法。系統實際上並未將七份 DLL 和 EXE 檔的內容全部載入到記憶體中，而是讓它們被共享。

視窗的顯示

對使用者來說，一個程式在螢幕上顯示一些訊息或視窗是非常重要的，它會提升使用者和程式彼此間的親和性與互動關係。

同樣地，對 Win32 程式員來說，顯示一個視窗也相當重要，然而描述一個「視窗類別」也刻不容緩，它用來定義視窗的外觀和視窗的行爲（通常回應一些像是使用者敲擊鍵盤或滑鼠移動等事件）。一個視窗事實上僅是一個視窗類別的實體（instance）。當程式員撰寫 Win32 程式時，他們會花較多的時間在「設計視窗類別」之上，反而花較少的時間去撰寫「產生視窗」的程式碼。

視窗產生之前，必須先定義其視窗類別。定義一個類別的過程稱爲註冊，註冊一個類別並不會產生任何視窗。然而，如果視窗類別已註冊完成，你便可利用此類別來產生視窗。通常一個程式僅僅只需註冊一個視窗類別。不過，假如你想要產生不同的視窗外觀和不同的視窗行爲，你也可註冊多個視窗類別。

視窗函式

毫無疑問，撰寫 Windows 程式最重要的關鍵便在於撰寫其視窗函式。你將花上很多很多的時間在這個工作上。視窗類別的視窗函式描述了應如何去顯現視窗及如何對事件做適當的回應 -- 亦即根據使用者不同的要求，來執行不同的處理動作。

視窗函式的設計相當簡單，其型式如下：

```
LRESULT WINAPI WndProc (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) ;
```

函式名稱不一定要叫做 WndProc，你可以依自己的意思取你想要的名稱。如果你在你的程式中註冊了許多個視窗類別，你可以爲每個視窗函式取不同的名稱。

每當視窗有任何事件發生時，系統便會呼叫其視窗函式來處理。函式的第一個參數 hwnd，表示視窗代碼（window handle）。記住，你可以單單使用一個視窗類別來產生許多個視窗，也就是說單單使用一個視窗函式來處理各個視窗實體（instance）的事件，這個視窗函式只須從視窗代碼便可判斷出該對哪一個視窗做動作[3]。

第二個參數是 uMsg，主要在識別視窗訊息的動作型式。微軟公司在 WinUser.H 表頭檔中定義了許多標準訊息，每一個訊息都以一個特定的數值來表示。表 **1.1** 列出一些標準的視窗訊息。

表 1.1 一些標準的視窗訊息

Windows 表頭檔中的識別字	數值	說明
WM_PAINT	0x0000000F	視窗須要做重繪動作
WM_CLOSE	0x00000010	使用者要求關閉視窗
WM_CHAR	0x00000102	使用者按下鍵盤上的一個字元
WM_MOUSEMOVE	0x00000200	使用者在視窗中移動滑鼠
WM_LBUTTONDOWN	0x00000201	使用者按下滑鼠左鍵

表 **1.1** 僅定義了五個標準的視窗訊息，事實上一個視窗所能處理的訊息有好幾百種。你並不須全部都要知道，也不要因此感到恐懼。通常，視窗函式只在乎少數幾個訊息，而忽略大部分訊息。舉個例，以下程式片段是一個典型的視窗函式的基本架構。

```
LRESULT WINAPI WndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
   LRESULT lResult = 0; // assume zero return

   switch (uMsg) {
      case WM_PAINT:
         // Paint the window's client area
```

[3] 對於 C++ 程式員來說，這個視窗代碼之於視窗類別，就相當於 C++ 類別成員函式中所隱藏的 this 參數一樣。

```
        ...
        break;

     case WM_CHAR:
        // Add a character to a buffer.
        ...
        break;

     default:
        // For all messages not explicitly handled,
        // do default processing.
        lResult = DefWindowProc(hwnd, uMsg, wParam, lParam);
        break;
  }
  return(lResult);
}
```

很明顯可以看出，以上的視窗函式只用到了數百種當中的二個視窗訊息：WM_PAINT 和 WM_CHAR，其它訊息並沒有被處理，皆交給 Win32 的 DefWindowProc 去處理。這個函式是作業系統的一部分，它是由一長串龐大的 switch case 敘述句構成，就像我們之前所看到的 WndProc 函式一樣，處理許許多多的視窗訊息。

舉個例，假如 DefWindowProc 函式要處理 WM_NCPAINT 訊息，當它接收到這個視窗訊息之後，會畫出所有 non-client area 區域的視窗元件（如標題欄、視窗邊框及功能表）。DefWindowProc 也處理 WM_SYSCOMMAND 訊息，使得任何來自視窗的系統功能表中的選項皆能被正確地執行。DefWindowProc 處理很多視窗訊息，但不是全部訊息；仍然有很多視窗訊息未被你的視窗函式及 DefWindowProc 函式處理，它們被全然地忽略了。

最後我所要討論的是視窗函式的最後兩個參數，wParam 和 lParam。這兩個參數都是 32 位元變數，它們實際表示的意義將依 uMsg 所代表的訊息不同而不同。舉個例，當一個視窗函式收到一個 WM_MOUSEMOVE 訊息，其 wParam 參數的意義將表示滑鼠的哪一個鍵被壓下，以及鍵盤上是否有控制鍵（Control key）或移位鍵（Shift key）被按住，而 lParam 所代表的則是滑鼠在 client area 的位置。如果視窗函式收到是 WM_CHAR 訊息，則 wParam 代表使用者按下的鍵的字元值，lParam 則爲一些旗標值。假如視窗函式收到一個 WM_PAINT 訊息，其 wParam 和 lParam 參數皆無意義。

設計視窗函式時，你必須非常有創造力。當你的視窗函式收到了一個視窗訊息，你可能會做以下之動作：

- 傳訊息給 DefWindowProc 函式。這個函式的詳細描述出現在 ***Microsoft Win32 Programmer's Reference*** 一書中。任何 DefWindowProc 函式無法辨認的訊息，將使得該函式傳回 0 值。
- 自行處理訊息。處理的形式通常藉助 case 敘述，或所謂的訊息剖析器（定義於 WindowsX.H 中）。任何動作都可以執行，每一訊息皆應有適當的傳回值。
- 自行處理訊息，並呼叫 DefWindowProc 函式。通常你會在視窗函式中使用 case 敘述來處理訊息，並明顯再去呼叫 DefWindowProc 函式。這表示要執行 DefWindowProc 中的動作，以及你自己加上去的額外動作。每一個訊息皆應有一個適當的傳回值，通常（但不一定）就是 DefWindowProc 的傳回值。
- 視窗函式當收到了一個未曾定義的訊息，所以忽略該訊息。但仍然應該傳回一個適當的值 (通常是 0)。

視窗函式的維護

當你第一次看到 switch 敘述出現的時候，想像一下如果你加入數以打計的 case 敘述來處理訊息，會出現什麼光景？是的，視窗函式會變得非常龐大。事實上，我曾看過綿延數千行的 switch 敘述句，令人眼花瞭亂。請不要用這種設計方式，因爲你的視窗函式將因此難以維護、修改、閱讀與了解。過去幾年，已經有些人想出了一些方法，幫助我們做維護的工作。在眾多的方法當中，有二種方法較引人注目：訊息剖析器（message crackers）和訊息映射表（message maps）。這兩種方法有一個共同特色，就是每一個視窗訊息皆有自己的處理函式。換言之，假如你想要處理 WM_SIZE 和 WM_CHAR 兩個訊息，你必須建立兩個函式，而這兩個函式通常會被命名爲 OnSize 和 OnChar。

訊息剖析器（Message Crackers）

訊息剖析器被大部分的 C 語言程式員採用，它靠著定義在 WindowsX.H 檔中的巨集（macros）來完成，你必須在含入 Windows.H 表頭檔之後，立即含入 WindowsX.H 檔。本書所有的視窗函式皆以訊息剖析器的方式來設計[4]。

下面是一個例子，教你如何以訊息剖析函式來處理 WM_SIZE 訊息：

```
void WndClassName_OnSize(HWND hwnd, UINT state, int cx, int cy) {
     ...
}
```

你可以在 WindowsX.H 表頭檔中找到 WndClassName_OnSize 函式原型，你可以利用 "WM_SIZE" 字串去搜尋。以下是一些有關於使用此一函式的注意事項。

- 由 WindowsX.H 檔中的定義可知，此函式並未傳回任何資訊（因爲它使用 void 宣告），這表示作業系統並不在乎 WM_SIZE 訊息的傳回值。
- 函式名稱之前皆有一前置詞 "WndClassName_"，這是因爲，我們的程式允許註冊多個視窗類別，爲使每一個視窗類別可以產生不同的動作來處理 WM_SIZE 訊息，因此 C 語言必須在每一個 OnSize 函式前加入不同的前置詞以便區分，以正確對映到其視窗類別。
- 所有訊息剖析函式的第一個參數都是視窗代碼，表示此函式目前處理的是哪一個視窗的訊息。回想一下，一個視窗類別可以產生出多個視窗，而其視窗函式所描述的行爲係針對依此一視窗類別所產生的所有視窗。也就是說，所有這些視窗在收到 WM_SIZE 訊息後的反應動作都是一樣的。視窗代碼也常常被其他處理視窗的動作所引用。

[4] 如你想知道有關訊息剖析器的更完整資料，請參考附錄A所提的 ***Advanced Windows***，Microsoft Press，1995。

- 其餘的參數並不是 wParam 和 lParam。這是一個主要的優點！早先，我曾提過 wParam 和 lParam 參數的意義將依訊息的不同而不同。假如你沒有使用訊息剖析器來替代一大堆 switch 敘述的話，你就必須自己去解譯這兩個參數的意義，極易出錯！然而，當你使用了訊息剖析器，這些巨集會在將訊息傳遞給訊息剖析函式之前，自動爲你解譯 wParam 和 lParam 兩個參數的意義。這個解譯動作即稱爲 “cracking”，這正是「訊息剖析器」名稱的由來。

說了這麼多，並沒有告訴你 WndClassName_OnSize 函式是如何地被呼叫。當你使用訊息剖析器，你仍然必須建立並維護你的視窗函式，函式看起來像這樣：

```
LRESULT WINAPI WndClassName_WndProc (HWND hwnd, UINT uMsg,
  WPARAM wParam, LPARAM lParam) {

  switch (uMsg) {
    HANDLE_MSG(hwnd, WM_SIZE,       WndClassName_OnSize);
    HANDLE_MSG(hwnd, WM_MOUSEMOVE,  WndClassName_OnMouseMove);
  }
  return(DefWindowProc(hwnd, uMsg, wParam, lParam));
}
```

當然，這函式是一個視窗類別函式，它的函式原型必須和所有視窗類別函式一樣。只不過現在我們以 HANDLE_MSG 巨集（定義在 WindowsX.H 中）取代一般所用的 switch - case 敘述。這個巨集有三個參數：第一個參數 hwnd 是視窗代碼，第二個參數指定你想要處理的訊息（如 WM_SIZE），第三個參數指定處理此一訊息的訊息函式名稱（例如 WndClassName_OnSize）。

這個 HANDLE_MSG 巨集中有一個 return 動作，傳回訊息剖析函式的傳回值。倘若訊息剖析函式宣告爲 void，則此傳回值爲 0。使用訊息剖析器時，大部份程式員所犯的錯誤是忘了在視窗函式的 switch 敘述中加入適當的 HANDLE_MSG 巨集；我自己也常常忘記。是的，我爲我想要處理的訊息撰寫訊息剖析函式，但難免會有所疏失，致使我所撰寫的訊息剖析函式未被呼叫，因爲我忘了在 switch 敘述中指定適當的巨集。**記住，不要忘記在你的視窗函式中加入你所要處理的訊息的訊息剖析函式。**

訊息映射機制（Message Maps）

C++ 程式員所使用的 Windows 類別庫，例如 Microsoft's Foundation Classes（MFC） 都是使用訊息映射（Message Maps）的方法來處理訊息。在微軟的 Visual C++ 整合環境中，你可以輕易利用 ClassWizard 對話盒（圖 **1.1**），在你的視窗函式中增加一個訊息處理函式。

在這個對話盒中，首先請於最上端選取一個視窗類別名稱，然後從列示清單（List Box）中選取你想要處理的訊息，加入到你的視窗類別函式中。只需按下【Add Function】鈕，Visual C++ 便會在你的程式碼中加入一個訊息映射函式。一個處理 WM_SIZE 訊息的函式如下：

```
void CAboutDlg::OnSize(UNIT nType, int cx, int cy)
{
     CDialog::OnSize(nType, cx, cy);

    // TODO: Add your message handler code here.

}
```

這個由 ClassWizard 所產生的函式看起來和之前所討論的訊息剖析函式的寫法大同小異。以下是我的一份簡短說明：

- 不管屬於那一個視窗類別，此函式都叫做 OnSize。當你使用 MFC 來撰寫程式，這一點辦得到，因爲 MFC 會爲每一個視窗類別產生一個 C++ 類別，而此訊息處理函式（如上述之 OnSize）會被放置在 C++ 類別的可存取範圍內（如上述之 CAboutDlg）。
- MFC 的訊息映射函式並沒有把視窗代碼當做參數來傳遞，這是因爲這些函式已經暗含了 this 參數，使其能辨別視窗。藉由這個參數，你在訊息映射函式中所使用的任何視窗處理函式，皆會對收到 WM_SIZE 的那個視窗做動作。這使你的程式碼更具可讀性，同時更易於管理。
- 和訊息剖析函式一樣，訊息映射機制也會解譯訊息的 wParam 和 lParam 參數，並傳遞給訊息映射函式。

- 由於 MFC framework 使用 C++ 繼承觀念來修改視窗類別的行爲，所以 ClassWizard 會爲你加上一個對基礎類別（base class）的訊息映射函式的呼叫動作。在稍早的範例中，ClassWizard 加上了一個明顯的 CDialog::OnSize 函式呼叫動作，這是因爲 CAboutDlg 類別是從 CDialog 類別衍生而來。

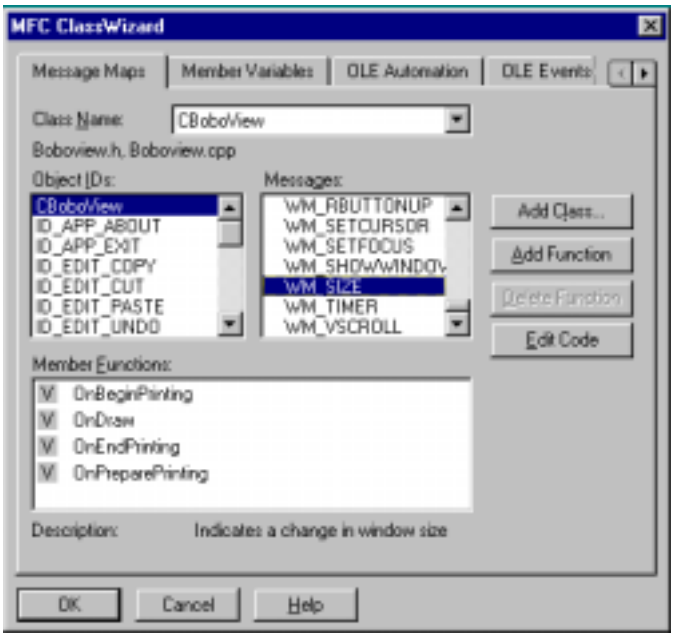


圖 1.1　Class Wizard 的交談窗

這裡我並沒有提到 MFC framework 如何去呼叫 CAboutDlg 函式，其中的處理方法比起訊息剖析器的方法（僅加入一行 HANDLE_MSG 到視窗函式中）還來得複雜許多。在 MFC 之中，CAboutDlg::OnSize 訊息函式必須首先被加入某個類別的定義（在 .H 檔）之中，然後必須有一個訊息項目（entry）被加到類別的訊息映射表（message map table）中（定義於 CPP 檔）。幸運的是 ClassWizard 會自動幫你做好這些瑣碎的工作[5]。

[5] 有關 MFC 和視窗函式的更多相關資訊，請參閱 Visual C++ 技術文件。

我不打算在本書對 MFC 做更進一步的討論，我想這些簡單的介紹已經足以使你對 MFC 和 Visual C++ 帶來的利益有所瞭解。本書中我將使用定義於 WindowsX.H 表頭檔中的訊息剖析器，來撰寫各個程式。

準備註冊一個視窗類別

你的程式一旦被啓動，首先必須初始化一個 WNDCLASSEX 結構，並在產生任何視窗之前，利用 RegisterClassEx 函式來註冊此視窗類別。這個 WNDCLASSEX 的結構定義如下：

```
typedef struct _WNDCLASSEX {
   UNIT         cbSize;
   UNIT         style;
   WNDPROC      lpfnWndProc;
   int          cbClsExtra;
   int          CbWndExtra;
   HINSTANCE    hInstance;
   HICON        hIcon;
   HCURSOR      hCursor;
   HBRUSH       hbrBackground;
   LPCTSTR      lpszMenuName;
   LPCTSTR      lpszClassName;
   HICON        hIconSm;
} WNCLASSEX;
```

結構中的資料成員分別描述了視窗類別的屬性。表 **1.2** 簡單地說明了每一成員的意義。

表 1.2　WNDCLASSEX 結構中的資料成員

資料成員	描述
cbSize	表示 WNDCLASSEX 的結構大小。通常使用 sizeof(WNDCLASSEX) 即可得知。
style	視窗類別的型式。所有類別型式的識別字的前頭都有一前置詞 CS_（例如 CS_VREDRAW），這主要是爲了不和視窗型式的前置詞 WS_ 混淆。
lpfnWndProc	視窗類別中的視窗函式（之前討論過）的記憶體位址（在行程的 4GB 虛擬位址空間中）。
cbClsExtra	告訴系統當註冊此一視窗類別時，需保留多少額外的位元組。本章稍後的「視窗類別的屬性」一節中將進一步討論之。
cbWndExtra	告訴系統當以此視窗類別產生一個視窗時，需保留多少額外的位元組。本章稍後的「視窗的屬性」一節中將進一步討論之。
hInstance	正在登錄視窗類別之 EXE 或 DLL 的執行個體（instance）代碼。對 EXE 來說，這應該就是 WinMain 函式所獲得的第一個參數。對 DLL 來說，這應該是 DllMain 函式所獲得的第一個參數。本章稍後的「視窗類別的權限範圍」一節中有更詳細的討論。
hIcon	視窗的圖示（icon）代碼。這個圖示通常內含一張 32x32 pixels 的圖，和一張 16x16 pixels 的圖。
hCursor	系統在視窗工作區（client area）中所顯示的游標的代碼。
hbrBackground	系統欲用以填充視窗工作區（client area）的畫刷的代碼。
lpszMenuName	視窗類別的選單。應該是一個以 NULL 爲結束符號的字串，標示出 RC 檔中的一套選單。所有由此類別所製造出來的視窗，都應該使用此套選單。
lpszClassName	視窗類別的名稱。它應該是一個以 NULL 爲結束符號的字串。
hIconSm	視窗的小圖示代碼。這個圖示顯示在系統工作列（Taskbar）及視窗的左上角。此欄常被設爲 NULL，於是系統便以 hIcon 所指定的 16x16 圖示來替代。如果此欄所指定的是 32x32 圖示，系統會將之縮小成 16x16 圖示。假如你想要顯示一個與 hIcon 所指定的圖示不一樣的小圖示，你就必須設定此一欄位。

WNDCLASSEX 結構中最重要的欄位是 lpszClassName，它用來表示視窗類別的名稱。每當你註冊一個視窗類別，你必須爲類別指定一個名稱，也就是以 lpszClassName 指定一

個以 '\0' 結尾的字串。然後你才能夠使用這個名稱，呼叫 CreateWindowEx 函式（稍後介紹）來產生一個或多個視窗。這個函式會與所有已註冊的視窗類別名稱作比對，假如比對成功，就會產生出此一視窗類別的一個實體（instance），並且傳回其視窗代碼。如果比對失敗，CreateWindowEx 函式會傳回 NULL。

註冊一個視窗類別

一旦設定好 WNDCLASSEX 結構之後，你必須註冊此視窗類別，讓作業系統知道它的存在。這可藉由 RegisterClassEx 函式來完成：

```
ATOM RegisterClassEx(CONST WNDCLASSEX *pwc);
```

通常，程式初始化一個 WNDCLASSEX 結構後，不久便會去呼叫 RegisterClassEx 函式。這個 RegisterClassEx 函式會傳回一個用來辨識視窗類別名稱（在 lpszClassName 欄位中被指定的名稱）的 atom 號碼。如果這個視窗類別無法成功註冊，RegisterClassEx 函式會傳回 INVALID_ATOM（定義爲 0）。

以下就是註冊一個視窗類別的詳細步驟。

```
LRESULT WINAPI MyWndClass_WndProc(HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam){

   // This is the window class's window procedure
   // discussed in the previous section.
   .
   .
   .
}

int WINAPI WinMain(HINSTANCE hinstExe, HINSTANCE hinstExePrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   WNDCLASSEX wc;
   ATOM atomClassName;
```

```
// Initialize all members in the structure to 0 (zero).
// NOTE: Other sample source code presented throughout this book
// will use the adgINITSTRUCT macro (discussed in Appendix A) to
// initialize a data structure for use.
ZeroMemory(&wc, sizeof(wc));

// Set the size of the WNDCLASSEX structure.
wc.cbSize = sizeof(wc);

// Give the window class a name.
wc.lpszClassName = "My Window Class Name";

// Tell the system the address of this class's window procedure.
wc.hInstance = hinstExe;  // First parameter to WinMain.

// NOTE: All other members can remain zero but are usually
// initialized as shown later.

// Allow the window to receive mouse double-click messages.
wc.style = CS_DBLCLK;

// This window class requires no additional class information.
wc.cbClsExtra = 0;

// Window instances created of this class require no additional information.
wc.cbWndExtra = 0;

// Tell the system which icon should be displayed when the mouse
// needs to show a large icon to represent windows of this class.
// IDI_APP identifies an icon in the EXE file's resource section.
wc.hIcon = LoadIcon(hinstExe, IDI_APP);

// Tell the system which cursor should be displayed when the mouse
// is positioned over a window's client area.  IDC_ARROW identifies
// a standard arrow cursor offered by the system.
wc.hCursor = LoadCursor(NULL, IDC_ARROW);

// Windows of this class should have their background painted using the
// color selected by the user in the Desktop. Properties dialog box.
wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);

// Tell the system what menu template should be used when creating a
// menu for each window instance of this class.
// IDM_APP identifies a menu template in the EXE file's resource section.
wc.lpszMenuName = MAKEINTRESOURCE(IDM_APP);
```

```
   // Tell the system which icon should be displayed when the system
   // needs to show a small icon to represent windows of this class.
   // The IDI_APP icon contains a 16 x 16 and a 32 x 32 pixel image.
   wc.hIconSm = wc.hIcon;

   // Register the window class with the operating system.
   atomClassName = RegisterClassEx(&wc);

   if (atomClassName == INVALID_ATOM) {

      // The window class could not be registered. Call GetLastError
      // to determine why. Usually it's because a class with the same
      // name has already been registered.
   } else {

      // The window class was successfully registered. Windows can now
      // be created from this class.
   }

   // The remainder of the application's code goes here.
   .
   .
   .
}
```

一旦視窗類別註冊成功，程式可以根據此視窗類別，利用 CreateWindowEx 函式產生一個（或多個）視窗。

RegisterClassEx 做些什麼事

當你呼叫 RegisterClassEx 函式，它會執行以下動作：

1. 檢查類別名稱。假如類別名稱已被同一個 EXE 或 DLL 註冊過，則 RegisterClass 傳回 INVALID_ATOM。
2. 如果此類別的型態被指定爲 CS_GLOBALCLASS，而當系統檢查類別名稱時，發現此名稱已被此一行程位址空間中的任何 EXE 或 DLL 註冊過，那麼 RegisterClassEx 會傳回 INVALID_ATOM。關於 CS_GLOBALCLASS，本章稍後的「視窗類別的權限範圍」中有更多討論。
3. 配置一塊記憶體，準備用以儲存視窗類別的內容。 這塊記憶體必須足以存放

WNDCLASSEX 結構的所有資訊，再加上由 cbClsExtra 欄位所指定的位元組個數。

4. 將 WNDCLASSEX 結構內容複製到記憶體中，並設定其額外位元組內容爲 0。這些額外的位元組將會保留給類別的視窗函式使用，作業系統不會用到它。這些額外的位元組所儲存的資訊可被每一個據此類別而產生出來的視窗共享，這些額外位元組的管理完全由程式員來控制。

5. 配置一塊記憶體用以儲存類別選單的名稱。lpszMenuName 欄位將被系統改變爲指向此一記憶體區塊的指標。

6. 將類別名稱加進系統的 local atom table[6] 中，並將此 atom 值儲存到視窗類別的記憶體區塊中。這個 atom 值可藉由 GetClassWord 函式（並以 GCW_ATOM 爲參數）得到。GCW_ATOM 將在稍後討論。

7. RegisterClassEx 函式的傳回值即是上述所說的 atom 值。一旦知道了 atom 值，便能在程式中使用它。此 atom 值代表類別的名稱。舉個例，要爲一個註冊好的視窗類別產生一個視窗實體，程式動作如下：

```
// Create and initialize the members in the wc structure.
WNDCLASSEX wc;
.
.
.
// Register the window class with the operating system.
ATOM atomClassName = RegisterClassEx(&WndClass);

if (atomClassName != INVALID_ATOM) {

    // Create a window instance based on the window class.
    HWND hwnd = CreateWindowEx(0, MAKEINTATOM(atomClassName), NULL,
       WS_CHILD,...);
}
.
.
```

[6] 在 Windows 作業系統中，有一個 global atom table，可被所有行程共享。此外每一個行程各自擁有一個 local atom table。系統也擁有一個 local atom table，僅供系統使用 -- 這意味一個行程不能使用那些公開的 atom 相關函式來存取 system's local atom table。

視窗類別的屬性

WIN32 API 的某些函式允許你取得和修改一個已註冊的視窗類別資訊。這一節中我將描述這些函式。

GetClassInfoEx 函式能夠取得已註冊視窗類別的 WNDCLASSEX 結構：

```
BOOL GetClassInfoEx(HINSTANCE hinst, LPCSTR lpszClassName, PWNDCLASSEX pwc);
```

函式的第一個參數 hinst 用來識別是誰註冊這個視窗類別(可能是 EXE 也可能是 DLL)。如果你試圖取得 Windows 的 system global classes（稍後詳論）的任何資訊，hinst 應該放置 NULL。第二個參數 lpszClassName，若不是類別名稱（一個以 '\0' 結尾的字串）就是類別名稱的 atom 值（可藉由 MAKEINTATOM 巨集獲得）。最後一個參數 pwc，是指向 WNDCLASSEX 結構的指標，該結構內必須已經填妥一些必要的資訊，你應該在呼叫 GetClassInfoEx 之前先將結構中的 cbSize 欄位設爲 sizeof(WNDCLASSEX)。

使用 GetClassInfoEx，你應該知道四件事情：

- 當你註冊一個視窗類別，系統內部爲你配置的視窗類別資料結構中，並不儲存著類別名稱，而是儲存其 atom 值。所以 GetClassInfoEx 函式在回返之前，必須先將 WNDCLASSEX 的 lpszClassName 欄位設爲一個字串 -- 也就是你傳給 GetClassInfoEx 的第二個參數。目前尚未有任何公開函式，能由一個 atom 值得到其對應的類別名稱（字串）。函式 GetAtomName 和 GlobalGetAtomName 沒有辦法派上用場，因爲上述這個 atom 值既不是行程的區域變數，也不是系統的全域變數，而是系統的區域變數[7]。

[7] 在註 6 中我曾說過，系統的 local atom table 不能被一個 atom 函式(不管是區域性或全域性)存取。然而，你卻可以使用一個公開函式 GetClipboardFormatName 來存取 system's local atom table：

```
int GetClipboardFormatName(UINT uFormat, LPTSTR lpszFormatName, int cchFormatName);
```

此函式會針對一個 system-local atom 傳回一個字串。你也可以利用這個函式將一個 registered window message atom（由 RegisterWindowMessage 函式傳回，稍後討論）轉換爲一個字串。

- 當你註冊一個視窗類別，系統內部會複製此類別的選單名稱。當 GetClassInfoEx 函式回返，lpszMenuName 將被改變爲「指向系統內部選單名稱」的字串指標。這是作業系統的一大缺失！你不應該直接存取一個作業系統的字串，也不應該在這記憶體位址上做任何寫入動作。
- GetClassInfoEx 會以傳進來的 hinst 參數做爲 WNDCLASSEX 結構中的 hInstance 設定值。假如你要這個值，可以呼叫 GetClassLong(hwnd, GCL_HMODULE) [8]。
- GetClassInfoEx 僅能在呼叫此函式的那個行程中取得一個已註冊的視窗類別的相關資訊。舉個例，如果你想利用 GetClassInfoEx 取得 “NOTEPAD” 類別的相關資訊，會導致失敗，因爲呼叫 GetClassInfoEx 函式的人（行程）並不是 NOTEPAD。

GetClassInfoEx 函式將 WNDCLASSEX 結構填滿後，會傳回視窗類別的 atom 值（縱使這函式的原型被宣告爲 BOOL）。不,必說，GetClassInfoEx 沒辦法傳回類別的額外位元組（extra bytes）的內容。Win32 並不提供與 GetClassInfoEx 相反的函式，讓你以一次函式動作就設定整個視窗類別的資訊。GetClassInfoEx 與我稍後即將討論的函式們有什麼區別呢？最大的區別就是它不需要視窗代碼，就能夠存取視窗類別的資料。

GetClassWord、GetClassLong、SetClassWord 和 SetClassLong 函式允許你取得和設定一個已註冊視窗類別的類別屬性（欄位內容）：

```
DWORD     GetClassLong(HWND hwnd, int nIndex);
WORD      GetClassWord(HWND hwnd, int nIndex);
DWORD     SetClassLong(HWND hwnd, int nIndex, LONG dwNewLong);
WORD      SetClassWord(HWND hwnd, int nIndex, WORD wNewWord);
```

要取得或改變視窗類別的任何一個屬性（欄位），你必須提供處理對象（一個視窗）的

[8] 在 16 位元視窗環境下，HMODULE 和 HINSTANCE 代表兩個不同的東西。但在 Win32 中，HMODULE 和 HINSTANCE 是相同的，可以交替使用。

視窗代碼，並指定一個屬性識別字。你必須以視窗代碼做爲第一個參數，傳遞給前述的四個函式。第二個參數 nIndex 用以指定你想要取得或改變的屬性。表 **1.3** 列出了類別屬性，以及上述四個函式的第二參數會用到的識別字。每一個識別字都有一個前置詞，不是 GCW_ 就是 GCL_。底線之前的 W 或 L 分別表示所要取得的值是一個 16 位元值（WORD）或一個 32 位元值（LONG）。

表 1.3　類別的屬性和它們的識別字

類別屬性	Windows.H 識別字	屬性可被更改嗎？
Atom	GCW_ATOM	No
Style	GCL_STYLE	Yes
Window procedure address	GCL_WNDPROC	Yes
Class extra bytes	GCL_CBCLSEXTRA	No
Window extra bytes	GCL_CBWNDEXTRA	Yes
Registrant's HINSTANCE	GCL_HMODULE[9]	No
Icon	GCL_HICON	Yes[10]
Small icon	GCL_HICONSM	Yes
Cursor	GCL_HCURSOR	Yes
Background brush	GCL_HBRBACKGROUND	Yes
Menu name	GCL_MENUNAME	No

[9] 在 16 位元視窗環境下，HMODULE 和 HINSTANCE 有點不同。然而在 Win32 環境中它們完全相同。爲了讓開發人員容易將現有的 16 位元程式碼移植到 Win32 去，微軟並沒有以一個新的 GCL_HINSTANCE 識別字來取代 GCL_MODULE。但是你要知道，當你要求一個視窗類別的 HMODULE 時，你所取得的是它的 HINSTANCE。

[10] 當你改變視窗類別的大圖示和小圖示，現已存在的視窗並不會立即受到影響。如果你要改變現存視窗的圖示，可利用 WM_SETICON 和 WM_GETICON 兩訊息。

表 **1.3** 的最後一個欄位表示，你是否能夠使用 SetClassWord 或 SetClassLong 來改變類別屬性。舉個例子，下面這樣的呼叫：

```
SetClassLong(hwnd, GCL_HMODULE, hInstance);
```

其實可以這樣的呼叫取代之：

```
GetClassLong(hwnd, GCL_HMODULE);
```

這是因爲，SetClassLong 將傳回目前類別的 HMODULE，而不會改變其值。

修改視窗類別結構的最常見理由，是爲了在產生新視窗之前，改變其視窗屬性。不幸的是，SetClassLong 和 SetClassWord 都需要傳入一個你想修改的類別的視窗代碼，而這是非常不方便的（侯俊傑註：因爲視窗通常尚未產生）。Win32 API 對於這個問題並沒有提供適當的解決方法。如果你想改變一視窗類別的屬性，你必須先產生此類別的視窗，再呼叫 SetClassLong 或 SetClassWord 函式改變之，然後殺掉此視窗，然後再依修改過的類別產生新的視窗。

下面這個例子告訴你如何在你的行程中改變所有編輯視窗（侯俊傑註：Windows 內建的 edit control）的滑鼠游標，以箭頭游標來替代 I 字型游標：

```
// Create a window instance of the edit window class.
HWND hwndEdit = CreateWindowEx(0, "edit", "",WS_OVERLAPPED,
  0, 0, 0, 0, NULL, NULL, hInstance, NULL);

// Change the cursor handle to that of the up arrow.
SetClassLong(hwndEdit, GCL_HCURSOR, (LONG) LoadCursor(NULL, IDC_UPARROW));

// Destroy the edit window.
DestroyWindow(hwndEdit);

// From this point on, all edit windows will display an up arrow instead
// of an I-beam mouse cursor when the mouse enters their client area.
```

由於系統總是會在顯示游標之前先去看看視窗類別的屬性，所以執行過上述程式碼之後，再產生的編輯視窗，將擁有箭頭游標而非 I-beam 游標。注意，即使編輯視窗的類別是屬於 system global class，每一行程仍需註冊屬於它自己的編輯視窗類別。這意味上述

的程式片段僅僅改變此一行程內的編輯視窗的游標，其他行程的編輯視窗類別並不受影響。

當你註冊一個視窗類別，你可以自己建立一些額外資訊。只要在呼叫 RegisterClassEx 函式之前先設定好 WNDCLASSEX 結構中的 cbClsExtra 欄位即可（用以告訴系統說，你需要多少個額外位元組）。系統從來不會直接存取這些位元組（當類別被註冊後，系統通常將額外位元組的內容設爲 0）。你可以藉由呼叫 GetClassWord、GetClassLong、SetClassWord 或 SetClassLong，並傳入一索引值給 nIndex 參數，於是取得或設定這些額外位元組。舉個例，以下程式碼會將系統時間（system time）設定在類別的額外位元組的前四個位元組中：

```
SetClassLong(hwnd, 0, GetTickCount());
```

第一個參數 hwnd 表示視窗代碼。第二個參數 0 表示視窗類別的額外位元組的偏移量（offset），本例應該放的是一個長整數[11]。注意，這裡表示的是一個 byte offset，而不是一個 long offset。所以，如果你要在一個視窗類別的額外位元組中儲存兩個長整數值的話，我們先得設定 WNDCLASSEX 的 cbClsExtra 爲 8（也就是 2*sizeof(long)），然後再以下面動作存取第二個長整數的值：

```
SetClassLong(hwnd, 4, lSomeValue);
```

第三個參數表示我們要存入的值。由於處理這些動作相當繁複且容易出錯，所以我寫了一個巨集來簡化這些工作。我將這個巨集的原始碼放在本書附錄 A 中，並且使用於各章的範例程式中。

---

[11] 附帶一提的是，預先定義好的索引值如 GCW_ATOM, GCL_STYLE, GCL_WNDPROC, GCL_CBCLSEXTRA, GCL_CBWNDEXTRA, GCL_HMODULE, GCL_HICON, GCL_HICONSM, GCL_HCURSOR, GCL_HBACKGROUND 和 GCL_MENUNAME 都是負值，不會與你自設的索引值發生衝突。

視窗類別的權限範圍

在一個系統之中，許多程式註冊視窗類別，許多 DLLs 也註冊視窗類別，於是系統如何規畫組織這些視窗類別，成爲一件重要的大事。在 16 位元視窗環境下，我們可以在一個程式中註冊視窗類別，而在另一個程式中產生此類別的實體（也就是視窗）。有時候，程式無法註冊某個視窗類別，原因是已有另一個程式註冊了同名的視窗類別。在一個良好的作業系統中，這種情況不應存在，因爲一個行程不應該影響另一個行程，也不應該受另一個行程的影響。

Win32 環境中，上述情況不會發生，因爲 Win32 要求每一個行程對其欲產生的視窗，註冊對應的視窗類別。一個程式不能替另一個程式註冊視窗類別，也不能產生一個「根據其他程式所註冊之視窗類別」而產生的視窗實體。

註冊好的視窗類別只能在一個行程中使用，不能跨行程使用。目前，在每一個行程中，視窗類別可能有三種不同的類型：module local、process global 和 system global。

Module Local 視窗類別

EXE 或 DLL 都可以註冊 “Module Local” 視窗類別，並且僅供自己使用。舉個例，如果一個程式內含以下程式碼：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstExePrev,
                    LPSTR lpszCmdLine, int nCmdShow) {

  WNDCLASSEX wc;
  adgINITSTRUCT(wc, TRUE);

  wc.lpszClassName = "Exe-registered WndClass";
  wc.hInstance = hInstance;  // EXE is the registrant.
  // Set wc's remaining members
  .
  .
  .
  // Register the class
  RegisterClassEx(&wc);
  .
```

```
   .
   .
   return(0);
}
```

然後下列程式碼在一個 DLL 中執行：

```
HWND DllCreateAnExeWindow () {

     return(CreateWindowEx(0, "Exe-registered WndClass", ..., g_hinstDll, ...));
}
```

DllCreateAnExeWindow 永遠不會成功，它會傳回 NULL。原因是，這個視窗類別係被 EXE 註冊，其 HINSTANCE 是 WinMain 的第一個參數。當 DllCreateAnExeWindow 函式被呼叫，它收到正確的視窗類別名稱，"Exe-registered WndClass"，然而它收到的 DLL HINSTANCE 是被放在一個全域變數 g_hinstDll 中。當你呼叫 CreateWindowEx 時，系統不僅比較了視窗類別的名稱，也會將「視窗類別註冊者」的 HINSTANCE 值和「視窗實體產生者」的 HINSTANCE 值拿來比較。這兩個值必須匹配，否則系統不會答應你的要求（產生視窗）。

所以，問問你自己，是否有什麼方法可以在一個 DLL 中產生「由 EXE 註冊的視窗類別」的視窗實體，或是有什麼方法可以在一個 EXE 中產生「由 DLL 註冊的視窗類別」的視窗實體。答案是 yes！事實上，有兩種方法可以做到。第一種方法比較簡單：設法讓 HINSTANCE 的值匹配。舉個例，讓我們重寫 DllCreateAnExeWindow 如下：

```
HWND DllCreateAnExeWindow () {

   // Get the EXE's HINSTANCE value
   HINSTANCE hinstExe = GetModuleHandle(NULL);

   return(CreateWindowEx(0, "Exe-registered WndClass", ..., hinstExe, ...));
}
```

請注意我所呼叫的 GetModuleHandle 及其參數 NULL。此呼叫促使 GetModuleHandle 傳回行程位址空間中的這個 EXE 的 HINSTANCE。如果 CreateWindowEx 被呼叫，這 EXE 的 HINSTANCE 已被取得並放置於 hinstExe 變數中，所以視窗可以順利被產生出來。

不幸的是，如果我們在一個 DLL 中註冊視窗類別，並且期望在一個 EXE 或另一個 DLL 中產生該類別的視窗實體，上述方法並不適用。爲了解決這個問題我們必須取得註冊者的 HINSTANCE，並以其值做爲參數傳遞之，使其它的 EXEs 或 DLLs 能產生視窗實體。我相信你一定認爲這種作法太不方便了，應該有更好的解決方案才行。嗯，我們很幸運，解決方法是利用“Process Global”視窗類別。

Process Global 視窗類別

所謂“Process Global”視窗類別是，它可被一個行程位址空間中的 EXE 以及任何 DLLs 使用。通常是在一個 DLL 內含一些有用的視窗函式時派上用場。當這個 DLL 被載入行程位址空間時，DllMain 函式會被呼叫，我們讓它爲 DLL 所提供的每一個視窗類別呼叫 RegisterClassEx 函式。之後，此程式（或其它 DLLs）便可以呼叫 CreateWindowEx 函式來產生這些視窗類別的實體（也就是視窗）。

有一個很好的例子可以用來解釋這個方法，那就是 Windows 95 和 Windows NT 3.51 所提供的 new common controls。微軟公司已經建立了一個巨大且極爲有用的視窗類別集合，像是 up/down control、tree listbox、status box 和 toolbox。這些視窗類別的視窗函式包含在一個 DLL 檔中，名爲 ComCtl32.DLL，此檔由作業系統提供。預設情況下，程式並不能產生這些視窗類別的實體，因爲 Win32 的每一個行程都必須爲其所使用到類別做註冊動作。爲了使用這些類別，這個 DLL 必須被載入到行程的位址空間中[12]。載入之後，此一 DLL 即爲可能用到的每一個視窗類別呼叫 RegisterClassEx 函式。

現在，程式只要簡單地呼叫 CreateWindowEx 並傳入一個 EXE's HINSTANCE，便可產生上述視窗類別的實體。爲什麼可以這樣呢？因爲在呼叫 RegisterClassEx 函式之前，

---

[12] 一個程式會藉由呼叫 LoadLibrary，或呼叫由 ComCtl32.DLL 提供的 InitCommonControls，將所用到的 DLLs 載入到行程位址空間中：

```
VOID WINAPI InitCommonControls(VOID);
```

ComCtl32.DLL 會將它的每一個視窗類別的 WNDCLASSEX 結構中的 CS_GLOBALCLASS 屬性設爲 on。當系統看到視窗類別是以 CS_GLOBALCLASS 型態來註冊，便會以 "Process Global" 視窗類別來看待此視窗類別，而不是以 "Module Local" 視窗類別待之。事實上，CS_GLOBALCLASS 的存在也就僅僅是用來區分 "Process Global" 和 "Module Local" 兩種視窗類別的不同而已。

其實啊，CS_GLOBALCLASS 爲 on，就是告訴系統說類別註冊者的 HINSTANCE 值和視窗產生者的 HINSTANCE 值不需要做匹配，只要比較類別名稱即可。於是 EXE 和 DLL 都可以順利產生 "Process Global" 視窗類別（如 new common controls）的視窗實體。

System Global 視窗類別

當一個行程（process）被產生出來，作業系統會爲它註冊 "System Global" 視窗類別。使用者執行一個程式（例如 Notepad），系統會爲此行程產生一個 4GB 位址空間，並產生此行程的主執行緒，然後系統會自動註冊一些馬上可以被此行程使用的視窗類別。程式碼僅需簡單呼叫 CreateWindowEx，即可爲每一個 "system Global" 視窗類別[13]產生一個或多個視窗實體。

欲使用這些 "System Global" 視窗類別，可呼叫 CreateWindowEx，並將你想要產生的視窗類別的名稱傳入。以下列出一些比較一般性的 "System Global" 視窗類別，你可以直接在你的程式中使用：

```
button, combobox, edit, listbox, mdiclient, scrollbar, static
```

[13] 事實上，Windows 95 會註冊一整組 "System Global" 視窗類別。如果有一個應用程式對這些視窗類別的屬性做了改變，系統會偵測到這些改變，並且以相同的類別名稱註冊一個 "Process Global" 視窗類別。這個動作對程式員來說是透明的。由於很少有程式會去修改 "System Global" 視窗類別的屬性，所以註冊一個新類別是很少見的。也就是說，系統會有效運用記憶體資源，除非絕對必要，每當產生一個行程時，並不會爲它註冊一些新的視窗類別。

但是在 Windows NT 環境下，每當一個行程被產生出來，系統真的會去註冊一組新的視窗類別。

雖然這些類別時常被用作子視窗控制元件，但它們和你自己註冊的 “Process Global” 視窗類別並沒有不同，唯一的差別就是每一個行程都可以使用之。

系統也會爲行程註冊幾個私下使用的視窗類別，一旦系統需要，便自動產生出這些視窗類別的視窗實體。舉個例，當你呼叫 DialogBox 或 CreateDialog，系統會產生「對話盒類別」的實體（本書第二章對這些類別有更詳細說明）；你不需要呼叫 CreateWindowEx 並將對話盒類別的名稱傳入，DialogBox 和 CreateDialog 內部會自行呼叫 CreateWindowEx 來產生視窗。

註銷（Unregistering）一個視窗類別

雖然不常使用，但你還是可以呼叫 UnregisterClass 去註銷一個視窗類別：

```
BOOL UnregisterClass(LPCTSTR lpszClassName, HINSTANCE hInstance);
```

這個函式有兩個參數：lpszClassName 代表你想註銷的類別名稱（也可以使用 MAKEINTATOM 巨集所產生的一個 atom 值取代之），hInstance 則用以識別此一類別的註冊者（一個 EXE 或 DLL）。

UnregisterClass 函式首先確定此視窗類別是否存在；如果存在，便檢查有沒有存在此類別的任何視窗實體。如果類別存在而且沒有它的任何視窗實體，UnregisterClass 便會註銷此視窗類別，此後再不可能利用此類別去產生任何視窗實體。如果兩個條件沒有全部成立的話，UnregisterClass 便會失敗，傳回 FALSE。

注意，一個 DLL 可以被載入行程的位址空間中，此時它可能已經註冊了一些 “Module Local” 或 “Process Global” 視窗類別。此 DLL 也可能從行程的位址空間中被卸載（unload）-- 通常是藉由 FreeLibrary。如果這樣的事情發生，可能會有一些已產生視窗或即將產生視窗的視窗函式，不再存在於行程的位址空間中。企圖使用那些類別的視窗，會導致存取失敗（access violation）。如果系統能時時檢查，看看一個註冊了某些視窗類別的 DLL 是否已被卸載（unload），然後系統能夠自動註銷該 DLL 註冊的所有視窗類別，那便非常理想。然而這對作業系統而言是件很困難的工作，你必須自己檢查你的視

窗所需要用到的 DLL 是否還存在於行程位址空間中。

雖然，當一個 DLL 從行程位址空間中卸載（unload）時，系統不能夠自動註銷相關的視窗類別，但是當行程結束時，系統倒是能夠自動註銷行程的所有視窗類別。事實上，由於作業系統有此性質，所以大部分程式設計者從不使用 UnregisterClass 函式。

產生視窗

一旦視窗類別被註冊了，你便能夠藉由呼叫 CreateWindowEx，傳入視窗類別名稱（或是以 MAKEINTATOM 巨集所取得的一個類別名稱的 atom 值）作爲第二個參數（lpszClassName），產生一個或多個視窗實體：

```
HWND CreateWindowEx(DWORD dwExStyle, LPCTSTR lpszClassName,
   LPCTSTR lpWindowName, DWORD dwStyle,
   int X, int Y, int nWidth, int nHeight,
   HWND hWndParent, HMENU hMenu, HINSTANCE hInstance, LPVOID lpParam);
```

這個函式會送出數個訊息給對應的視窗類別的視窗函式。這些訊息列於表 **1.4** 中。雖然訊息會依表中所列之順序送出，但此順序並不保證永遠不變，微軟可能會在未來插入一些額外訊息。在撰寫視窗函式時，你應該盡可能讓你的訊息處理常式（message handler）獨立，無法再細分；也就是說，每一個訊息處理常式不應依賴其他的訊息的處理結果。

表 1.4 CreateWindowEx 所送出的訊息

視窗訊息	說明
WM_GETMINMAXINFO	CreateWindowEx 送出此訊息的目的是爲了幫助決定視窗的初始大小[14]。大部份視窗函式之所以要處理這個訊息，是因爲它們需要調整視窗的極小化或極大化。
WM_NCCREATE	此訊息被送出是爲了讓 DefWindowProc 能爲視窗執行一些初始化動作。大部份視窗函式皆會忽視此訊息，並將它交給 DefWindowProc 處理。
WM_NCCALCSIZE	CreateWindowEx 送出此訊息是爲了計算視窗的 client-area 大小和位置。大部份視窗函式皆忽視此一訊息，並將它交給 DefWindowProc 處理。
WM_CREATE	此訊息被送出是爲了讓視窗函式處理一些初始化動作，通常是爲了初始化一個視窗的狀態，或者產生子視窗。視窗函式可以回返值 - 1 表示初始化失敗，於是視窗便會被摧毀，於是 CreateWindowEx 傳回 NULL 給呼叫者。
WM_SIZE	這個訊息被送出是爲了將視窗的初始大小告知視窗函式。有些視窗函式會處理此一訊息，爲的是重新安排視窗的 client area 的內容。
WM_MOVE	這個訊息被送出是爲了將視窗起始位置告知視窗函式。大部份視窗函式皆忽視此一訊息，並將它交給 DefWindowProc 去處理。

[14] 事實上，這個 WM_GETMINMAXINFO 訊息最先被傳送給視窗，竟成了作業系統的一個蟲(bug)，但是爲了與過去相容，所以這個蟲未曾被修復。每回視窗的大小被改變時，系統皆會送出 WM_GETMINIMAXINFO 訊息給視窗函式。爲什麼先送出 WM_GETMINMAXINFO 訊息給視窗函式會是一個蟲呢？其原因是因爲視窗尚未被完全地初始化（藉著處理 WM_NCCREATE 和 WM_CREATE 訊息）。所以，當一個視窗函式最先收到 WM_GETMINIMAXINFO 訊息可能尚未有足夠的資訊來正確地處理這個訊息。通常，處理這訊息必須包含一小段程式碼來檢查視窗是否已完全地被初始化。如果視窗尚未被初始化，則 WM_GETMINMAXINFO 訊息的其它處理動作將會被略過。

以下程式片段告訴你，如果視窗初始化失敗，應該如何放棄這個視窗的產生過程：

```
#define GCL_DATA 0
.
.
.
LRESULT lResult = 0;
.
.
.
case WM_CREATE:
   pData = HeapAlloc(GetProcessHeap(), 0, BUFFERSIZE);
   if (pData == NULL)
      lResult = -1;   // Halt creation of window
   else
      SetClassLong(hwnd, GCL_DATA, (LONG) pData);
      // lResult initialized to 0, window creation OK.
   }
   break;

case WM_DESTROY:
   // Free block of memory created during WM_CREATE message.
   pData = GetClassLong(hwnd, GCL_DATA);
   if (pData != NULL)
      HeapFree(GetProcessHeap(), 0, pData);
   break;
.
.
.
return(lResult);
```

上述程式片段也顯示了 WM_DESTROY 如何去執行視窗的清除工作。這個例子釋放了由 WM_CREATE 所配置的記憶體區塊。縱使 WM_CREATE 訊息的傳回值是 -1，也必須注意系統送出的 WM_DESTROY 訊息。如果檢查到 pData 不是 NULL，就必須呼叫 HeapFree 釋放記憶體。

識別行程和執行緒

可能會有某些程式（通常是系統層次的工具軟體）想要去操控一個行程，該行程擁有一個執行緒並且產生了一個視窗。舉個例，你可能希望讓使用者選擇螢幕上的一個視窗，然後將產生該視窗之行程結束掉。下面是一段程式例：

```
DWORD dwProcessId;
HANDLE hProcess;

// Find the handle to Calculator's window (as an example).
HWND hwnd = FindWindow(NULL, "Calculator");
GetWindowThreadProcessId(hwnd, &dwProcessId);
hProcess = OpenProcess(PROCESS_TERMINATE, FALSE, dwProcessId);
TerminateProcess(hProcess, 0);
CloseHandle(hProcess);
```

上述程式碼根據視窗代碼（window handle）取得一個行程識別碼（process ID），然後再利用此一行程識別碼取得「行程核心物件」（process kernel object）的代碼（handle）。GetWindowThreadProcessId 函式規格如下：

```
DWORD GetWindowThreadProcessId(HWND hwnd, LPDWORD lpdwProcessId);
```

這個函式接收一個視窗代碼作爲第一個參數，傳回產生此一視窗之執行緒的識別碼。假如你想要獲得「擁有此執行緒」之行程的識別碼，你必須傳入一個 DWORD 位址作爲 lpdwProcessId 參數。但假如你對行程識別碼沒有興趣，可以傳入 NULL。在先前的程式片段中，我只對行程識別碼感興趣，不在乎執行緒識別碼。

現在，使用此一行程識別碼，我便可以藉由 OpenProcess 函式得到一個行程核心物件（process kernel object）的代碼：

```
HANDLE OpenProcess(DWORD fdwAccess, BOOL fInherit, Dword IDProcess);
```

第一個參數 fdwAccess，告訴系統我們企圖對這個行程做些什麼事。在先前的程式片段中，我傳入 PROCESS_TERMINATE，因爲我想結束這個行程。當我呼叫 OpenProcess，它先做一個安全檢查，如果檢查不通過，OpenProcess 會傳回 NULL。由於 Windows 95

不像 Windows NT 一樣有安全檢查，所以在 Windows 95 中對行程的動作總是沒有問題。第二個參數 fInherit，表示這個代碼（handle）是否能被任何新衍生的子行程所繼承。最後，我們將 GetWindowThreadProcessId 傳回來的行程 ID，當作爲 OpenProcess 的第三個參數 IDProcess。

現在，有了行程物件代碼，我們可以呼叫 TerminateProcess 來終止行程：

```
BOOL TerminateProcess(HANDLE hProcess, UNIT uExitCode);
```

其中的 hProcess 參數表示我們想終止的是哪一個行程，uExitCode 參數表示行程的結束代碼。結束行程之後，我們呼叫了 CloseHandle 關閉行程核心物件，並從系統記憶體中釋放它。如果沒呼叫 CloseHandle，對應的系統記憶體會遺失掉，直到我們的行程結束爲止（才再歸還給系統）。

視窗的屬性

CreateWindowEx 會根據一個已註冊類別產生視窗。倘若 CreateWindowEx 在系統中發現指定的類別，它會配置一塊記憶體，用以儲存視窗實體的相關資訊。這塊記憶體包含了視窗類別結構的一些資訊，以及 CreateWindowEx 所收到的一些資訊。系統會根據 WNDCLASSEX 結構中的 cbWndExtra 大小，增加此記憶體區塊的大小，並將額外的大小初始化爲 0。每一個視窗實體都只能夠看到它自己的那塊額外位元組。就像視窗類別的額外位元組一樣。這些位元組僅能被視窗函式使用。

Win32 API 內含一些函式，允許你在產生一個視窗之後，取得和修改視窗結構的資訊。這些函式將在稍後小節中介紹。

產生出一個視窗之後，GetClassName 可用來取得其視窗類別的名稱：

```
int GetClassName(HWND hwnd, LPTSTR lpszClassName, int nMaxCount);
```

第一個參數 hwnd 是視窗的代碼。第二個參數 lpszClassName 是一緩衝區位址，緩衝區中填了類別名稱。最後一個參數 nMaxCount 表示緩衝區的最大長度。傳回值爲類別名稱字串的長度。

GetWindowWord，GetWindowLong，SetWindowWord 和 SetWindowLong 允許你取得和改變一個已存在之視窗的屬性：

```
DWORD     GetWindowLong(HWND hwnd, int nIndex);
WORD      GetWindowWord(HWND hwnd, int nIndex);
DWORD     SetWindowLong(HWND hwnd, int nIndex, LONG dwNewLong);
WORD      SetWindowWord(HWND hwnd, int nIndex, WORD wNewWord);
```

要獲得或改變視窗屬性，你必須提供一個已存在之視窗的代碼，以及你想要設定的屬性識別字。視窗代碼將做爲第一個參數，傳遞給上述四個函式。第二個參數 nIndex 代表你想要取得或改變的某個屬性。**表 1.5** 列出一些視窗屬性和識別字名稱，可以被做爲第二個參數傳遞給上述四個函式。所有識別字之前都有一個前置詞 GWL_。底線之前的字母 L（也就是 LONG 的意思）表示所取得的值是一個 32 位元值。

注意，如果「呼叫上述函式」之執行緒所隸屬的行程，和「產生那些視窗」之執行緒所隸屬的行程是同一個，SetWindowWord 和 SetWindowLong 才能正確無誤地運作。換句話說，雖然一個行程能取得（Get）另一行程所產生的視窗的屬性，但一個行程不能改變（Set）由另一行程所產生的視窗的屬性。

表 1.5 視窗屬性和其識別字

視窗屬性	Windows.H 中的識別字	屬性能否被更改
視窗函式位址	GWL_WNDPROC	Yes
產生者的 HINSTANCE	GWL_HINSTANCE	Yes
視窗的父視窗	GWL_HWNDPARENT	Yes
視窗的 ID / 選單（menu）	GWL_ID	Yes
視窗的擴充型態	GWL_EXSTYLE	Yes
風貌（style）	GWL_STYLE	Yes
使用者自定資料	GWL_USERDATA	Yes

和視窗類別屬性不一樣的是，所有視窗屬性都是長整數，必須使用 GetWindowLong 或 SetWindowLong 去取得或修改。所有的視窗屬性皆能被修改（而某些視窗類別的屬性卻僅能讀取不能修改）。大部份視窗屬性都可以從名稱上知其意義，這裡我僅提出一些注意事項。

第一，當某一行程的執行緒詢問「由另一行程之執行緒所產生的視窗」的視窗函式位址時，GetWindowLong 會傳回 0x00000000，這是因爲微軟認爲，一個視窗函式的位址對於其它行程根本不具任何實際用途與意義。當然啦，它其實對於偵錯程式（像 Spy[15]）和本章最後的 Voyeur 程式相當有用。然而，如果某一行程的執行緒詢問「由另一行程之執行緒所產生的視窗類別」的視窗函式位址，可以成功，這是因爲微軟看漏了這個問題。他們曾想阻止之，但作業系統開發者忘了處理。

第二，雖然作業系統允許你改變一個視窗的 HINSTANCE 值，我強烈建議你不要這麼做。微軟之所以這樣允許，是爲了回溯相容，然而卻有些程式利用這項能力去做些不好的事。

第三，當你呼叫 SetWindowLong 並使用 GWL_STYLE 或 GWL_EXSTYLE，系統並不僅只是改變視窗屬性以反映出新的風貌，系統會先送出一個 WM_STYLECHANGING 訊息給視窗函式，其 wParam 參數被設爲 GWL_STYLE 或 GWL_EXSTYLE，而 lParam 參數被設爲一個 STYLESTRUCT 結構位址，該結構定義如下：

```
typedef struct tagSTYLESTRUCT {
   DWORD  styleOld;
   DWORD  styleNew;
} STYLESTRUCT, * LPSTYLESTRUCT;
```

這個結構的 styleOld 欄位內含目前的風貌旗標，而 styleNew 欄位則爲新風貌旗標。視窗函式能夠查核 StyleNew 欄位中的旗標，如果它喜歡的話甚至可以修改之。WM_STYLECHANGING 訊息被處理過後，系統會更新視窗的旗標，並送出一個 WM_STYLECHANGED 訊息給視窗函式（其 wParam 和 lParam 參數的意義和

[15] Visual C++ 所附的 Spy++ 程式可以顯示出一個視窗的視窗函式位址，不過它是利用另一種技術獲得這項資訊。

WM_STYLECHANGING 訊息一樣）。無論做了任何改變，視窗函式皆會知道它的視窗風貌已經被改變了[16]。

呼叫 SetWindowWord 或 SetWindowLong 通常是爲了操控某一特定視窗的相關資料。以下告訴你如何以額外位元組（extra bytes）來記錄程式的執行時間：

1. 定義一個代表視窗額外位元組（window extra bytes）的識別字：

```
#define GWL_STARTTIME (0)
```

2. 註冊視窗類別，保留 4 個額外位元組給這個類別的每一視窗：

```
WNDCLASSEX wc;
.
.
.
wc.cbWndExtra = sizeof(LONG);
RegisterClassEx(&wc);
```

3. 視窗產生之後，將系統時間儲存於視窗的額外位元組中：

```
case WM_CREATE:
     SetWindowLong(hwnd, GWL_STARTTIME, GetTickCount());
     break;
```

4. 計算視窗執行的總時間：

```
DWORD dwSecondsRunning = (GetTickCount() -
                          GetWindowLong(hwnd, GWL_STARTTIME)) / 1000;
```

在這個例子中，我只需要 4 個額外位元組去儲存時間資料。微軟希望往後在每一視窗中皆能儲存一個 4 位元組數值，並且預先在每一個視窗的內部資料結構中配置一個 4 位

[16] 當一個視窗類別的風貌（style）被改變，WM_STYLECHANGING 和 WM_ STYLECHANGED 訊息並不會被送至視窗函式。舉個例，下列這一行不會產生任何視窗訊息：

```
SetClassLong(hwnd, GCL_STYLE, lNewStyle);
```

元組區塊。你可以藉著呼叫 GetWindowLong 或 SetWindowLong 並使用 GWL_USERDATA 去取得或設定這 4 位元組的值。就像任何你所配置的額外位元組一樣，系統不能去操控它 -- 除了在視窗產生時設定其初值爲 0。

這 4 位元組的空間非常有用。舉個例子，你可以產生一個對話盒，其中包含了幾個按鈕（buttons）、列示盒（ListBox）和編輯盒（edit）。如果你要在這些小視窗之間流通資訊，便需要找一塊空間來存放這些資訊。但由於你並沒有註冊這些 "System Global" 視窗類別，所以你不能使用它們的任何額外位元組（extra bytes）。然而，微軟承諾，這些類別的視窗函式絕對不會存取 GWL_USERDATA 所表示的 4 個位元組，所以你可以在 GWL_USERDATA 處儲存你自己的 32 位元值，並確保資料不會被系統破壞。

當你註冊自己的視窗類別，其視窗函式不應使用它自己的 GWL_USERDATA。請將此值保留給那些產生並操控視窗實體（衍生自你的視窗類別）的程式碼使用。對於類別使用者而言，如果想要在你的類別實體之間流通資料的話，事情將因此簡單得多。你並不需要這 4 個位元組，如果你需要，設定 WNDCLASSEX 結構中的 cbWndExtra 欄位即可 -- 畢竟你正在註冊這個類別，不是嗎？

視窗風貌（style）

CreateWindowEx 函式的第四個參數 dwStyle，允許你指定視窗風貌（style）。此設定僅會對此一視窗造成影響（如果指定的是類別中的視窗風貌，則會影響此類別所產生之所有視窗）。dwStyle 參數是一個 32 位元值，前 16 位元（較高字組）記載風貌相關資訊，由作業系統負責解析。作業系統定義了一些風貌旗標（常數），表 **1.6** 列出了所有這些旗標常數。請注意，每一個常數都有一個前置詞 WS_，表示此一識別字是一個 "window style"。

表 1.6　視窗類別的風貌（style）

風貌種類	識別字	實際數值
視窗型態（type）	WS_OVERLAPPED	0x00000000L
	WS_POPUP	0x80000000L
	WS_CHILD	0x40000000L
初始狀態（Initial states）	WS_MINIMIZE	0x20000000L
	WS_VISIBLE	0x10000000L
	WS_DISABLED	0x08000000L
	WS_MAXIMIZE	0x01000000L
截割風格(Clipping styles)	WS_CLIPSIBLINGS	0x04000000L
	WS_CLIPCHILDREN	0x02000000L
外觀（Appearance）	WS_CAPTION	0x00C00000L
	WS_BORDER	0x00800000L
	WS_DLGFRAME	0x00400000L
	WS_THICKFRAME	0x00040000L
	WS_HSCROLL	0x00100000L
	WS_VSCROLL	0x00200000L
	WS_SYSMENU	0x00080000L
放大縮小能力	WS_MINIMIZEBOX	0x00020000L
	WS_MAXIMIZEBOX	0x00010000L
輸入焦點的移換次序	WS_GROUP	0x00020000L
	WS_TABSTOP	0x00010000L

注意，WS_CAPTION 是 WS_BORDER 和 WS_DLGFRAME 的組合。由於一個視窗不能同時既是 WS_BORDER 又是 WS_DLGFRAME，所以如果這兩種風貌同時出現，Windows 會將它視爲 WS_CAPTION。

另外也請注意，WS_MINIMIZEBOX 和 WS_MAXIMIZEBOX 的實際數值與

WS_GROUP 和 WS_TABSTOP 的實際數值一樣。當視窗是一個對話盒子視窗，系統會把這些位元視爲 WS_GROUP 和 WS_TABSTOP，用以決定輸入焦點的移換次序。然而如果視窗擁有標題欄（Caption），系統會把這些位元視爲 WS_MINIMIZEBOX 和 WS_MAXIMIZEBOX。

作業系統本身並不解析 dwStyle 參數的後 16 個位元（較低字組），這 16 個位元的意義視不同的視窗類別而不同。視窗類別的視窗函式決定了這些位元的意義。微軟已經爲每一個"System Global"視窗類別（例如列示窗 listbox、按鈕 button、捲動軸 scrollbar）定義了它們的風貌。如果你註冊自己的視窗類別，就可以爲這 16 個位元產生出自己的定義。**表 1.7** 列出作業系統的每一個 "System Global" 視窗類別，並且每一個類別所使用的前置詞。

表 1.7 "System Global" 視窗類別的風貌（style）識別字前置詞

全域視窗類別（System global class）	前置詞（Prefix）
對話盒（DIALOG）	DS_
按鈕（BUTTON）	BS_
複式清單（COMBOBOX）	CBS_
編輯窗（EDIT）	ES_
列示盒（LISTBOX）	LBS_
MDI 程式的 Client 視窗（MDICLIENT）	MDIS_
捲動軸（SCROLLBAR）	SBS_
靜態文字（STATIC）	SS_

DS_ 專門用在對話盒的資源描述檔中。當 DialogBox 和 CreateDialog 被呼叫時，系統會依此型式產生對話盒。

檢閱 Windows 表頭檔中的識別字實際數值，你會發現有很多數值是重複的。舉個例，BS_DEFPUSHBUTTON 和 SS_CENTER 兩者都是 0x00000001。這不會引起任何衝突，

因爲只有視窗類別的視窗函式，才會使用這些位元，而不同類別的視窗函式會針對這些位元做出不同的解釋。

一個視窗函式能藉著 GetWindowLong 取得風貌（style）相關資訊。一旦取得後，該函式會將它與一個特定識別字做 AND 運算，以確定該種風貌是 on 或是 off。這份資訊可用來修改視窗的行爲。

從 Windows 3.0 版開始，微軟增加了幾個新的視窗風貌，稱爲視窗的擴充風貌（extended window style），並指定做爲 CreateWindowEx 的第一個參數 dwExStyle 的內容。這個擴充風貌也是一個 32 位元長整數值。然而，微軟保留了所有 32 個位元，讓作業系統去解析其意義。沒有任何一個擴充位元可以被用來指定與某特定類別相關的資訊。**表 1.8** 列出在 WinUser.H 表頭檔中定義的一些擴充風貌（最前面都有一個前置詞 WS_EX_）。

表 1.8　視窗類別的延伸型態

擴充風貌（extended style）種類	識別字（Identifier）	實際數值（Value）
外觀 (Appearance)	WS_EX_DLGMODALFRAME	0x00000001
	WS_EX_STATICEDGE	0x00020000
	WS_EX_CONTEXTHELP	0x00000400
	WS_EX_TOOLWINDOW	0x00000080
	WS_EX_WINDOWEDGE	0x00000100
	WS_EX_CLIENTEDGE	0x00000200
	WS_EX_APPWINDOW	0x00040000
通訊 (Communication)	WS_EX_TOPMOST	0x00000008
	WS_EX_TRANSPARENT	0x00000020
	WS_EX_NOPARENTNOTIFY	0x00000004
	WS_EX_ACCEPTFILES	0x00000010
	WS_EX_MDICHILD	0x00000040
	WS_EX_CONTROLPARENT	0x00010000

擴充風貌（extended style) 種類	識別字（Identifier)	實際數值（Value）
定向（Orientation）	WS_EX_LEFT	0x00000000
	WS_EX_LTRREADING	0x00000000
	WS_EX_RIGHTSCROLLBAR	0x00000000
	WS_EX_RIGHT	0x00001000
	WS_EX_RTLREADING	0x00002000
	WS_EX_LEFTSCROLLBAR	0x00004000

Window Properties

Window Properties 提供給程式員一個方法，讓他們把資料關聯到某個視窗去。如果你想要把一份資料和某個視窗產生關係，但是你並沒有註冊該視窗的視窗類別，那麼 window properties 會是一個很好的利器。如果你沒有註冊該視窗類別，你就不知道 WNDCLASSEX 中指定了多少個額外位元組（extra bytes）。雖然 GetClassInfoEx 或 GetClassLong 可以取得這些額外位元組的資料，但你大概可以想像，既然它們被配置出來，它們大概也已經被此類別的視窗函式使用了。爲了你自己的目的而使用這些額外位元組，可能會影響到這個視窗的原始行爲。

所謂 window properties，允許你以一個字串名稱（而不是直接修改視窗結構內容）來和視窗做資料的關聯動作。只有 32 位元數值才可以被當做是一個 properties。由於系統必須以 properties 的名稱字串來和視窗產生關聯，所以比起視窗的額外位元組（extra bytes）而言，properties 的執行速度比較慢，需要的記憶體也比較大。

Win32 API 提供 5 個函式來處理視窗的 properties：

- SetProp ，用來增加或修改視窗的某個 properties 內容：

```
BOOL SetProp(HWND hwnd, LPCTSTR lpString, HANDLE hData);
```

- RemoveProp，用來刪除視窗的一個 property：

```
HANDLE RemoveProp(HWND hwnd, LPCTSTR lpString);
```

- GetProp，用來取得視窗的一個 property：

```
HANDLE GetProp(HWND hwnd, LPCTSTR lpString);
```

- EnumProps 和 EnumPropsEx，用來獲得一個視窗的所有 properties 項目：

```
int EnumProps(HWND hwnd, PROPENUMPROC lpEnumFunc);
int EnumPropsEx(HWND hwnd, PROPENUMPROCEX lpEnumFunc, LPARAM lParam);
```

以下例子中告訴你如何從一個 modal 對話盒中取得一些資訊，並將資訊傳回給呼叫者：

1. 呼叫端配置記憶體，用來儲存由 modal 對話盒所取得的資訊：

```
#define MAX_USERS_NAME_LEN    (30)
.
.
.
pData = malloc(MAX_USERS_NAME_LEN + 1);
DialogBoxParam(hinstExe, MAKEINTRESOURCE(IDD_USERNAME), hwnd,
   UserName_DlgProc, (LONG) pData);
// The local memory block will contain the user's name.
.
.
.
```

2. 對話盒函式將記憶體代碼存入對話盒的 properties 中：

```
case WM_INITDIALOG:
   // The lParam contains the last parameter value passed to
   // DialogBoxParam. This is the handle to the local block of memory.
   SetProp(hwnd, "Memory", (HANDLE) lParam);
   // Perform any other initialization for the dialog box.
   .
   .
   .
```

3. 當使用者按下【OK】鈕，對話盒於是將資料寫入記憶體區塊中：

```
case IDOK:
   pszName = (PSTR) GetProp(hwnd, "Memory");
   GetDlgItemText(hwnd, ID_USERNAMEEDITBOX, (LPSTR) pszName, MAX_USERS_NAME_LEN);
   .
   .
   .
   EndDialog(hwnd, IDOK);
   break;
```

4. 當對話盒被摧毀，property 也應被刪除。你並不需要自己動手做這個動作，因爲當視窗被摧毀時，系統會自動將與該視窗有關聯的所有 properties 刪除。不過，自行移除 properties 會使你的觀念更清晰：

```
case WM_DESTROY:
   RemoveProp(hwnd, "Memory");
   break;
```

當然，你也可以總是呼叫 GetWindowLong 和 SetWindowLong 並使用 GWL_USERDATA 識別字，將 4 個位元組和視窗關聯在一起。事實上這個方法比使用 properties 更快更容易。然而當你有很多資料要和一個視窗產生關聯，4 個位元組將不夠使用。解決這個問題的最好方法就是配置一塊足夠大的記憶體，將你想要處理的資料放入，然後將指向此記憶體區塊的指標存入 GWL_USERDATA 之中。由於 4 位元組的 GWL_USERDATA 的出現，window properties 已經日漸式微了。

視窗訊息

視窗訊息爲一個未帶正負號的 32 位元整數，介於 0x00000000 到 0xFFFFFFFF 之間。微軟將視窗訊息劃分爲四個區段，並定義出每一個區段的意義（請看表 **1.9**）。

表 1.9 視窗訊息的四個區域

訊息區段	區段意義
0x00000000 ~ WM_USER[17] - 1	所有標準視窗訊息皆定義在此範圍內，包括所有以 WM_ 爲前置詞的訊息。你不應自行產生一個落於此範圍內的訊息。
WM_USER ~ WM_APP[18] - 1	與某一類別有特定關係的訊息。
WM_APP ~ 0x0000BFFF	與某一應用程式有特定關係的訊息，可被單一行程使用。

[17] 在 WinUser.H 表頭檔中，WM_USER 被定義爲 0x00000400。

[18] 在 WinUser.H 表頭檔中，WM_APP 被定義爲 0x00008000。

訊息區段	區段意義
0x0000C000 ~ 0x0000FFFF	"System Global" 字串訊息。也就是由 RegisterWindowMessage 傳回的訊息數值。
0x00010000 ~ 0xFFFFFFFF	微軟保留的訊息，只供系統使用。

類別專屬整數訊息（Class-Defined Integer Messages）

當你註冊自己的視窗類別，你可以定義一些訊息去執行某些特定的工作。假設你產生了一個 INFO 視窗類別，擁有一塊記憶體並且允許其它視窗存取這塊記憶體。你可以爲此 INFO 視窗產生一個與類別相關的訊息；一旦視窗收到這個訊息，就傳回記憶體區塊的位址。這些類別專屬訊息應該被定義在表頭檔（例如 Info.H）中。這個表頭檔案應該被含入於所有即將傳送訊息給 INFO 視窗的模組之中。Info.H 內容如下：

```
#define IM_GETMEMORY  (WM_USER + 0)
```

Info.C 之中內含 INFO 視窗類別的視窗函式，函式片段如下：

```
#include "Info.H"
.
.
.
case IM_GETMEMORY:
   lResult = GetWindowLong(hwnd, GWL_MEMORYHANDLE);
   break;
.
.
.
return(lResult);
```

類別專屬訊息僅送到曾有定義該訊息的類別視窗中。如果將類別專屬訊息送到其它的類別視窗中，會產生不可預期的後果。

16 位元 Windows 表頭檔所定義的類別專屬訊息可以被送往所有“System Global”類別。請注意每一個訊息都有一個獨一無二的前置詞（表 **1.10**）。

表 1.10 "System Global" 視窗類別的訊息識別字的前置詞

"System Global" 視窗類別	訊息字首
對話盒（DIALOG）	DM_
按鈕（BUTTON）	BM_
複式清單（COMBOBOX）	CB_
編輯窗（EDIT）	EM_
列示盒（LISTBOX）	LB_
MDI 程式之 Client 視窗（MDICLIENT）	無
捲動軸（SCROLLBAR）	SBM_
靜態文字（STATIC）	STM_

微軟在開發 Win32 API 時，決定改變代表這些訊息的實際數值。舉個例，在 16 位元視窗環境下，BM_SETCHECK 被定義爲 WM_USER+1，而在 Win32 環境下，BM_SETCHECK 訊息被定義爲 0x00F1。這表示這個訊息不再是類別專屬訊息的一部份，而被移到了標準訊息區段。微軟當然可以這麼做，因爲作業系統和 "System Global"視窗類別都是由它製造的[19]。

然而，如果你曾看過 new common control，你就會發現，那些訊息被定義在類別專屬訊息區段中。舉個例，TB_ENABLEBUTTON（代表工具列上的一個按鈕究竟是致能 enabled，或是除能 disabled）被定義爲 WM_USER+1。注意，系統不允許這個區段內的訊息被廣播到所有其它視窗中。

---

[19] 微軟之所以讓系統類別訊息小於 WM_USER，爲的是做 16/32 位元之間的 thunking 動作。有一些 16 位元程式會送 EM_GETLINE 訊息給 non-edit 控制元件視窗，由於系統不知道這些視窗類別以相同的語意來解釋 EM_GETLINE 訊息，所以系統不能夠正確地 "thunk"（轉換）其參數。如果設定與系統控制元件相關的訊息都小於 WM_USER，Win32 便能夠確保讓你送出這些訊息給 non-system class 的視窗。（譯註：Thunking 可分爲 thunk down 和 thunk up 兩種。32 位元 API 呼叫 16 位元 API，需經過 thunk down，反之需經過 thunk up。Windows 95 的 thunking 動作稱爲 flat thunk 或 universal thunk，Windows NT 的 thunking 動作稱爲 generic thunk）

應用程式專屬整數訊息（Application-Specific Integer Messages）

這個範圍的整數值被保留給某一特定行程。換句話說，當你設計一個視窗類別，其視窗函式不應定義任何訊息在這範圍之中。一個程式可以定義一些它知道絕對不會與其它視窗訊息衝突的特定訊息。一個程式如果想要使用 PostThreadMessage 傳送訊息給一個執行緒（而不是一個視窗），它可能會這麼做：

```
BOOL PostThreadMessage(DWORD idThread, UINT Msg, WPARAM wParam, LPARAM lParam);
```

程式如果想要 "subclassing" 它所產生的某些視窗，它也可以定義一些特定訊息。那些訊息將會被「subclass procedure」處理，而不會被原來的視窗函式處理。本書第五章「視窗的 Subclassing 和 Superclassing」中有更詳細的討論。

注意，就像類別專屬視窗訊息一樣，系統也不允許應用程式專屬訊息被廣播給系統中其他的視窗。

系統全域訊息（System-Global String Message）

如果你想送出一個訊息，但不確定哪一個視窗會收到它，這時候你可以產生一個系統全域訊息。當你註冊這個系統全域訊息，就是告訴系統說，現在又多了一個新的「標準」訊息，任何程式所產生的任何視窗皆可以使用它。

RegisterWindowMessage 可以用來註冊一個新的訊息：

```
UINT RegisterWindowMessage(LPCTSTR lpString);
```

這個函式接受一個字元字串，傳回一個介於 0x0000C000 到 0x0000FFFF 之間的數值。而在系統內部，RegisterWindowMessage 就像 RegisterClassEx 一樣，在系統的 local atom table 建立一個 atom 值。如果其它程式亦以相同的字元字串作爲參數，呼叫 RegisterWindowMessage，系統會在第一時間傳回相同的數值。由於這樣，所以不同的視窗可以使用相同的整數值來表示相同的訊息。

任何一個程式如果想使用這些新訊息，就應該在初始化期間呼叫 RegisterWindowMessage。一旦新訊息被註冊後，會一直存在，直到關機或系統重新啓動爲止 -- 因爲 Windows 沒有提供一個可以註銷訊息的函式。

視窗和程式的生與死

以下是一個 Win32 程式的基本架構：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstExePrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   WNDCLASSEX wc;
   ATOM atomClassName;
   .
   .
   .
   // Initialize the wc structure.
   atomClassName = RegisterClassEx(&wc);
   // Create a window of the registered class.
   hwnd = CreateWindowEx(0, MAKEINTATOM(atomClassName), ...);

   return(0);
}
```

這段碼註冊了一個視窗類別並產生一個視窗實體。行程的主執行緒從 WinMain 傳回，會使得 C-Runtime Library 執行一些清除工作；然後執行緒和行程都結束掉。當一個執行緒終止，作業系統便摧毀由這個執行緒所產生的視窗。而當行程終止，系統會註銷視窗類別。

由於產生視窗之後，執行緒和行程立即結束，所以上述程式毫無作用。爲了避免這種事情發生，WinMain 之中應該引進一個所謂的訊息迴路：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstExePres,
   LPSTR lpszCmdLine, int nCmdShow) {

   WNDCLSSEX wc;
```

```
   ATOM atomClassName;
   MSG msg;
   .
   .
   .
   // Initialize the wc structure.
   atomClassName = RegisterClassEx(&wc);

   // Create a window of the registered class.
   hwnd = CreateWindowEx(0, MAKEINTATOM(atomClassName), ...);

   // A simple message loop
   while (GetMessage(&msg, NULL, 0, 0)) {
      DispatchMessage(&msg);
   }
   // End of the simple message loop

   return(0);
}
```

這個訊息迴路內含一個對 GetMessage 的呼叫動作。如果 GetMessage 傳回 TRUE，於是呼叫 DispatchMessage。當 DispatchMessage 回返，while 迴圈會繼續呼叫 GetMessage。現在，讓我說明這裡將發生什麼事。視窗一旦產生，目的只有一個，就是回應視窗訊息。有兩種方法可以將訊息送給視窗函式：**send** 或 **post**。SendMessage[20] 可以 “Send”一個訊息：

```
LRESULT SendMessage(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
```

如果產生視窗的那個執行緒呼叫了上述函式，那麼 SendMessage 會直接呼叫對應的視窗函式。視窗函式會處理這個訊息，並傳回一個 LRESULT 值給 SendMessage。SendMessage 然後會將這個傳回值傳給呼叫者。假如執行緒呼叫 SendMessage 並送出訊息給其他執行緒所產生的視窗，那麼系統內部的動作會複雜許多，但最後的結果（亦即傳回 LRESULT）是一樣的。

[20] 其它可以 “send” 訊息的 Win32 函式還有 SendNotifyMessage、SendMessageTimeout 和 SendMessageCallback。爲了方便起見，我只討論 SendMessage。

我們可以使用 PostMessage 去 "post" 一個訊息：

```
BOOL PostMessage(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
```

此函式直接將一個視窗訊息放置在訊息佇列（message queue）之中。系統的每一個執行緒都擁有自己的訊息佇列。如果一個行程產生了 20 個執行緒，則就有 20 個訊息佇列。PostMessage 會藉由 hwnd 參數得知要將訊息送給那一個執行緒，然後將訊息加入到此執行緒的訊息佇列的尾端。如果一個執行緒產生了 100 個視窗，則 "post" 到其中任何一個視窗的訊息，都將被加入這個執行緒的訊息佇列之中。

當執行緒呼叫 GetMessage，此函式會檢查執行緒的訊息佇列，看看是否有訊息存在。如果佇列中沒有任何訊息存在的話，系統會將此執行緒排除在 CPU 排程之外，於是執行緒呈現睡眠狀態。當此執行緒的訊息佇列中出現訊息，系統會喚醒此執行緒，並且 GetMessage 會從此佇列中將訊息拷貝到一個 MSG 資料結構，此結構的位址可由 GetMessage 的第一個參數獲得。GetMessage 會將此訊息從執行緒的訊息佇列中移除，並將此訊息傳回到執行緒的訊息迴路中。如果 GetMessage 傳回 TRUE，於是便會執行訊息迴路內的程式碼。先前的程式例只是簡單呼叫了 DispatchMessage 並傳入由 GetMessage 獲得的 MSG 結構：

```
LRESULT DispatchMessage(CONST MSG *lpMsg);
```

DispatchMessage 的動作和 SendMessage 非常類似：它直接呼叫視窗類別的視窗函式，並傳入 MSG 資料結構，其內包含了 hwnd，uMsg，wParam，和 lParam。視窗函式會傳回給 DispatchMessage 一個 LRESULT 值，而 DispatchMessage 會再將此值傳回給呼叫它的執行緒。我們很清楚地看在，訊息迴路中並沒有用到 LRESULT 值。事實上，我從未看過任何程式在訊息迴路中使用 LRESULT 值做任何事情。

注意，GetMessage 只會從呼叫者（執行緒）的訊息佇列中取訊息。如果你在你的行程中產生其它執行緒，而這些執行緒使用 CreateWindowEx 產生視窗，那麼每一個執行緒都必須有它們自己的訊息迴路。另外必須注意的是，執行緒的訊息迴路會使得執行緒沒有辦法結束，而由於執行緒沒有辦法立刻結束，由此執行緒所產生的任何視窗也就不會被

摧毀，它們仍然等著處理行程的視窗訊息。

當然，當 GetMessage 傳回 FALSE，執行緒的訊息迴路便會結束。GetMessage 之所以會傳回 FALSE 是因爲它從執行緒訊息佇列中獲得了 WM_QUIT 訊息。WM_QUIT 不會被任何視窗函式處理，因爲訊息迴路已經終止，不會再執行 DispatchMessage。

現在，我們來探討訊息佇列中的 WM_QUIT 如何結束視窗。以下將敘說其因果關係。首先，當使用者按下視窗的關閉鈕（Close box）或選擇視窗的系統功能表中的【Close】選項，會有一個 WM_CLOSE 訊息送往視窗函式。這個訊息表示使用者要求關閉視窗。我們不需要特別處理這個訊息，通常都是交由 DefWindowProc 去處理，它會呼叫 DestroyWindow。

接下來，DestroyWindow 告訴作業系統說，視窗即將被摧毀。作業系統於是送出 WM_DESTROY 訊息給視窗函式[21]。爲了回應這個訊息，視窗函式應該呼叫 PostQuitMessage：

```
BOOL PostQuitMessage(int nExitCode);
```

此函式會放置一個 WM_QUIT 訊息到呼叫者的執行緒訊息佇列[22]中，然後立刻回返，以便視窗函式能完成 WM_DESTROY 的處理。呼叫 PostQuitMessage 有其絕對必要性，因爲做爲一個程式發展者，我們需要隨時保持視窗和執行緒（或行程）之間的聯繫。我常常看到程式員在處理 WM_DESTROY 訊息時忘記呼叫 PostQuitMessage。如果忘了呼叫它，視窗仍會被摧毀，但執行緒卻不會結束：因此行程仍然活著！於是我們現在有了一

---

[21] 事實上，呼叫 DestroyWindow 可以促使系統摧毀許多視窗，而不僅僅是 DestroyWindow 參數所指定的那個視窗。舉個例子，父視窗如果被摧毀，其所產生的子視窗也會被摧毀。此外，其所產生的子視窗、孫視窗、曾孫視窗也都會被摧毀。首先，系統會送給每個視窗一個 WM_DESTROY 訊息 -- 由 DestroyWindow 所指定的視窗開始。然後，那些個視窗會再收到 WM_NCDESTROY 訊息。然而，WM_NCDESTROY 訊息是從輩份最小的視窗開始送，所以 DestroyWindow 所指定的視窗將是最後收到 WM_NCDESTROY 訊息的人。

[22] 事實上只是設定一個位元旗標而已，不是真的有一個 WM_QUIT 訊息。這一點在 ***Advance Windows*** 一書（Microsoft Press, 1995）中有更詳盡的討論。

個沒有視窗的行程。由於沒有任何使用者介面，使用者不知道行程還在執行，更不知道它還在使用系統資源。唯一能終止這種行程的方法，就是使用【Close Program】對話盒（侯俊傑註：按下 Ctrl-Alt-Del），或是簽退（log off），或是關機（shut down）。

讓我告訴你爲什麼 DefWindowProc 對於 WM_DESTROY 訊息的預設處理行爲不是去呼叫 PostQuitMessage。原因是一個程式可能產生很多個視窗，而通常只有一個視窗會被指定爲執行緒（或行程）生命的代表。如果 DefWindowProc 預設行爲是呼叫 PostQuitMessage，那麼當第一個視窗（任一視窗）被關閉，執行緒會立即終止執行緒的訊息迴路。想想看，當使用者開啓一個【File Open】對話盒並選擇【OK】之後，整個程式竟然就結束了，豈不是很可笑嗎？

一旦 WM_QUIT 進入執行緒訊息佇列中排隊，表示執行緒已處理完視窗的 WM_DESTROY 訊息。接下來視窗便會收到一個 WM_NCDESTROY 訊息，之後便將視窗摧毀。現在，訊息迴路再次獲得控制權，GetMesage 會獲得 WM_QUIT 訊息，於是傳回 FALSE，訊息迴路便告終止。這時候視窗已經被摧毀，所以當執行緒結束，系統之中再也沒有由此執行緒所產生的視窗，系統再也不必強迫摧毀任何視窗了！

當系統強制摧毀視窗（因爲產生視窗的那個執行緒結束了生命），系統並沒有送出 WM_CLOSE、WM_DESTROY 或 WM_NCDESTROY 訊息給視窗函式。怎麼可以這樣呢？是的，因爲此刻執行緒已經結束了，再不能執行任何程式碼。因此視窗不會被告知它將被摧毀。如果視窗函式想要在處理 WM_CLOSE 訊息時提醒使用者儲存尚未儲存的資料，在系統強制摧毀的情況下將不會進行；此外在 WM_DESTROY 或 WM_NCDESTROY 訊息所做的一些例行清除工作，也都不會執行起來。摧毀視窗的最好方法就是呼叫 DestroyWindow；最好不要依賴系統的強制摧毀，因爲彼時產生視窗的那個執行緒已經結束了，許多清理工作無法進行。

關閉系統

當使用者企圖離開或關閉系統，系統會通知所有正在執行的行程[23]。系統並不是真的一一去通知，它只是廣播 WM_QUERYENDSESSION 訊息給每一個 unowned（無父視窗）並擁有 WS_OVERLAPPED 風格的視窗。請注意，如果一個行程產生了五個 unowned 並且是 overlapped 的視窗，每個視窗都將收到 WM_QUERYENDSESSION 訊息。如果行程沒有產生任何這類視窗，就不會收到任何 WM_QUERYENDSESSION 訊息，於是當使用者要離開（簽退）或關閉系統時，視窗們不會獲得任何通知。

當系統送出 WM_QUERRYENDSESSION 訊息，便是告訴視窗說使用者想要退出這個 Windows session，而此一訊息即是詢問視窗是否介意。WM_QUERRYENDSESSION 和 WM_CLOSE 極為類似，都是要求結束一個活動期（session）。處理 WM_QUERRYENDSESSION 訊息時，視窗函式可以決定是否拒絕這項請求；如果拒絕，Windows session 就不會結束。如果你同意，視窗函式必須傳回 TRUE；如果你不同意，視窗函式必須傳回 FALSE。

通常，視窗函式在處理 WM_QUERYENDSESSION 訊息時，會檢查是否有些資料尚未被儲存。舉個例子，如果你啓動 WordPad，輸入一些文字但不要儲存它們，然後嘗試關閉系統。當 WordPad 收到 WM_QUERYENDSESSION 訊息，會顯示一個對話盒如圖 1.2。

---

[23] 一個程式可以使用 ExitWindowsEx 將系統關閉：

```
BOOL ExitWindowsEx(UINT uFlags, DWORD dwReserved);
```

這個函式會將 WM_QUERYENDSESSION 訊息廣播給所有 unowned、overlapped 視窗 -- 被「呼叫 ExitWindowsEx 之執行緒」所產生的視窗除外，因爲系統假設，呼叫 ExitWindowsEx 的執行緒自己知道系統正在關閉，而且一個執行緒也很難知道另一個執行緒何時會試圖關閉系統。

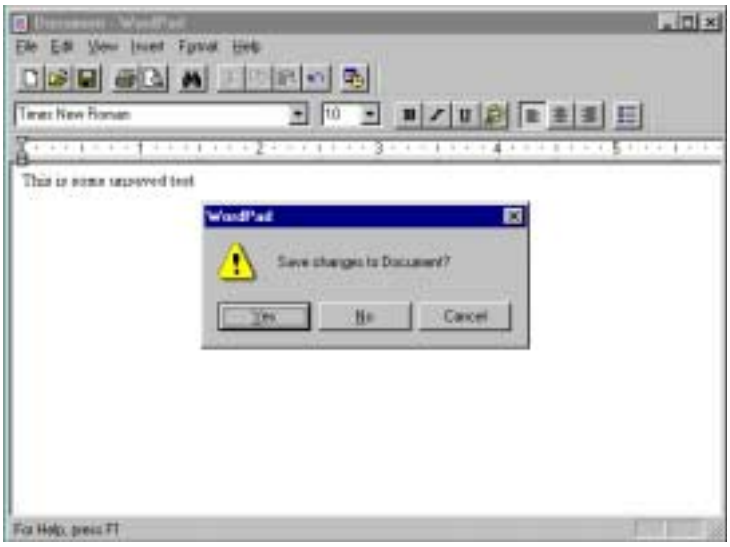


圖 1.2　WordPad 詢問使用者是否要儲存更改過的資料。

如果使用者在圖 **1.2** 中選擇【Yes】，就會出現【檔案儲存】對話盒。而當使用者儲存檔案之後，系統會收到 TRUE，於是可以繼續做一些系統關閉前的準備工作。如果使用者選擇的是【No】，將看不到任何反應發生，而系統還是收到 TRUE。如果使用者選擇的是【Cancel】，系統會收到 FALSE，於是 Windows session 不會被結束。如果我們把 WM_QUERYENDSESSION 交給 DefWindowProc 去處理，系統總是會收到 TRUE。

所有視窗都回應了 WM_QUERYENDSESSION 之後，系統會廣播 WM_ENDSESSION 訊息給所有視窗。如果 Windows session 即將結束，WM_ENDSESSION 的 wParam 參數會是 TRUE，反之則爲 FALSE。下面這段文字說明了 WM_ENDSESSION 的訊息狀態：

> If the fEndSession parameter is TRUE, the Windows session can end any time after all applications have returned from processing this message. Therefore, an applications should perform all tasks required for termination before returning from this message. The application need not call the DestroyWindow or PostQuitMessage function when the session is ending.

讓我們仔細檢驗上述最後一句話。如果你沒有呼叫 DestroyWindow 或 PostQuitMessage，誰會去做呢？答案是沒有人會去做！的確，你的視窗不會收到像是 WM_CLOSE、WM_DESTROY 或 WM_NCDESTROY 之類的訊息。取而代之的是，在系統送出 WM_ENDSESSION 訊息給所有 unowned、overlapped 視窗之後，便針對這個 Windows session 中的每一個正在執行的行程，強制呼叫 TerminateProcess 以結束之。這意味著沒有一個行程能夠乾淨地結束。依我之見，微軟不應該鼓勵程式員撰寫這樣的程式。

讓我以實例來告訴你微軟自己的 WordPad 的一些古怪行爲。開啓 WordPad，使用【檢視】選單關掉視窗工具列（Toolbar），然後按下視窗右上角的關閉盒，結束 WordPad。再重新開啓 WordPad，你會發現工具列不見了。這是因爲當使用者關閉視窗時，WordPad 儲存其【檢視】選單中的設定，並將之儲存到 Registry 中。

現在，再回到 WordPad 的【檢視】選單下，打開視窗工具列，但這次不要再按下關閉盒去關閉視窗，而是進入系統的 TaskBar 並退出系統。於是系統會送出 WM_QUERRYENDSESSION 訊息並緊跟著一個 WM_ENDSESSION 訊息給 WordPad 視窗。WordPad 行程會被終止，其視窗會被摧毀。現在，我們再登入系統並執行 WordPad。你認爲會看到什麼東西？我們會看到 WordPad 工具列嗎？

我認爲我們應該看到工具列，因爲上次我們使用 WordPad 時它是存在的。然而，當你啓動 WordPad 時，你卻看不到工具列，那是因爲當我退出系統時，WordPad 已經結束了。也就是說其執行緒沒有機會更新 Registry 的設定以反應程式狀態。WordPad 只不過是擁有這種奇怪行爲的眾多程式中的一個例子，其它許多程式在 Windows session 終了時也都未曾去更新設定。我建議你不要遵循微軟文件中所指示的方法，以 WM_ENDSESSION 訊息去結束視窗；你最好自行完成。以下是程式的基本架構：

```
BOOL Cls_OnQueryEndSession (HWND hwnd) {
   BOOL fOKToEndSession = TRUE;
   if (g_fIsDataUnsaved) {
      // Present message box
```

```
      int n = MessageBox(hwnd, "Do you want to save changes?",
         "Application caption", MB_YESNOCANCEL | MB_ICONWARING);

      if (n == IDYES) {
         .
         .
         .
         // Present a file save dialog box here
      }

      if (n == IDCANCEL)
         fOKToEndSession = FALSE;
    }
    return(fOKToEndSession);
}

////////////////////////////////////////////////////////////////////////
void Cls_OnEndSession (HWND hwnd, BOOL fEnding) {

   if (fEnding) {

      // If the session is really ending, explicitly destroy the window.
      DestroyWindow(hwnd);

      // DestroyWindow sends WM_DESTROY and WM_NCDESTROY messages.
   }
}

////////////////////////////////////////////////////////////////////////
void Cls_OnDestroy (HWND hwnd) {

    // The main window is being destroyed; signal the message loop to terminate
    // so that the thread and process terminate.
    PostQuitMessage(0);
}

////////////////////////////////////////////////////////////////////////
void Cls_OnClose (HWND hwnd) {

    // We handle WM_CLOSE the same way we would handle a logoff or shutdown.
    BOOL fOKToClose = Cls_OnQueryEndSession(hwnd);
    Cls_OnEndSession(hwnd, fOKToClose);
}
```

上面的訊息剖析函式在 Voyeur 程式中有詳細的討論。你可以在 Voyeur.C 中找到它們。

Voyeur 程式

Voyeur 程式（Voyeur.EXE），顯示於**列表 1.1-1.4** 中，用來展示本章出現過的這些觀念：

- 如何使用訊息剖析器來產生一個視窗函式，以及如何註冊一個視窗類別。
- 如何產生單一視窗實體。
- 如何使用各種函式取得視窗及類別的相關資訊。
- 如何正確結束程式 – 即使系統正在關閉中。

執行這個程式，主視窗如**圖 1.3** 所示。Voyeur 的工作區（client area）將顯示被指定視窗的相關資訊。使用者首先在視窗工作區中按下滑鼠右鍵，不要放開，將滑鼠移動到你想要窺視的視窗上，然後放開滑鼠。於是 Voyeur 開始對此視窗做窺視動作。

圖 1.3 Voyeur 的起使視窗

在 Voyeur 視窗上按下滑鼠右鍵會使 Voyeur 視窗退至背景，其它視窗則統統提升到上面。

然後 Voyeur 會改變滑鼠的游標成爲一對「眼睛」，表示它能窺視螢幕上的所有視窗。當滑鼠游標進入某一視窗時，此視窗會出現一個粗外框以提示你。選擇了你所要窺視的視窗之後，Voyeur 會更新其工作區資訊，並將它自己帶到最上層，於是你可以很清楚地看到 Voyeur 視窗，不需花太多時間去尋找。

圖 1.4 是 Voyeur 的窺視結果。在這個例子中，Voyeur 窺視了 Windows 95 的工作列（Taskbar）並顯示其相關資訊。多麼方便啊，Voyeur 能解析類別風格、視窗風格、視窗擴充風格，它可以顯示每一種風格的文字名稱，及其十六進位值。類別的額外位元組、視窗的額外位元組、視窗的 properties 等等等也都顯示在**圖 1.4** 中。

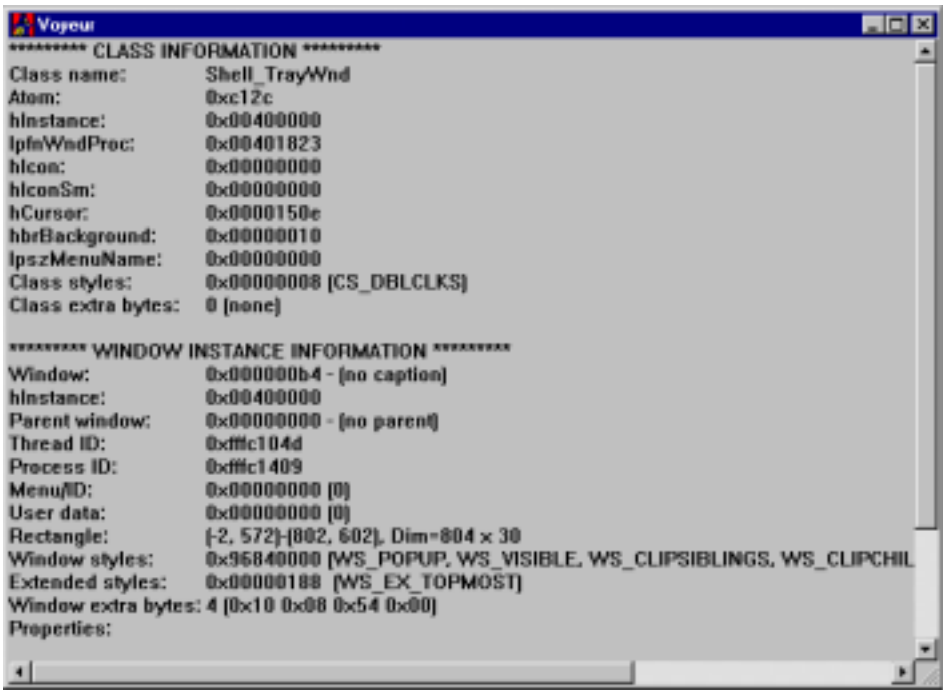


圖 1.4 Windows 95 工作列的相關資訊

此程式所使用的檔案列於**表 1.11** 中。

表 1.11　Voyeur 程式所用到的檔案

檔案	說明
Voyeur.C	內含 WinMain 函式、主視窗類別的視窗函式，以及訊息剖析器函式。
VoyHelp.C	內含輔助函式（helper function），目的在取得某個視窗及其類別的相關資訊，並將結果放置於編輯窗中。
Voyeur.RC	內含對話盒面版（dialog template），及其圖示（icon）。
Resource.H	內含 Voyeur.RC 檔中所有資源的識別碼（ID）。
Voyeur.ICO	主視窗圖示檔。
Voyeur.CUR	窺視游標。
Voyeur.MAK	Visual C++ 專案所需的 makefile。

Voyeur 程式的初始化動作

Voyeur 首先註冊主視窗的視窗類別，然後產生主視窗。視窗的所有初始化動作皆在訊息剖析函式 Voyeur_OnCreate 中處理。Voyeur 利用 CreateWindowEx 產生一個“edit”視窗，用以顯示被窺視之視窗的相關資訊。現在，我將設定類別名稱爲“edit”並指定一些與類別相關的風格，像是 ES_READONLY、ES_LEFT 和 ES_MULTILINE。一旦 edit 視窗產生出來，EM_SETTABSTOPS 訊息就會被送過去，使“edit”視窗的文字排列整齊些。如果“edit”視窗不能被順利產生，必須立刻停止 Voyeur 主視窗的建立。在這種情況下，WinMain 會收到一個由 CreateWindowEx 傳回的 NULL 值，並結束這個程式。

注意，我所產生的“edit”視窗的高度和寬度皆是 0 圖素（pixel）。我之所以能夠這麼做是因爲，我總是在主視窗的視窗函式收到 WM_SIZE 訊息時（也就是在 Voyeur_OnSize 之中），調整“edit”視窗的大小。WM_SIZE 訊息發生在 WM_CREATE 訊息之後。

窺視視窗

Voyeur 容許你在主視窗工作區中按下滑鼠右鍵，不要放開，然後移動滑鼠去選擇另一個視窗。由於“edit”視窗覆蓋了主視窗的整個工作區，所以主視窗的視窗函式不會收到

WM_RBUTTONDOWN 訊息；此訊息將由 "edit" 視窗處理之。然而，當使用者在 "edit" 視窗中按下滑鼠按鍵時，系統會送出一個 WM_PARENTNOTIFY 訊息給主視窗。所以主視窗的視窗函式應該捕捉 WM_PARENTNOTIFY 訊息：

```
void Voyeur_OnParentNotify (HWND hwnd, UINT msg, HWND hwndChild, int idChild)
{

   switch (msg) {

   case WM_RBUTTONDOWN:
      // When the user clicks the secondary mouse button over the edit
      // window, the system notifies the parent window by sending the
      // WM_PARENTNOTIFY message. Once here, we enter "peer" mode.

      // Send Voyeur's window to the back of the window manager's list.
      // This causes any windows that are overlapped by Voyeur to become
      // visible, allowing the user to peer into these windows.
      SetWindowPos(hwnd, HWND_BOTTOM, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);

      // Force all mouse messages to come to this window.
      SetCapture(hwnd);

      // Change the mouse cursor to eyes to give the user a visual indication
      // that Voyeur is "peering."
      SetCursor(LoadCursor(GetWindowInstance(hwnd),
      MAKEINTRESOURCE(IDC_EYES)));

      // Set the window handle of the last viewed window to NULL.
      adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, NULL);
      break;
   }
}
```

首先我們必須確定，之所以被通知，是否因爲使用者按下滑鼠右鍵。如果是的話，我們應該呼叫 SetWindowPos，並以 HWND_BOTTOM 作爲第二個參數，將我們的視窗置於所有視窗之後。這會使得原先被覆蓋在 Voyeur 視窗之下的視窗重新出現在上頭。

Windows 系統內部保存了一份所有視窗的列表，稱爲 window manager's list。除了寬度和高度之外，也應該有一個地方用來描述視窗距離螢幕有多遠。這前後覆疊的位置稱爲視窗的 z-order，表示三維座標系統中的 z 軸。螢幕最上層視窗是在 window manager's list

的最上層部份。當要對螢幕繪出視窗，Windows 會根據 z-order 的順序（並去除覆蓋的部份）來顯示。這可以保證 window manager's list 中較接近底部的視窗不會覆蓋到較接近頂部的視窗。

當使用者點選了某個程式的標題，Windows 會將此視窗強制帶到最上層，成爲作用中的（active）視窗。當某個視窗被移到 window manager's list 的最頂部時，它的所有子視窗及被擁有（owned）的突冒式（pop-up）視窗（包括對話盒）也會被移至最頂部。視窗可以輕易改變它們在 window manager's list 中的次序 -- 只要呼叫 SetWindowPos 就可以改變 z-order。

從 Windows 3.1 開始，微軟增加了一個新的視窗擴充風格：WS_EX_TOPMOST。任何一個以此風格產生出來的視窗，總是會位於其它「未以此風格產生出來的視窗」的上面。這個風格非常有用，它可以保持 Windows 的 Help engine（求助系統）一直可見 -- 即使使用者切換程式，內含輔助文字的那個視窗仍然不會變成灰暗。Windows 同時也保證，WS_EX_TOPMOST 視窗之任何子視窗，以及擁有這種風格之 owned、pop-up 視窗，也都是 "topmost" -- 即使它們並沒有被指定 WS_EX_TOPMOST 風格[24]。

唯一能夠改變視窗之 "topmost" 狀態的，便是 SetWindowPos。這個函式能夠改變視窗的 z-order 位置，並設定或消除 WS_EX_TOPMOST 旗標。你不能企圖藉由 GetWindowLong 函式去取得目前視窗的擴充風格，然後切換 WS_EX_TOPMOST 位元，然後再利用 SetWindowLong 設定新風格。你不能夠企圖這樣子改變視窗的 "topmost" 狀態。如果你這麼做，Windows 會忽略你的請求。

SetWindowPos 的第二個參數用來改變視窗的 z-order。有幾個新的識別字被定義在 WinUser.H 之中，可做爲 SetWindowPos 的第二個參數，用以決定視窗的 z-order 位置和 "topmost" 狀態。**表 1.12** 列出了這些定義。

---

[24] 曾經在一場 Windows 3.1 研討會中，有一位發言人指出，這項新特色或許是最常被人誤用的一個。另一種選擇是讓使用者自己決定要以何種方式來使用你的程式 -- Windows Help 程式中就有一個選項，可以讓使用者決定是否要將此程式置於最上層。

表 1.12　SetWindowPos 中的識別字，可以用來改變一個視窗的 z-order。

Windows.H 中的識別字	實際數值	意義
HWND_TOP	NULL	如果這是一個最上層（topmost）視窗，它會位於其它所有最上層視窗之上。如果它不是一個最上層視窗，那麼它會位於所有「非最上層視窗」的上方。
HWND_BOTTOM	1	將視窗置於 z-order 的最底端。如果視窗是一個最上層（topmost）視窗，這個設定會使它遺失「最上層」屬性。
HWND_TOPMOST	-1	改變視窗的狀態，使它成爲一個最上層（topmost）視窗。
HWND_NOTOPMOST	-2	把視窗置於所有「非最上層視窗」之上。如果此視窗原本是個最上層（topmost）視窗，它將變成一個「非最上層」視窗。

藉由使用表中的識別字，我們便能夠將 Voyeur 視窗置於 window manager's list 的頂端或底端。Voyeur_OnParentNotify 函式中的這一行：

```
SetWindowPos(hwnd, HWND_BOTTOM, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);
```

會使得 Voyeur 主視窗和 "edit" 視窗被置於 window manager's list 的最底端。其它原先被 Voyeur 視窗所覆蓋的視窗，都會向上移一層，並且收到 WM_NCPAINT 和 WM_PAINT 訊息以便重繪視窗（SWP_NOMOVE 和 SWP_NOSIZE 旗標會導致位址和大小的參數被忽略）。

當 Voyeur 視窗被置於所有視窗之後，Voyeur_OnParentNotify 會告訴 Windows，無論滑鼠位於螢幕何處，此後所有滑鼠移動訊息皆應被送至 Voyeur_WndProc 函式中。接著，程式載入「眼睛」游標，使之成爲滑鼠的游標形狀，用以告訴使用者說 Voyeur 已經開始 "peer"（窺視）模式了。

最後，我們必須記住滑鼠最近經過的視窗的代碼。我使用附錄 A 中的技巧，利用視窗額外位元組（extra bytes）來完成。所以我需要建立一個資料結構，它有一個欄位，放置視

窗的 handle[25]。我可以使用附錄 A 的巨集來設定（或取得）額外位元組的值：

```
typedef struct {
   HWND hwndLastSubject;      // Handle for last peered window
} VOYEUR_WNDEXTRABYTES;
.
.
.
adgGETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject);
adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, NULL);
```

當父視窗收到 WM_PARENTNOTIFY 訊息，我就將這些額外位元組設爲 NULL。這是告訴 Voyeur 程式說，滑鼠尚未開始通過視窗。hwndLastSubject 內含的是 Voyeur 剛通過的視窗的代碼。這個值將在 Voyeur_OnMouseMove 和 Voyeur_OnRButtonUp 函式中使用，用來顯示或隱藏滑鼠底下的視窗外框。

選取一個視窗

預設情況下，沒有任何視窗擁有滑鼠專屬權（mouse capture），意思是系統會將滑鼠訊息送給滑鼠游標下的視窗。然而，使用 SetCapture 函式，一個視窗可以“竊取”其它視窗的所有滑鼠訊息：

```
HWND SetCapture(HWND hwnd);
```

當執行緒呼叫 SetCapture，便是告訴系統把所有滑鼠訊息送至指定的視窗中 – 不論滑鼠在螢幕何處，也不論滑鼠游標在哪一個視窗內。由於 Voyeur_OnParentNotify 呼叫了 SetCapture 並以自己的視窗代碼做爲參數，所以 Windows 的所有滑鼠訊息如 WM_RBUTTONDOWN 、 WM_RBUTTONUP 、 WM_RBUTTONDBCLK ， 以 及 WM_MOUSEMOVE 等等都將直接送至 Voyeur 主視窗。

---

[25] 我不使用 GWL_USERDATA，因爲我自己實作了這個視窗類別。記住，你應該總是把 GWL_USEDATA 位元組留給你的類別的使用者使用。既然你是這個類別的實作者，你應該很容易設定你所需要的「視窗額外位元組」才是。

一旦滑鼠捕捉權被設定，Voyeure 視窗將會收到所有 WM_MOUSEMOVE 訊息。每當收到一個 WM_MOUSEMOVE 訊息，它必須判斷滑鼠在哪個視窗上，並更新顯示內容，以反映出這個視窗及其類別的相關資訊。這一部份程式碼在 Voyeur_OnMouseMove 中。

要判斷滑鼠游標落在哪一個視窗上，我們必須取得滑鼠在螢幕上的位置。我們不能使用 Voyeur_OnMouseMove 中的 x 和 y 參數，因爲這個值是相對於 Voyeur 工作區的相對座標。我必須使用 GetMessagePos：

```
DWORD GetMessagePos(VOID);
```

這個函式傳回滑鼠游標位置，此位置是最新從執行緒訊息佇列中取出的。傳回值的較低字組（LOWORD）表示 x 座標，較高字組（HIWORD）表示 y 座標。這些座標都以整個螢幕爲座標系統。我們可以使用下列程式碼來判斷滑鼠游標位於哪一個視窗之上：

```
DWORD dwMousePos = GetMessagePos();
POINT ptMouse;
HWND hwndSubject;
.
.
.
// Get the handle of the window under the mouse cursor.
ptMouse.x = LOWORD(dwMousePos);
ptMouse.y = HIWORD(dwMousePos);
hwndSubject = WindowFromPoint(ptMouse);
```

接下來我們必須檢查看看這個視窗是不是 Voyeur 自己產生的。這樣可以避免窺視 Voyeur 本身視窗及其類別：

```
// If window is created by Voyeur, ignore it.
if (GetWindowThreadProcessId(hwndSubject, NULL) == GetCurrentThreadId())
   return;
```

如果這個視窗和我們先前的目標（視窗）一樣的話，不需做做任何動作。否則，我們必須移去先前視窗的外框，並在目前視窗之上加上一個外框。這兩個動作由 Voyeur_DrawWindowFrame 函式完成：

```
// If our new subject is the same as our last subject, there is no need to
// update our display.
if (hwndLastSubject == hwndSubject)
   return;

// Remove the frame, if any, around the currently selected window,
if (hwndLastSubject != NULL)
   VoyHelp_DrawWindowFrame(hwndLastSubject);

// Draw a frame around our new window.
VoyHelp_DrawWindowFrame(hwndSubject);
```

最後，VoyHelp_UpdateWindowInfo 函式被呼叫，將新視窗的相關資訊填入“edit”視窗。同時，我們也把新視窗的視窗代碼儲存到視窗額外位元組中：

```
// Update the window's information. This function is in the VoyHelp.C
// source file.
VoyHelp_UpdateWindowInfo(GetDlgItem(hwnd, IDC_WNDINFO), hwndSubject);

// Save the handle to the most recent subject window.
adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, hwndSubject);
```

選取視窗之後

當你放開滑鼠右鍵，你便完成了視窗的選取動作。Voyeur 如果收到 WM_RBUTTONUP 訊息，便去呼叫 Voyeur_OnRButtonUp 函式。這個函式非常的簡單，僅有一行用來取消滑鼠捕捉權：

```
void Voyeur_OnRButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {

    // We must have mouse capture or we will never get here because the edit
    // child window always covers our entire client area.

    // Allow other windows to receive mouse messages.
    ReleaseCapture();
}
```

一旦滑鼠捕捉權被取消，系統會送出 WM_ONCAPTURECHANGED 訊息給視窗：

```
// I have defined message cracker macros for the WM_CAPTURECHANGED message
// because Microsoft did not do this in the WindowsX.H header file.

/* void Cls_OnCaptureChanged(HWND hwnd, HWND hwndNewCapture) */
#define HANDLE_WM_CAPTURECHANGED(hwnd, wParam, lParam, fn) \
  ((fn)(hwnd), (HWND)(wParam)), 0L)
#define FORWARD_WM_CAPTURECHANGED(hwnd, hwndNewCapture, fn) \
  (void)(fn)((hwnd), WM_CAPTURECHANGED, (WPARAM)(HWND)(hwndNewCapture), 0L)


void Voyeur_OnCaptureChanged (HWND hwnd, HWND hwndNewCapture) {

  HWND hwndLastSubject = (HWND)
  adgGETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject);

  // If we don't "peer" into a window, we don't have to remove its
  // surrounding frame.
  if (hwndLastSubject != NULL)
    VoyHelp_DrawWindowFrame(hwndLastSubject);

  // Force Voyeur to appear on top of all other windows.
  BringWindowToTop(hwnd);
}
```

不幸地是，微軟忘了爲 WM_CAPTURECHANGED 訊息增加一個對應的訊息剖析器，所以我們必須自行建立一個，並將它放入 Voyeur.C 檔中。如果你另有程式需要處理這個訊息，可以「偷」這些巨集來使用。Voyeur_OnCaptureChanged 函式移除了「最近被滑鼠經過的視窗」的外框，並取消滑鼠捕捉權[26]。

最後一行：

```
BringWindowToTop(hwnd);
```

告訴 Windows 將 Voyeur 和 "edit" 視窗帶回到 window manager's list 的最頂端。記得在 Voyeur_OnParentNotify 函式中我們曾經使用 SetWindowPos 函式將 Voyeur 主視窗設到

[26] Windows NT 3.51 並不支援 WM_CAPTURECHANGED 訊息，但未來版本將會支援。爲了這個原因，我自行送出 WM_CAPTURECHANGED 訊息給 Voyeur 主視窗內的 Voyeur_OnRButtonUp 函式。

window manager's list 的最底端。在此，我們也可以使用 SetWindowPos 來取代 BringWindowToTop：

```
SetWindowPos(hwnd, HWND_TOP, 0, 0, 0, 0, SWP_NOMOVE｜SWP_NOSIZE);
```

事實上，這正是微軟所提供的 BringWindowToTop 函式的內部動作。

回憶一下，Voyeur_OnParentNotify 呼叫 SetCursor，爲的是將滑鼠游標變爲一對「眼睛」。然而，Voyeur_OnRButtonUp 卻沒有呼叫 SetCursor 將滑鼠游標設回一般的箭頭式樣。爲什麼？我並沒有設定滑鼠的形狀爲箭頭，爲什麼卻變回箭頭形狀了呢？原來呀，如果滑鼠未被設定捕捉權，那麼只要滑鼠在某個視窗中移動，Windows 便會送出 WM_SETCURSOR 訊息給該視窗的視窗函式。DefWindowProc 收到此訊息後會自動將滑鼠游標設回此一視窗類別所註冊的游標形狀。所以，一旦我們呼叫了 ReleaseCapture 函式，滑鼠的游標形狀會從「眼睛」改變回到原來的「箭頭」。而當視窗鎖定滑鼠的捕捉權時，系統不會送出 WM_SETCURSOR 訊息。

描繪視窗外框

Voyeur 的 Voyeur_DrawWindowFrame 函式可以描繪出視窗外框，如果呼叫第二次，可移除外框。

在視窗外圍畫框，主要的要求是這個外框必須可見。聽起來好像是廢話，但在繪圖模式下，視窗的顏色千變萬化，你要選擇什麼顏色，才能讓這個外框一定會被看到呢？最好的答案是不管視窗什麼顏色，選擇與它相反的顏色就對了。如此便能保證畫出來的外框一定看得見。好處還不只如此，如果我們在同一視窗上再畫一次，便能達到移除視窗外框的效果。所以，描繪視窗外框和移除視窗外框，只需一個函式。

VoyHelp_DrawWindowFrame 函式的程式碼如下：

```
// This function draws a frame around a given window. The frame is drawn in
// the inverse screen color, which allows a second call to this function to
// restore the screen display to its original appearance.
void VoyHelp_DrawWindowFrame (HWND hwndSubject) {
```

```
   HDC hdc;
   RECT rc;
   HPEN hpen;

   // Retrieve location of window on screen
   GetWindowRect(hwndSubject, &rc);

   // Get a device context that allows us to write anywhere within the window.
   // NOTE: GetDC would allow us to write only in the window's client area.
   hdc = GetWindowDC(hwndSubject);

   // Save the original device context attributes.
   SaveDC(hdc);

   // To guarantee that the frame will be visible, tell Windows to draw the
   // frame using the inverse screen color.
   SetROP2(hdc, R2_NOT);

   // Create a pen that is 3 times the width of a nonsizeable border. The
   // color will not be used to draw the frame, so its value could be
   // anything. PS_INSIDEFRAME tells windows that the entire frame should be
   // enclosed within the window.
   hpen = CreatePen(PS_INSIDEFRAME, 3 * GetSystemMetrics(SM_CXBORDER),
      RGB(0, 0, 0));
   SelectObject(hdc, hpen);

   // We must select a NULL brush so that the contents of the window will not
   // be overwritten.
   SelectObject(hdc, GetStockObject(NULL_BRUSH));

   // Draw the frame. Because the device context is relative to the window,
   // the left-top corner is (0, 0) and the right-bottom corner is (width of
   // window, height of window).
   Rectangle(hdc, 0, 0, rc.right - rc.left, rc.bottom - rc.top);

   // Restore the original attributes and release the device context.
   RestoreDC(hdc, -1);
   ReleaseDC(hwndSubject, hdc);

   // We can destroy the pen only AFTER we have restored the DC because the DC
   // must have valid objects selected into it at all times.
   DeleteObject(hpen);
}
```

這個函式首先取得整個視窗的螢幕座標，接下來利用 GetWindowDC 取得整個視窗的 device context（DC）。根據傳回來的這個 DC，Windows 允許我們在整個視窗範圍內繪圖。如果使用 GetDC 函式，則我們只能在視窗的工作區（client-area）中繪圖。

在畫出視窗外框之前，我們必須先指定 DC 所使用的畫筆，其顏色必須與視窗的顏色相反。這可藉由設定 ROP2 值爲 R2_NOT 來完成。然後，選擇一支粗筆使能見度較佳。指定筆的型態爲 PS_INSIDEFRAME，如此一來 Windows 才會將外框畫在視窗內緣。指定筆寬爲原來視窗邊框的 3 倍，這樣就可以在任何螢幕解析度之下顯示最佳效果。

如果你在 10,000x10,000 個圖素的螢幕解析度下執行 Windows（誰不希望如此！），畫一條高度爲 1 個圖素的線，相信誰也看不到。然而，無論你在什麼解析度下，微軟保證視窗邊框一定可以看得到。所以，根據視窗邊框而畫出的這個外框一定也可以看得見。

CreatePen 的最後一個參數用來指定畫筆顏色。由於 ROP2 的值設爲 R2_NOT，所以畫視窗外框時畫筆的顏色並無作用。因爲這個原因，此時的顏色可以是任意值。我選擇黑色。

由於 Rectangle 函式會根據 DC 所選用的筆刷來塗滿視窗，所以我們必須選用 NULL_BRUSH，於是我們可以只畫矩行外框而不塗滿內部，也就不會影響到視窗內部的資料。

函式 Rectangle 用來繪製外框。由於座標係相對於 DC，所以視窗的左上角座標是（0,0），右下角座標是視窗的寬和高。

其它函式用來釋放 DC 和清除畫筆。注意，執行順序很重要，DC 必須能夠在任何時候掌握其內的繪圖物件。如果你在釋放 DC 之前刪除畫筆，會造成錯誤。釋放 DC 之後，你才能刪除畫筆。

顯示類別資訊和視窗資訊

VoyHelp_SetClassInfo（在 VoyHelp.C 檔中）接受一個字串位址和視窗代碼，此視窗之類別資訊會填入到字串緩衝區中。 GetClassName 函式用來取得視窗的類別名稱。一旦我們有了類別名稱，便可藉由呼叫 GetClassWord 和 GetClassLong 來取得其它的類別資訊。你更可以利用 GetClassInfoEx 取而代之，會更爲方便些。我之所以不使用它的原因，是因爲先前我曾提過，GetClassInfoEx 不能獲得其它行程之執行緒所註冊的類別資訊。

VoyHelp_SetClassInfo 函式的其餘部份是將一些資訊加到字串之中。由於所有資訊最後都被放到 "edit" 視窗之中，所以 Voyeur 工作區的重繪動作就交由 "edit" 視窗來負責，程式再不需要擔心 WM_PAINT 訊息的處理。

要在 Voyeur 視窗工作區中加上 "Extra Bytes：" 資料，可使用 VoyHelp.C 中的 VoyHelp_AppendExtraBytes。此函式對「類別額外位元組」和「視窗額外位元組」都適用。如果你想取得「類別額外位元組」的內容，可在函式的最後一個參數中傳入 GCL_CBCLSEXTRA，如果你想取得「視窗額外位元組」的內容，可在函式的最後一個參數中傳入 GCL_CBWNDEXTRA。但是取得額外位元組的過程並不如我們所想像那麼直接，問題在於系統並沒有提供一個 GetClassByte 函式。所以，如果某一類別擁有 3 個「額外位元組」，呼叫：

```
GetClassWord(hwnd, 2);
```

會產生問題，因爲這個動作實際上會同時取得第三個和第四個額外位元組的值，但因爲此類別其實並沒有第四個額外位元組，所以系統會對你發出抱怨！我們必須動點手腳來告訴系統說我們並未要求其實並不存在的額外位元組。爲了解決這個問題，我們必須重複呼叫 GetClassWord，每次將其偏移值加 1，並將每次傳回的字組的 LOBYTE 或 HIBYTE 內容附加到字串中。相關細節請看程式碼。

VoyHelp_SetWindowInfo 和 VoyHelp_SetClassInfo 很類似，其大部份資訊的取得方法都是使用 GetWindowWord 和 GetWindowLong。GetWindowText 可以取得視窗標題欄文字（假如父視窗存在的話，還會取得父視窗的標題文字）[27]。視窗的執行緒 ID 和行程 ID 可使用 GetWindowThreadProcessId 函式取得，視窗的位置和大小則可使用 GetWindowRect 函式取得。

要在視窗中加上“Windows Extra Bytes :”資料，可使用 VoyHelp_AppendExtraBytes 函式並傳入 GCL_CBWNDEXTRA 作爲第三個參數。如果要加上“Wnd styles :”資料，只需使用：

```
GetWindowLong(hwnd, GWL_STYLE);
```

的傳回值中的較高 16 個位元，還記得嗎，較低 16 個位元的意義視視窗類別而定，Voyeur 無法分辨這些位元的意義。

填入 Style 相關資訊

Voyeur 可以對付三種不同的風格位元：類別風格、視窗風格和視窗擴充風格。如果我們有一個函式可以轉換這些風格位元爲對應的字串，一定很有用。

VoyHelp_AppendStyleStrings 可以接受一個字串指標、一個指向 STYLELIST 結構陣列的指標，以及一個 DWORD，用以表示要被檢查的風格位元。每一個 STYLELIST 結構都內

---

[27] 在這裡，讓我們來探討一下 GetWindowText 函式。只有當目標視窗是被呼叫端行程所產生，這個函式才會送出一個 WM_GETTEXT 訊息給該視窗。如果你呼叫 GetWindowText 並傳入一個由其他行程之執行緒產生出來的視窗代碼，作業系統內部會直接呼叫該視窗的 DefWindowProc，用以取得該視窗的標題文字。那段碼（侯俊傑註：指的是 DefWindowText）是由呼叫端執行緒執行之，而不是由產生該視窗之執行緒執行之。由於 GetWindowText 是這麼運作，所以你應該避免在你的視窗函式中攔截 WM_GETTEXT 訊息；如果你要攔截，至少你也要提供像 DefWindowProc 函式一樣的行爲 。

微軟這樣設計 GetWindowText，使得執行緒能夠取得一個視窗的標題文字 -- 甚至即使產生該視窗之執行緒處於停滯狀態。例如，工作列（Taskbar）必須總是能夠在一個按鈕中顯示一個視窗標題。

含一個風格數值，以及一個字串，如果風格旗標爲 on，該字串就應該有內容。這些陣列宣告如下：

```
typedef struct
   DWORD dwID;
   LPTSTR szName;
} STYLELIST;

#ifdef UNICODE
#define TABLEENTRY(Value) Value, L#Value
#else
#define TABLEENTRY(Value) Value, #Value
#endif

const STYLELIST g_szClassStyles[] = {
   TABLEENTRY(CS_VREDRAW),
   TABLEENTRY(CS_HREDRAW),
   .
   .
   .
   { 0, NULL }
};

const STYLELIST g_szWindowStyles[] = {
   TABLEENTRY(WS_POPUP),
   TABLEENTRY(WS_CHILD),
   .
   .
   .
   { 0, NULL }
};

const STYLELIST g_szExWindowStyles[] = {
   TABLEENTRY(WS_EX_DLGMODALFRAME),
   TABLEENTRY(WS_EX_NOPARENTNOTIFY),
   .
   .
   .
   { 0, NULL }
};
```

這種作法非常簡單而且富有彈性，適用於未來的任何 Windows 版本。不幸的是它仍然有一些缺點。舉個例子，一個 "overlapped" 視窗並沒有任何風格位元爲 on，Voyeur 本來

應該顯示 “WS_OVERLAPPED”，但它卻未做任何顯示。此外，像 WM_CAPTION 是由 WS_BORDER 和 WS_DLGFRAME 組合起來表示，結果 edit 視窗中只出現兩個組合位元的代表字串，卻沒有出現 “WM_CAPTION” 。另一個缺點是，Voyeur 無法決定視窗風格位元中的 #17 位元的意義究竟是 WS_TABSTOP 還是 WS_MAXIMIZEBOX。同樣情形也發生在 #18 位元，它不知道該解釋爲 WS_GROUP 還是 WS_MINIMIZEBOX。爲了這個原因，Voyeur 會將兩種風格都顯示出來。

Voyeur.ICO

程式列表 1.1　Voyeur.C

```
#0001  /******************************************************************
#0002  Module name: Voyeur.C
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Displays class and window information for a selected window.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single-line comment */
#0013  #include <tchar.h>
#0014  #include "Resource.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////////
#0018
#0019
#0020  // Prototype of function contained in VoyHelp.C source file.
#0021  void VoyHelp_DrawWindowFrame (HWND hwndSubject);
#0022  void VoyHelp_UpdateWindowInfo (HWND hwndEdit, HWND hwndSubject);
#0023
#0024
#0025  ////////////////////////////////////////////////////////////////////
#0026
#0027
#0028  // The following structure is for the window's extra bytes.
#0029  // For more information, see the macros presented in Appendix A.
#0030  typedef struct {
#0031     HWND hwndLastSubject;      // Handle for last peered window.
```

```
#0032 } VOYEUR_WNDEXTRABYTES;
#0033
#0034
#0035 ////////////////////////////////////////////////////////////////
#0036
#0037
#0038 // The ID given to the edit child window.
#0039 #define IDC_WNDINFO       1000
#0040
#0041
#0042 ////////////////////////////////////////////////////////////////
#0043
#0044
#0045 BOOL Voyeur_OnCreate (HWND hwnd, LPCREATESTRUCT lpCreateStruct) {
#0046
#0047    // Create the edit control that fills the client area.
#0048    HWND hwndEdit = CreateWindowEx(0, __TEXT("EDIT"),
#0049       __TEXT("Click and drag the right mouse button to select a window."),
#0050       WS_CHILD | WS_HSCROLL | WS_VSCROLL | WS_VISIBLE | ES_AUTOHSCROLL |
#0051       ES_AUTOVSCROLL | ES_LEFT | ES_MULTILINE | ES_READONLY,
#0052       0, 0, 0, 0, hwnd, (HMENU) IDC_WNDINFO, GetWindowInstance(hwnd), NULL);
#0053
#0054    adgASSERT(IsWindow(hwndEdit));
#0055    if (hwndEdit != NULL) {
#0056
#0057       // Set a tab stop at 17 characters in. There are 4 horizontal dialog
#0058       // box units per character.
#0059       UINT uTabStop = 4 * 17;
#0060       Edit_SetTabStops(hwndEdit, 1, &uTabStop);
#0061    }
#0062
#0063    // When processing the WM_CREATE message, a return value of -1 indicates
#0064    // that the window could not be created. However, when using message
#0065    // crackers, the return value from the *_OnCreate function should be FALSE
#0066    // to indicate failure and TRUE to indicate success. The message cracker
#0067    // macro translates the cracker function's return value into a return value
#0068    // for the system. In other words, if we return FALSE from this function,
#0069    // the system gets -1, and if we return TRUE from this function, the system
#0070    // gets 0.
#0071    return(hwndEdit != NULL);   // Return TRUE if edit window created.
#0072 }
#0073
#0074
#0075 ////////////////////////////////////////////////////////////////
#0076
#0077
```

```
#0078  void Voyeur_OnSize (HWND hwnd, UINT state, int cx, int cy) {
#0079
#0080     // When the user resizes the main window, we must resize the edit child.
#0081     SetWindowPos(GetDlgItem(hwnd, IDC_WNDINFO), NULL,
#0082        0, 0, cx, cy, SWP_NOZORDER);
#0083  }
#0084
#0085
#0086  ////////////////////////////////////////////////////////////////
#0087
#0088
#0089  void Voyeur_OnParentNotify (HWND hwnd, UINT msg, HWND hwndChild, int idChild) {
#0090
#0091     switch (msg) {
#0092
#0093        case WM_RBUTTONDOWN:
#0094
#0095           // When the user clicks the right mouse button over the edit window,
#0096           // the system notifies the parent window by sending the
#0097           // WM_PARENTNOTIFY message. Once here, we enter "peer" mode.
#0098
#0099           // Send Voyeur's window to the back of the window manager's list.
#0100           // This causes any windows that Voyeur overlaps to become
#0101           // visible, allowing these windows to be "peered" into.
#0102           SetWindowPos(hwnd, HWND_BOTTOM, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);
#0103
#0104           // Force all mouse messages to come to this window.
#0105           SetCapture(hwnd);
#0106
#0107           // Change the mouse cursor to eyes. This provides a visual indication
#0108           // to the user that Voyeur is "peering."
#0109           SetCursor(LoadCursor(GetWindowInstance(hwnd),
#0110              MAKEINTRESOURCE(IDC_EYES)));
#0111
#0112           // Set the window handle of the last viewed window to NULL.
#0113           adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, NULL);
#0114           break;
#0115     }
#0116  }
#0117
#0118
#0119  ////////////////////////////////////////////////////////////////
#0120
#0121
#0122  void Voyeur_OnMouseMove (HWND hwnd, int x, int y, UINT keyFlags) {
#0123
```

```
#0124    // We must have mouse capture or we would never get here because the edit
#0125    // child window always covers our entire client area.
#0126
#0127    // Get the current mouse position in screen coordinates. NOTE: We cannot
#0128    // use the x and y parameters because they are relative to the window's
#0129    // client area.
#0130    DWORD dwMousePos = GetMessagePos();
#0131    POINT ptMouse;
#0132    HWND hwndSubject;
#0133    HWND hwndLastSubject = (HWND)
#0134    adgGETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject);
#0135
#0136    // Get the handle of the window under the mouse cursor.
#0137    ptMouse.x = LOWORD(dwMousePos);
#0138    ptMouse.y = HIWORD(dwMousePos);
#0139    hwndSubject = WindowFromPoint(ptMouse);
#0140
#0141    // If the window is created by Voyeur, ignore it.
#0142    if (GetWindowThreadProcessId(hwndSubject, NULL) == GetCurrentThreadId())
#0143       return;
#0144
#0145    // If our new subject is the same as our last subject, there is no need to
#0146    // update our display.
#0147    if (hwndLastSubject == hwndSubject)
#0148       return;
#0149
#0150    // Remove the frame, if any, around the currently selected window.
#0151    if (hwndLastSubject != NULL)
#0152       VoyHelp_DrawWindowFrame(hwndLastSubject);
#0153
#0154    // Draw a frame around our new window.
#0155    VoyHelp_DrawWindowFrame(hwndSubject);
#0156
#0157    // Update the window's information. This function is in VoyHelp.C.
#0158    VoyHelp_UpdateWindowInfo(GetDlgItem(hwnd, IDC_WNDINFO), hwndSubject);
#0159
#0160    // Save the handle to the most recent subject window.
#0161    adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, hwndSubject);
#0162 }
#0163
#0164
#0165 ////////////////////////////////////////////////////////////////
#0166
#0167
#0168
#0169 void Voyeur_OnCaptureChanged (HWND hwnd, HWND hwndNewCapture) {
```

```
#0170
#0171    HWND hwndLastSubject = (HWND)
#0172    adgGETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject);
#0173
#0174    // If we don't "peer" into a window, we don't have to remove its
#0175    // surrounding frame.
#0176    if (hwndLastSubject != NULL) {
#0177       VoyHelp_DrawWindowFrame(hwndLastSubject);
#0178
#0179       // Force Voyeur to appear on top of all other windows.
#0180       BringWindowToTop(hwnd);
#0181    }
#0182
#0183    adgSETWINDOWLONG(hwnd, VOYEUR_WNDEXTRABYTES, hwndLastSubject, NULL);
#0184 }
#0185
#0186
#0187 ////////////////////////////////////////////////////////////////
#0188
#0189
#0190
#0191 void Voyeur_OnRButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {
#0192
#0193    // We must have mouse capture or we would never get here because the edit
#0194    // child window always covers our entire client area.
#0195
#0196    // Allow other windows to receive mouse messages.
#0197    ReleaseCapture();
#0198
#0199 #ifndef WINDOWSNT_COMPATIBILITY
#0200
#0201    // Because Windows NT 3.51 doesn't support the new WM_CAPTURECHANGED
#0202    // message, I must explicitly send this message to Voyeur's main window.
#0203    // Unfortunately, this means that Voyeur's main window will receive this
#0204    // message twice when running under Windows 95. I have taken care to
#0205    // ensure that the code in Voyeur_OnCaptureChanged works with no ill side
#0206    // effects if it is called twice. When future versions of Windows NT
#0207    // support the WM_CAPTURECHANGED message, the following line should be
#0208    // removed.
#0209    FORWARD_WM_CAPTURECHANGED(hwnd, NULL, SendMessage);
#0210 #endif
#0211 }
#0212
#0213
#0214 ////////////////////////////////////////////////////////////////
#0215
```

```
#0216
#0217
#0218  BOOL Voyeur_OnQueryEndSession (HWND hwnd) {
#0219
#0220     // For demonstration purposes, pretend that there is unsaved data.
#0221     BOOL fIsDataUnsaved = TRUE;
#0222     BOOL fOKToEndSession = TRUE;
#0223
#0224     if (fIsDataUnsaved) {
#0225
#0226        // Present message box.
#0227        int n = MessageBox(hwnd, __TEXT("Do you want to save changes?"),
#0228           __TEXT("Voyeur - Example technique for proper shutdown"),
#0229           MB_YESNOCANCEL | MB_ICONWARNING);
#0230
#0231        if (n == IDYES) {
#0232
#0233           // You would present a File Save dialog box here.
#0234         }
#0235
#0236        if (n == IDCANCEL)
#0237           fOKToEndSession = FALSE;
#0238     }
#0239
#0240     return(fOKToEndSession);
#0241  }
#0242
#0243
#0244  ////////////////////////////////////////////////////////////////////
#0245
#0246
#0247  void Voyeur_OnEndSession (HWND hwnd, BOOL fEnding) {
#0248
#0249     if (fEnding) {
#0250
#0251        // If the session is really ending, explicitly destroy the window.
#0252        DestroyWindow(hwnd);
#0253
#0254        // DestroyWindow will send WM_DESTROY and WM_NCDESTROY messages.
#0255     }
#0256  }
#0257
#0258
#0259  ////////////////////////////////////////////////////////////////////
#0260
#0261
```

```
#0262
#0263  void Voyeur_OnDestroy (HWND hwnd) {
#0264
#0265     // The system will automatically destroy the edit child window.
#0266
#0267     // The main window is being destroyed; signal the message loop to terminate
#0268     // so that the thread and process terminate.
#0269     PostQuitMessage(0);
#0270  }
#0271
#0272
#0273  ////////////////////////////////////////////////////////////////
#0274
#0275
#0276  void Voyeur_OnClose (HWND hwnd) {
#0277
#0278     // We handle WM_CLOSE the same way we would handle a logoff or shutdown.
#0279     BOOL fOKToClose = Voyeur_OnQueryEndSession(hwnd);
#0280     Voyeur_OnEndSession(hwnd, fOKToClose);
#0281  }
#0282
#0283
#0284  ////////////////////////////////////////////////////////////////
#0285
#0286
#0287
#0288  LRESULT WINAPI Voyeur_WndProc (HWND hwnd, UINT uMsg,
#0289     WPARAM wParam, LPARAM lParam) {
#0290
#0291     switch (uMsg) {
#0292
#0293        // Standard Windows messages.
#0294        HANDLE_MSG(hwnd, WM_CREATE,         Voyeur_OnCreate);
#0295        HANDLE_MSG(hwnd, WM_SIZE,           Voyeur_OnSize);
#0296        HANDLE_MSG(hwnd, WM_PARENTNOTIFY,    Voyeur_OnParentNotify);
#0297        HANDLE_MSG(hwnd, WM_MOUSEMOVE,      Voyeur_OnMouseMove);
#0298        HANDLE_MSG(hwnd, WM_RBUTTONUP,      Voyeur_OnRButtonUp);
#0299        HANDLE_MSG(hwnd, WM_CAPTURECHANGED,  Voyeur_OnCaptureChanged);
#0300        HANDLE_MSG(hwnd, WM_CLOSE,          Voyeur_OnClose);
#0301        HANDLE_MSG(hwnd, WM_DESTROY,        Voyeur_OnDestroy);
#0302        HANDLE_MSG(hwnd, WM_QUERYENDSESSION, Voyeur_OnQueryEndSession);
#0303        HANDLE_MSG(hwnd, WM_ENDSESSION,     Voyeur_OnEndSession);
#0304     }
#0305     return(DefWindowProc(hwnd, uMsg, wParam, lParam));
#0306  }
#0307
```

```
#0308
#0309  ////////////////////////////////////////////////////////////////
#0310
#0311
#0312  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0313    LPSTR lpszCmdLine, int nCmdShow) {
#0314
#0315    WNDCLASSEX wc;
#0316    ATOM atomClassNameVoyeur;
#0317    HWND hwndMain;
#0318    MSG msg;
#0319
#0320    adgWARNIFUNICODEUNDERWIN95();
#0321
#0322    adgINITSTRUCT(wc, TRUE);
#0323    wc.lpfnWndProc   = Voyeur_WndProc;
#0324    wc.cbWndExtra    = sizeof(VOYEUR_WNDEXTRABYTES);
#0325    wc.hInstance     = hinstExe;
#0326    wc.hIcon        = LoadIcon(hinstExe, MAKEINTRESOURCE(IDI_VOYEUR));
#0327    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
#0328    wc.lpszClassName = __TEXT("Voyeur");
#0329    wc.hIconSm       = wc.hIcon;
#0330
#0331    atomClassNameVoyeur = RegisterClassEx(&wc);
#0332    adgASSERT(atomClassNameVoyeur != INVALID_ATOM);
#0333    if (atomClassNameVoyeur == INVALID_ATOM)
#0334      return(0);
#0335
#0336    hwndMain = CreateWindowEx(0, MAKEINTATOM(atomClassNameVoyeur),
#0337      __TEXT("Voyeur"), WS_OVERLAPPEDWINDOW | WS_VISIBLE,
#0338      CW_USEDEFAULT, SW_SHOW,  // NOTE: Win32 doc bug: y is ShowWindow
#0339                               // identifier when x is CW_USEDEFAULT.
#0340      CW_USEDEFAULT, 0,       // The system sets width and height.
#0341      NULL, NULL, hinstExe, NULL);
#0342
#0343    adgASSERT(IsWindow(hwndMain));
#0344    if (IsWindow(hwndMain)) {
#0345
#0346      // Continue to loop until a WM_QUIT message comes out of the queue.
#0347      while (GetMessage(&msg, NULL, 0, 0)) {
#0348
#0349        TranslateMessage(&msg);
#0350        DispatchMessage(&msg);
#0351      }
#0352    }
#0353
```

```
#0354    UnregisterClass(MAKEINTATOM(atomClassNameVoyeur), hinstExe);
#0355
#0356    // The message box has no owner because the main window
#0357    // has been destroyed.
#0358    adgMB(__TEXT("Voyeur terminated."));
#0359
#0360    return(0);
#0361  }
#0362
#0363
#0364  ////////////////////////////// End of File //////////////////////
```

程式列表 1.2　VoyHelp.C

```
#0001  /*****************************************************************
#0002  Module name: VoyHelp.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Voyeur helper functions.
#0006  *****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single-line comment */
#0013  #include <tchar.h>
#0014
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017
#0018
#0019  // This function draws a frame around a given window. The frame is drawn in
#0020  // the inverse screen color, which allows a second call to this function to
#0021  // restore the screen display to its original appearance.
#0022  void VoyHelp_DrawWindowFrame (HWND hwndSubject) {
#0023
#0024    HDC hdc;
#0025    RECT rc;
#0026    HPEN hpen;
#0027
#0028    // Retrieve location of window on-screen.
#0029    GetWindowRect(hwndSubject, &rc);
#0030
#0031    // Get a device context that allows us to write anywhere within the window.
```

```
#0032    // NOTE: GetDC would allow us to write only in the window's client area.
#0033    hdc = GetWindowDC(hwndSubject);
#0034
#0035    // Save the original device context attributes.
#0036    SaveDC(hdc);
#0037
#0038    // To guarantee that the frame will be visible, tell Windows to draw the
#0039    // frame using the inverse screen color.
#0040    SetROP2(hdc, R2_NOT);
#0041
#0042    // Create a pen that is three times the width of a nonsizeable border. The
#0043    // color will not be used to draw the frame, so its value could be
#0044    // anything. PS_INSIDEFRAME tells windows that the entire frame should be
#0045    // enclosed within the window.
#0046    hpen = CreatePen(PS_INSIDEFRAME, 3 * GetSystemMetrics(SM_CXBORDER),
#0047       RGB(0, 0, 0));
#0048    SelectObject(hdc, hpen);
#0049
#0050    // We must select a NULL brush so that the contents of the window will not
#0051    // be overwritten.
#0052    SelectObject(hdc, GetStockObject(NULL_BRUSH));
#0053
#0054    // Draw the frame. Because the device context is relative to the window,
#0055    // the top-left corner is (0, 0) and the lower right corner is (width of
#0056    // window, height of window).
#0057    Rectangle(hdc, 0, 0, rc.right - rc.left, rc.bottom - rc.top);
#0058
#0059    // Restore the original attributes and release the device context.
#0060    RestoreDC(hdc, -1);
#0061    ReleaseDC(hwndSubject, hdc);
#0062
#0063    // We can destroy the pen only AFTER we have restored the DC because the DC
#0064    // must have valid objects selected into it at all times.
#0065    DeleteObject(hpen);
#0066 }
#0067
#0068
#0069
#0070 ////////////////////////////////////////////////////////////////
#0071
#0072
#0073 typedef struct {
#0074    DWORD dwID;
#0075    LPTSTR szName;
#0076 } STYLELIST;
#0077
```

```
#0078
#0079 ////////////////////////////////////////////////////////////////
#0080
#0081
#0082 #ifdef UNICODE
#0083 #define TABLEENTRY(Value) Value, L#Value
#0084 #else
#0085 #define TABLEENTRY(Value) Value, #Value
#0086 #endif
#0087
#0088
#0089 ////////////////////////////////////////////////////////////////
#0090
#0091
#0092 const STYLELIST g_szClassStyles[] = {
#0093    TABLEENTRY(CS_VREDRAW),
#0094    TABLEENTRY(CS_HREDRAW),
#0095    TABLEENTRY(CS_KEYCVTWINDOW),
#0096    TABLEENTRY(CS_DBLCLKS),
#0097    TABLEENTRY(CS_OWNDC),
#0098    TABLEENTRY(CS_CLASSDC),
#0099    TABLEENTRY(CS_PARENTDC),
#0100    TABLEENTRY(CS_NOKEYCVT),
#0101    TABLEENTRY(CS_NOCLOSE),
#0102    TABLEENTRY(CS_SAVEBITS),
#0103    TABLEENTRY(CS_BYTEALIGNCLIENT),
#0104    TABLEENTRY(CS_BYTEALIGNWINDOW),
#0105    TABLEENTRY(CS_GLOBALCLASS),
#0106    { 0, NULL }
#0107 };
#0108
#0109 const STYLELIST g_szWindowStyles[] = {
#0110    TABLEENTRY(WS_POPUP),
#0111    TABLEENTRY(WS_CHILD),
#0112    TABLEENTRY(WS_MINIMIZE),
#0113    TABLEENTRY(WS_VISIBLE),
#0114    TABLEENTRY(WS_DISABLED),
#0115    TABLEENTRY(WS_CLIPSIBLINGS),
#0116    TABLEENTRY(WS_CLIPCHILDREN),
#0117    TABLEENTRY(WS_MAXIMIZE),
#0118    TABLEENTRY(WS_BORDER),
#0119    TABLEENTRY(WS_DLGFRAME),
#0120    TABLEENTRY(WS_VSCROLL),
#0121    TABLEENTRY(WS_HSCROLL),
#0122    TABLEENTRY(WS_SYSMENU),
#0123    TABLEENTRY(WS_THICKFRAME),
```

```
#0124    TABLEENTRY(WS_GROUP),
#0125    TABLEENTRY(WS_TABSTOP),
#0126    { 0, NULL }
#0127  };
#0128
#0129  const STYLELIST g_szExWindowStyles[] = {
#0130    TABLEENTRY(WS_EX_DLGMODALFRAME),
#0131    TABLEENTRY(WS_EX_NOPARENTNOTIFY),
#0132    TABLEENTRY(WS_EX_TOPMOST),
#0133    TABLEENTRY(WS_EX_ACCEPTFILES),
#0134    TABLEENTRY(WS_EX_TRANSPARENT),
#0135    { 0, NULL }
#0136  };
#0137
#0138
#0139  ////////////////////////////////////////////////////////////
#0140
#0141
#0142
#0143  // This function appends the text names of the styles to a string buffer. It
#0144  // is used for the class styles, window styles, and extended window styles.
#0145  void VoyHelp_AppendStyleStrings (LPTSTR szBuf, const STYLELIST Styles[],
#0146    DWORD dwStyleFlags) {
#0147
#0148    int nStyleIndex, nNumStyles = 0;
#0149
#0150    _tcscat(szBuf, __TEXT(" ")); // Start with an open paren.
#0151
#0152    for (nStyleIndex = 0; Styles[nStyleIndex].szName != NULL; nStyleIndex++) {
#0153
#0154       // If style bit is set, append style text to the string.
#0155       if (Styles[nStyleIndex].dwID & dwStyleFlags) {
#0156
#0157          // If this is not the first style, preface the string with ", ".
#0158          if (nNumStyles++ > 0)
#0159             _tcscat(szBuf, __TEXT(", "));
#0160
#0161          _tcscat(szBuf, Styles[nStyleIndex].szName);
#0162       }
#0163    }
#0164
#0165    // If no style string was appended, say so.
#0166    if (nNumStyles == 0)
#0167       _tcscat(szBuf, __TEXT("none"));
#0168
#0169    _tcscat(szBuf, __TEXT(")\r\n")); // End with a close paren.
```

```
#0170 }
#0171
#0172
#0173 ////////////////////////////////////////////////////////////////
#0174
#0175
#0176 void VoyHelp_AppendExtraBytes (LPTSTR szBuf, HWND hwndSubject,
#0177    int nExtraBytesID) {
#0178
#0179    int nExtraByteNum = 0;
#0180    int nExtraBytes = GetClassLong(hwndSubject, nExtraBytesID);
#0181    BYTE bByte;
#0182
#0183    WORD (WINAPI* pfnGetWord)(HWND hwnd, int nOffset) =
#0184       (nExtraBytesID == GCL_CBCLSEXTRA) ? GetClassWord : GetWindowWord;
#0185
#0186    wsprintf(_tcschr(szBuf, 0), __TEXT("%s extra bytes:\t%u ("),
#0187       (nExtraBytesID == GCL_CBCLSEXTRA) ? __TEXT("Class") : __TEXT("Window"),
#0188       nExtraBytes);
#0189
#0190    // Append the extra byte values to the string.
#0191    for (; nExtraByteNum < nExtraBytes; nExtraByteNum++) {
#0192
#0193       if (nExtraByteNum == 0) {
#0194
#0195          bByte = LOBYTE(pfnGetWord(hwndSubject, nExtraByteNum));
#0196       } else {
#0197
#0198          // Put a leading space after the first extra byte.
#0199          _tcscat(szBuf, __TEXT(" "));
#0200
#0201          // This is necessary to stay within the extra bytes.
#0202          bByte = HIBYTE(pfnGetWord(hwndSubject, nExtraByteNum - 1));
#0203       }
#0204       wsprintf(_tcschr(szBuf, 0), __TEXT("0x%02x"), bByte);
#0205    }
#0206
#0207    if (nExtraByteNum == 0) {
#0208
#0209       // Put none if there are no extra bytes.
#0210       _tcscat(szBuf, __TEXT("none"));
#0211    }
#0212
#0213    _tcscat(szBuf, __TEXT(")\r\n"));
#0214 }
#0215
```

```
#0216
#0217 ////////////////////////////////////////////////////////////
#0218
#0219
#0220 // This appends the class information about the passed-in window
#0221 // (hwndSubject) to the passed-in string.
#0222 void VoyHelp_SetClassInfo (LPTSTR szBuf, HWND hwndSubject) {
#0223
#0224    TCHAR szClassName[100] = { 0 };
#0225
#0226    // Put a heading at the beginning of this section.
#0227    _tcscat(szBuf, __TEXT("********* CLASS INFORMATION *********\r\n"));
#0228
#0229    // Get the class name of the window.
#0230    GetClassName(hwndSubject, szClassName, adgARRAY_SIZE(szClassName));
#0231
#0232    // NOTE: We must use GetClassWord/Long here instead of GetClassInfoEx
#0233    // because GetClassInfoEx does not work for classes that are registered by
#0234    // another process.
#0235    wsprintf(_tcschr(szBuf, 0),
#0236       __TEXT("Class name:\t%s\r\n")
#0237       __TEXT("Atom:\t0x%04x\r\n")
#0238       __TEXT("hInstance:\t0x%08x\r\n")
#0239       __TEXT("lpfnWndProc:\t0x%08x\r\n")
#0240       __TEXT("hIcon:\t0x%08x\r\n")
#0241       __TEXT("hIconSm:\t0x%08x\r\n")
#0242       __TEXT("hCursor:\t0x%08x\r\n")
#0243       __TEXT("hbrBackground:\t0x%08x\r\n")
#0244       __TEXT("lpszMenuName:\t0x%08x\r\n")
#0245       __TEXT("Class styles:\t0x%08x"),
#0246       szClassName,                              // Class name
#0247       GetClassWord(hwndSubject, GCW_ATOM),         // Atom
#0248       GetClassLong(hwndSubject, GCL_HMODULE),       // hinst of registerer
#0249       GetClassLong(hwndSubject, GCL_WNDPROC),       // Window procedure
#0250       GetClassLong(hwndSubject, GCL_HICON),        // Handle of icon
#0251       GetClassLong(hwndSubject, GCL_HICONSM),       // Handle of small icon
#0252       GetClassLong(hwndSubject, GCL_HCURSOR),       // Handle of cursor
#0253       GetClassLong(hwndSubject, GCL_HBRBACKGROUND),  // Handle of brush
#0254       GetClassLong(hwndSubject, GCL_MENUNAME),      // Menu name
#0255       GetClassLong(hwndSubject, GCL_STYLE));       // Class-style flags
#0256
#0257    VoyHelp_AppendStyleStrings(szBuf, g_szClassStyles,
#0258    GetClassLong(hwndSubject, GCL_STYLE));
#0259
#0260    VoyHelp_AppendExtraBytes(szBuf, hwndSubject, GCL_CBCLSEXTRA);
#0261
```

```
#0262    _tcscat(szBuf, __TEXT("\r\n"));
#0263 }
#0264
#0265
#0266 ////////////////////////////////////////////////////////////////
#0267
#0268
#0269 // This function is called by the system when we call EnumPropsEx in the
#0270 // following SetWindowInfo function.
#0271 BOOL WINAPI VoyHelp_PropEnumProcEx (HWND hwnd, LPTSTR lpszString,
#0272    HANDLE hData, DWORD dwData) {
#0273
#0274    LPTSTR szBuf = (LPTSTR) dwData;
#0275    if ((DWORD) lpszString <= 0x0000FFFF) {
#0276       // The property is an atom
#0277       wsprintf(_tcschr(szBuf, 0), __TEXT("    Atom %04x = 0x%08x\r\n"),
#0278          lpszString, hData);
#0279    } else {
#0280       // The property is a string
#0281       wsprintf(_tcschr(szBuf, 0), __TEXT("    %s = 0x%08x\r\n"),
#0282          lpszString, hData);
#0283    }
#0284    return(TRUE);               // Continue enumerating properties
#0285 }
#0286
#0287
#0288 ////////////////////////////////////////////////////////////////
#0289
#0290
#0291
#0292 // This appends the window instance information about the passed-in window
#0293 // (hwndSubject) to the passed-in string.
#0294 void VoyHelp_SetWindowInfo (LPTSTR szBuf, HWND hwndSubject) {
#0295
#0296    TCHAR szWndText[100], szWndPrntText[100];
#0297    HWND hwndParent = (HWND) GetWindowLong(hwndSubject, GWL_HWNDPARENT);
#0298    DWORD dwProcessID;
#0299    RECT rc;
#0300
#0301    // Put a heading at the beginning of this section.
#0302    _tcscat(szBuf,
#0303    __TEXT("********* WINDOW INSTANCE INFORMATION *********\r\n"));
#0304
#0305    // Get text of subject window.
#0306    if (GetWindowText(hwndSubject, szWndText, adgARRAY_SIZE(szWndText)) == 0)
#0307       _tcscpy(szWndText, __TEXT("(no caption)"));
```

```
#0308
#0309    // Get text of subject window's parent window.
#0310    if (IsWindow(hwndParent)) {
#0311
#0312       if (GetWindowText(hwndParent, szWndPrntText,
#0313          adgARRAY_SIZE(szWndPrntText)) == 0) {
#0314
#0315          _tcscpy(szWndPrntText, __TEXT("(no caption)"));
#0316       }
#0317    } else {
#0318
#0319       _tcscpy(szWndPrntText, __TEXT("(no parent)"));
#0320    }
#0321
#0322    GetWindowRect(hwndSubject, &rc);
#0323    GetWindowThreadProcessId(hwndSubject, &dwProcessID);
#0324
#0325    // NOTE: The WndProc is not shown because Windows always returns NULL when
#0326    // a thread in one process queries GWL_WNDPROC for a window created by a
#0327    // thread in another process.
#0328    wsprintf(_tcschr(szBuf, 0),
#0329       __TEXT("Window:\t0x%08x - %s\r\n")
#0330       __TEXT("hInstance:\t0x%08x\r\n")
#0331       __TEXT("Parent window:\t0x%08x - %s\r\n")
#0332       __TEXT("Thread ID:\t0x%08x\r\n")
#0333       __TEXT("Process ID:\t0x%08x\r\n")
#0334       __TEXT("Menu/ID:\t0x%08x (%d)\r\n")
#0335       __TEXT("User data:\t0x%08x (%d)\r\n")
#0336       __TEXT("Rectangle:\t(%d, %d)-(%d, %d), Dim=%d x %d\r\n")
#0337       __TEXT("Window styles:\t0x%08x"),
#0338       hwndSubject, szWndText,                    // Handle & caption
#0339       GetWindowLong(hwndSubject, GWL_HINSTANCE),  // Creator's hInstance
#0340       hwndParent, szWndPrntText,                  // Parent handle & caption
#0341       GetWindowThreadProcessId(hwndSubject, NULL), // Creating thread's ID
#0342       dwProcessID,                               // Process ID owning thread
#0343       GetWindowLong(hwndSubject, GWL_ID),        // HMENU/ID
#0344       GetWindowLong(hwndSubject, GWL_ID),
#0345       GetWindowLong(hwndSubject, GWL_USERDATA),   // User data
#0346       GetWindowLong(hwndSubject, GWL_USERDATA),
#0347       rc.left, rc.top, rc.right, rc.bottom,       // Rectangle
#0348       rc.right - rc.left, rc.bottom - rc.top,     // Dimensions
#0349       GetWindowLong(hwndSubject, GWL_STYLE));     // Styles
#0350
#0351    VoyHelp_AppendStyleStrings(szBuf, g_szWindowStyles,
#0352       GetWindowLong(hwndSubject, GWL_STYLE));
#0353
```

```
#0354    wsprintf(_tcschr(szBuf, 0), __TEXT("Extended styles:\t0x%08x "),
#0355       GetWindowLong(hwndSubject, GWL_EXSTYLE));
#0356
#0357    VoyHelp_AppendStyleStrings(szBuf, g_szExWindowStyles,
#0358       GetWindowLong(hwndSubject, GWL_EXSTYLE));
#0359
#0360    VoyHelp_AppendExtraBytes(szBuf, hwndSubject, GCL_CBWNDEXTRA);
#0361
#0362    _tcscat(szBuf, __TEXT("Properties:\r\n"));
#0363    if (EnumPropsEx(hwndSubject, VoyHelp_PropEnumProcEx, (LONG) szBuf) == -1) {
#0364
#0365       // The window has no properties.
#0366       _tcscat(szBuf, __TEXT("   (none)\r\n"));
#0367    }
#0368
#0369    _tcscat(szBuf, __TEXT("\r\n"));
#0370 }
#0371
#0372
#0373 ////////////////////////////////////////////////////////////////
#0374
#0375
#0376 void VoyHelp_UpdateWindowInfo (HWND hwndEdit, HWND hwndSubject) {
#0377
#0378    TCHAR szBuf[8192] = { 0 };   // A very big string buffer
#0379
#0380    // Add the class information to the string.
#0381    VoyHelp_SetClassInfo(szBuf, hwndSubject);
#0382
#0383    // Add the window instance information to the string.
#0384    VoyHelp_SetWindowInfo(szBuf, hwndSubject);
#0385
#0386    // Update the edit class window with the subject's info.
#0387    SetWindowText(hwndEdit, szBuf);
#0388 }
#0389
#0390 ////////////////////////////////// End of File ////////////////////
```

Eyes.cur

程式列表 1.3　Voyeur.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "resource.h"
#0004
#0005 #define APSTUDIO_READONLY_SYMBOLS
```

```
#0006  //////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  //////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  //////////////////////////////////////////////////////////////////
#0017  //
#0018  // Icon
#0019  //
#0020
#0021  IDI_VOYEUR            ICON    DISCARDABLE     "VOYEUR.ICO"
#0022
#0023  #ifdef APSTUDIO_INVOKED
#0024  //////////////////////////////////////////////////////////////////
#0025  //
#0026  // TEXTINCLUDE
#0027  //
#0028
#0029  1 TEXTINCLUDE DISCARDABLE
#0030  BEGIN
#0031      "resource.h\0"
#0032  END
#0033
#0034  2 TEXTINCLUDE DISCARDABLE
#0035  BEGIN
#0036      "#include ""windows.h""\r\n"
#0037      "\0"
#0038  END
#0039
#0040  3 TEXTINCLUDE DISCARDABLE
#0041  BEGIN
#0042      "\r\n"
#0043      "\0"
#0044  END
#0045
#0046  //////////////////////////////////////////////////////////////////
#0047  #endif    // APSTUDIO_INVOKED
#0048
#0049
#0050  //////////////////////////////////////////////////////////////////
#0051  //
```

```
#0052  // Cursor
#0053  //
#0054
#0055  IDC_EYES             CURSOR  DISCARDABLE    "EYES.CUR"
#0056
#0057  #ifndef APSTUDIO_INVOKED
#0058  ////////////////////////////////////////////////////////////////
#0059  //
#0060  // Generated from the TEXTINCLUDE 3 resource.
#0061  //
#0062
#0063
#0064  ////////////////////////////////////////////////////////////////
#0065  #endif    // not APSTUDIO_INVOKED
```

程式列表 1.4　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by VOYEUR.RC
#0004  //
#0005  #define IDI_VOYEUR                  102
#0006  #define IDC_EYES                    103
#0007
#0008  // Next default values for new objects
#0009  //
#0010  #ifdef APSTUDIO_INVOKED
#0011  #ifndef APSTUDIO_READONLY_SYMBOLS
#0012  #define _APS_NEXT_RESOURCE_VALUE       104
#0013  #define _APS_NEXT_COMMAND_VALUE        40001
#0014  #define _APS_NEXT_CONTROL_VALUE        1000
#0015  #define _APS_NEXT_SYMED_VALUE          101
#0016  #endif
#0017  #endif
```

第２章

詳細剖析對話盒

對話盒（Dialog boxes）是程式與使用者之間最好的溝通橋樑。Visual C++ 提供了一個對話盒編輯器（dialog editor），讓程式員可以很輕易地設計出一個漂亮的對話盒。它減輕了程式員在計算視窗顯現位置等瑣事上的負擔。終端客戶也可使用此對話盒編輯器來設計一些漂亮的程式介面，再交給程式員來開發程式。

這一章我要探討對話盒內部的工作。特別是討論什麼是對話盒、如何爲對話盒建立面板（template），以及系統如何操控它。本章是下一章的基礎，下一章之中，我們將探討如何利用對話盒製作一些商用軟體。

對話盒就是視窗

你最需要知道的一個觀念就是，對話盒是個視窗，沒有什麼神奇的。只不過這個視窗通常包含了一些子視窗（一般稱爲控制元件，controls），當然它也可以全然不需任何子視窗。由於許多程式都在視窗之中再含子視窗，所以微軟添增了許多 WIN32 API 函式，用來處理視窗的型式。有許多開發人員把這一組函式稱爲 “Windows’ Dialog Box Manager”。有一個像 “Windows’ Dialog Box Manager” 這樣的東西彷彿會使對話盒比一般視窗更富麗堂皇，因此，我比較喜歡的說法是，其實並沒有什麼所謂的 “Windows’ Dialog Box Manager”，那只是一組讓我們更容易操作子視窗的函式而已。

後續各節中，我將檢驗一些用來處理「視窗之中有子視窗（也就是對話盒）」的 WIN32 函式。注意，有許多與對話盒相關的 WIN32 函式，其名稱都含有 "Dlg"，例如 GetDlgItem 函式：

```
HWND GetDlgItem(HWND hDlg, int nID);
```

函式名稱含有 "Dlg"，暗示著此函式僅在對話盒 -- 也就是以 DialogBoxParam 或 CreateDialogParam（或其他類似函式）所產生的對話盒 -- 中工作。但其實 "Dlg" 會帶來誤導作用，因爲這類函式所做的，就是去搜尋比對父視窗內的哪一個子視窗的 ID 和參數 nID 相同。GetDlgItem 函式其實際也可以在有 父 / 子 視窗關係的任何環境下使用，並不侷限於對話盒。幾乎所有似乎只能用於對話盒的 WIN32 函式，其實都是如此，你可以在你希望的視窗中任意使用那些函式。如果微軟把 GetDlgItem 改爲下面這樣，或許會更好些：

```
HWND GetChildWindowFromID(HWND hwndParent, int nIDChild);
```

設計對話盒的外觀

當你在 Windows 95 的「開始」選單中選擇「執行…」項目，螢幕上會顯示如圖 2.1 的對話盒。

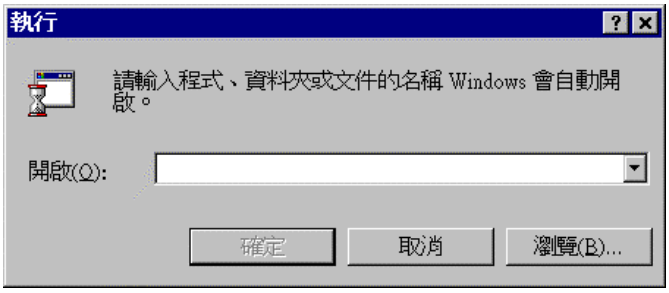


圖 2.1 「執行」對話盒

這個父視窗包含了七個子視窗，顯示於圖 2.2 中；所有子視窗的特性簡述於表 2.1 中。

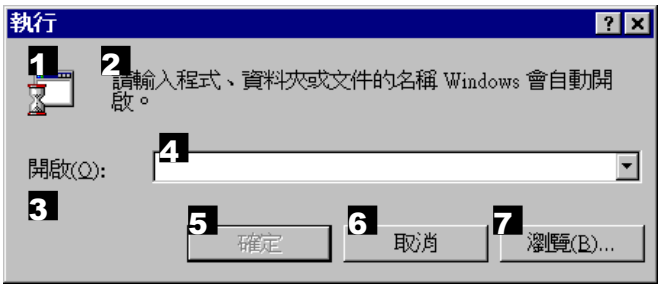


圖 2.2　「執行」對話盒，及其子視窗

表 2.1　「執行」對話盒的子視窗

#	視窗類別	說明
1	Static	內含圖示（icon），顯示於左上角。
2	Static	內含「請輸入程式…」的文字。
3	Static	內含「開啓(O)：」的文字。
4	ComboBox	內含過去所執行過的程式、資料夾和文件。
5	Button	【確定（OK）】按鈕。
6	Button	【取消（Cancel）】按鈕。
7	Button	【瀏覽（Browse）】按鈕。

如果不使用那些方便的 WIN32 函式來幫助我們產生這樣一個視窗，我們首先得呼叫一次 CreateWindowEx，產生一個父視窗，然後再呼叫七次 CreateWindowEx，產生七個子視窗。呼叫七次 CreateWindowEx 並不算太糟，特別是如果你把這些呼叫放在迴圈之中。但問題是我們必須爲每次呼叫指定正確的視窗類別、標題文字、視窗風格、x 座標、y 座標、視窗寬高…等等等。如果這些值（尤其是手動調整 x 座標、y 座標、寬度和高度）都要由我們決定的話，我們將會花費不少時間在設計子視窗的外觀上。

所以，爲了幫助我們設計這些子視窗，Visual C++ 提供了一個便利的工具，稱爲對話盒編輯器（dialog box editor）。這個編輯器允許我們選擇不同的子視窗類別，並使用拖放（drag-and-drop）方式將子視窗安置在父視窗上。你可以輕易移動子視窗並調整其寬度和

高度。對於 static 和 button 等的視窗類別，你甚至可以在設計期間就指定其正確的文字，不需要先設定一些特定的文字來替代。

每當你設計完你的對話盒，Visual C++ 會將對話盒的面板（template）寫入你的 RC 檔（資源檔）中。**圖 2.1** 的對話盒面板寫入資源檔的內容如下：

```
IDD_RUN DIALOG DISCARDABLE  0, 0, 230, 88
STYLE WS_POPUP | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CAPTION | WS_SYSMENU
CAPTION "Run"
FONT 8,"MS Sans Serif"
BEGIN
     ICON      "",IDC_ICON,8,10,18,20
     LTEXT     "Type the name of a program, folder, or document, and
Windows will open it for you.", IDC_STATIC,36,10,186,20
     LTEXT          "&Open:",IDC_STATIC,8,40,20,8
     COMBOBOX       IDC_RUNHISTORY,36,38,186,30,CBS_DROPDOWN |
                    CBS_AUTOHSCROLL | CBS_DISABLENOSCROLL | WS_TBSTOP
     DEFPUSHBUTTON   "OK",IDOK,64,64,50,14
     PUSHBUTTON     "Cancel",IDCANCEL,116,64,52,14
     PUSHBUTTON     "&Browse...",IDC_BROWSE,170,64,50,14
END
```

對話盒面板以文字描述，不能被任何 WIN32 函式處理。它必須先編譯成爲二進位形式，這個過程由資源編譯器（resource compiler）負責。資源編譯器會剖析這些文字的描述並產生二進位碼。在 Windows 95 和 Windows NT 3.51 中，微軟定義了一個新的對話盒面板：DIALOGEX，是原先的 DIALOG 的一個超集，格式如下：

```
DialogTemplateName DIALOGEX x, y, cx, cy, helpID
STYLE style
EXSTYLE exStyle
FONT height, name, weight, italic
CAPTION captionText
MENU menuName
CLASS className
LANGUAGE languageID
VERSION versionNumber
CHARACTERISTICS characteristicsValue
BEGIN
   CONTROL controlText, id, className, style,
```

```
     x, y, cx, cy, exStyle, helpID
     BEGIN
        data-element-1,
        ...
     END
   ...
END
```

本章後續各節中，我將使用這個新的 DIALOGEX 面板及其產生出來的二進位結構。不幸的是，Visual C++ 2.x 版的資源編譯器並不支援這個新的 DIALOGEX 面板。不過 WIN32 SDK 的資源編譯器倒是可以編譯這個新的 DIALOGEX 面板，並產生二進位碼，內含一個 DLGTEMPLATEEX 結構、一個可有可無的 FONTINFOEX 結構，以及零個或多個 DLGITEMTEMPLATEEX 結構。

這些新的 *EX 結構有以下特色：

- 可以指定對話盒及其子視窗的 Help IDs。
- 可以指定對話盒及其子視窗的擴充風格（extended window style）。
- 可以設定字形的種類與級數（即字型大小）。
- 可以將與控制元件有關的資料指定給子視窗。

舊的 DLGTEMPLATE 和 DLGITEMTEMPLATE 結構定義在 WinUser.H 標頭檔中。微軟並沒有定義舊的 FONTINFO 結構，或任何新的擴充結構，這製成我們在處理對話盒時有些不便。但是大部份時候我們並不需去處理這些資料結構。現在我們來探討這些新的對話盒結構。就從 DLGTEMPLATEEX 結構開始好了：

```
typedef struct { // dlttex
  WORD   wDlgVer;       // Always 1
  WORD   wSignature;    // Always 0xFFFF
  DWORD  dwHelpID;
  DWORD  dwExStyle;     // Such as WS_EX_TOPMOST
  DWORD  dwStyle;       // Such as WS_CAPTION
  WORD   cDlgItems;
  short  x;         // In pixels
  short  y;         // In pixels
```

```
   short   cx;         // In dialog box units
   short   cy;         // In dialog box units
   BYTE    bStartOfStrings;
   // Zero-terminated Unicode string for the menu name;
   // Zero-terminated Unicode string for the class name;
   // Zero-terminated Unicode string for the window title;
} DLGTEMPLATEEX, *PDLGTEMPLATEEX;
```

這個結構定義出父視窗的屬性。wSignature 欄位總是爲 0xFFFF。當剖析對話盒面板的二進位碼時，系統會先檢查這個欄位的值。如果是 0xFFFF，系統便知道這個面板是新的 DLGTEMPLATEEX 結構，而不是舊的 DLGTEMPLATE 結構。一旦系統知道這是一個新的 DLGTEMPLATEEX 結構，接下來便檢查 wDlgVer 欄位，看看是哪一種型式的 DLGTEMPLATEEX 結構。目前只存在一種型式，所以 wDlgVer 欄位值爲 1。這種設計使微軟在未來版本的 Windows 作業系統中，較容易擴充 DLGTEMPLATEEX 結構，彈性較大。

cDlgItems 欄位表示了子視窗的個數。注意，選單名稱、類別、名稱以及視窗標題都以註釋的方式出現在結構之中，因爲這些字串是不定長度且需以 '\0' 結尾。當資源編譯器產生此結構之二進位碼時，這些字串會被直接附加在 DLGTEMPLATEEX 結構之後，用的是 Unicode[1]字串而不是 ANSI 字串。Windows 95 也是如此。這很合理不是嗎？畢竟一個 Intel Win32 可執行檔不需重新編譯就可以在 Windows 95 和 Windows NT 下執行。在 Windows 95 中，作業系統產生任何視窗之前會先讀取這些字串並將 Unicode 字串轉換成 ANSI 字串。

如果 DLGTEMPLATEEX 結構的 style 欄位的 DS_SETFONT 位元是 on 的話，會有一個 FONTINFOEX 結構直接附於視窗標題之後，格式如下：

```
typedef struct {
   short   nPointSize;
   short   nWeight;     // Such as FW_NORMAL
   short   fItalic;     // TRUE or FALSE
```

[1] 如果想多瞭解 Unicode，請參閱 ***Win32 Programmers Reference*** 和我的 ***Advance Windows*** 一書。

```
   BYTE    bStartOfStrings;
   // Zero-terminated Unicode string for the font name;
} FONTINFOEX, *PFONTINFOEX;
```

這個結構允許你爲對話盒子視窗選擇一個特定字型。現在，除了描述父視窗的資訊外，尙有零個或多個 DLGITEMTEMPLATEEX 結構，用來描述每一個子視窗：

```
typedef struct {
   DWORD   dwHelpID;
   DWORD   dwExStyle;   // Such as WS_EX_CONTROLPARENT
   DWORD   dwStyle;     // Such as WS_TABSTOP
   short   x;       // In dialog box units
   short   y;       // In dialog box units
   short   cx;      // In dialog box units
   short   cy;      // In dialog box units
   DWORD   id;
   BYTE    bStartOfStrings;
   // Zero-terminated Unicode string for the class name;
   // Zero-terminated Unicode string for the control title;
   WORD    wExtraCount; // Usually 0
   // wExtraCount bytes of raw data;
 } DLGITEMTEMPLATEEX, *PDLGITEMTEMPLATEEX;
```

這個資料結構和 DLGTEMPLATEEX 結構非常類似。各個欄位用來描述子視窗的屬性。DLGITEMTEMPLATEEX 結構以一個 id 欄位取代 DLGTEMPLATEEX 結構中的選單欄位（一個以‘0’爲結束字元的字串），這是因爲子視窗不能擁有選單。事實上，CreateWindowEx 函式的技術文件曾指出，當產生的對象是子視窗時，hMenu 參數應該被視爲一個整數 ID。上述 id 欄位所代表的，正是你指定給每一個子視窗的整數 ID。它使你操控子視窗比較容易些。

讓我舉個例子。你可能會想送一個 LB_ADDSTRING 訊息給一個 listbox。爲了做到這個動作，你必須呼叫 SendMessage，並事先取得此 listbox 的代碼（handle）。然而，執行時期你沒有辦法得知 listbox 的代碼是什麼。因此，你可以在對話盒編輯器中指定一個 ID 給 listbox，此一 ID 始終都是一樣的，於是你便可以使用 SendDlgItemMessage 送出 LB_ADDSTRING 訊息了：

```
LRESULT SendDlgItemMessage(HWND hwndParent, int nID, UNITuMsg,
```

```
WPARAM wParam, LPARAM lParam);
```

第一個參數表示父視窗代碼，第二個參數 nID 就是你指定給子視窗的整數 ID。當你呼叫 SendDlgItemMessage，它會比對父視窗中的所有子視窗 ID，直到發現某個子視窗 ID 與參數 nID 相同。一旦找到，此函式便得知這個子視窗的視窗代碼，於是就可以呼叫 SendMessage 將訊息送出。這又是另一個「函式名稱中含有 "Dlg" 但可以在對話盒以外工作」的例子。是的，你可以使用 SendDlgItemMessage 將訊息傳送給子視窗，不管父視窗是以何種方式產生的。

緊隨在 id 欄位之後的是一些可變長度的欄位。前兩個都是以 '\0' 爲結尾字元的 Unicode 字串，分別表示子視窗類別名稱和子視窗標題。視窗標題之後緊接著是視窗的建立資訊，這些資訊由兩部份組成：一個 2 位元組數值，表示隨後有多少（位元組）的資料，然後是真正的資料內容。這些資料並不常被使用，所以我不打算詳細討論。

一旦 DLGTEMPLATEEX 結構、可有可無的 FONTINFOEX 結構、以及所有必要的 DLGITEMTEMPLATEEX 結構都被資源編譯器編譯爲二進位碼之後，會被放進你的 EXE 檔中。現在，爲了產生所有這些視窗，你必須使用一個 WIN32 函式將你的面板放到你的 EXE 檔中。下一節我們來討論這些函式。

對話視窗和子視窗

當我想要產生一個對話盒，我必須先取得一塊記憶體，存放 DLGTEMPLATEEX、FONTINFOEX 和 DLGITEMTEMPLATEEX 結構。通常我會呼叫 CreateDialogParam[2]：

```
HWND CreateDialogParam(HINSTANCE hinst, LPCSTR lpTemplateName,
  HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
```

CreateDialogParam 是一個很短、很簡單的函式。如果你有微軟的原始碼，你會發現它看起來像這樣：（爲了清楚說明，我移去了一些錯誤檢查碼）

```
HWND CreateDialogParam(HINSTANCE hinst, LPCSTR lpTemplateName,
  HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit) {

  // Find the dialog box template resource in the EXE file.
  HRSRC hrsrc = FindResource(hinst, lpTemplateName, RT_DIALOG);
  HGLOBAL hglblRes = LoadResource(hinst, hrsrc);

  // Obtain the memory address of the dialog box template.
  PDLGTEMPLATEEX pDlgTemplate = (PDLGTEMPLATEEX) LockResource(hglblRes);

  // Create the parent window and child windows using the
  // template contained in the memory block.
  HWND hwndDlg = CreateDialogIndirectParam(hinst, pDlgTemplate,
    hwndOwner, pfnDlgProc, lParamInit);
```

---

[2] WIN32 已不再支援 CreateDialog。但你仍然可以使用它，因爲在 WinUser.H 中有如下定義：

```
#define CreateDialogA(hInstance, lpName, hWndParent, lpDialogFunc) \
   CreateDialogParamA(hInstance, lpName, hWndParent, lpDialogFunc, 0L)

#define CreateDialogW(hInstance, lpName, hWndParent, lpDialogFunc) \
   CreateDialogParamW(hInstance, lpName, hWndParent, lpDialogFunc, 0L)

#ifdef UNICODE
#define CreateDialog  CreateDialogW
#else
#define CreateDialog  CreateDialogA
#endif // !UNICODE
```

```
   // NOTE: Windows NT does not require that resources be unlocked or freed;
   // however, Windows 95 does have this requirement.  It is always safer to
   // clean up than to leak memory for your process.
   UnlockResource(pDlgTemplate);
   FreeResource(hglblRes);

   // Return the handle of the parent window.
   return(hwndDlg);
}
```

你很容易的就可以看到 CreateDialogIndirectParam 負責了些什麼工作。讓我們詳細討論它的動作。以下是函式原型：

```
HWND CreateDialogIndirectParam(HINSTANCE hinst, PCDLGTEMPLATEEX pDlgTemplate,
     HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
```

第二個參數 pDlgTemplate，必須指向對話盒面板的二進位碼。CreateDialogIndirectParam 會開始剖析這塊記憶體，動作如下：

1. 將任何 DS_* 型態轉換爲 WS_* 型態。例如 DS_MODALFRAME 轉換爲 WS_EX_DLGMODALFRAME，DS_CONTEXTHELP 轉換爲 WS_EX_CONTEXTHELP，WS_DLGFRAME 轉換爲 WS_EX_WINDOWEDGE[3]。
2. 如果 DS_FONT 被指定，CreateFontIndirect 會根據記憶體區塊中的 FONTINFOEX 結構內容，產生一種邏輯字型。
3. 對話盒的寬度和高度可經由 MapDialogRect 函式轉換成以圖素（pixel）爲單位的值。
4. 呼叫 CreateWindowEx 並使用 DLGTEMPLATEEX 結構內的欄位來產生父視窗。就這父視窗而言，其擴充風格、類別名稱、標題文字、視窗風格、x 座標、y 座標、寬度、高度和選單資訊都已記錄在 DLGTEMPLATEEX 結構之中，而

[3] 產生對話盒主視窗時，不論 DS_CONTROL 是否被指定，CreateDialogIndirectParam 總是都會內含視窗擴充風格 WS_EX_CONTROLPARENT 。

CreateWindowEx 的 hInstance 參數和 hwndParent 參數（是的，父視窗也可以有擁有者或是父視窗），將使用 CreateDialogIndirectParam 所獲得的 hinst 和 hwndOwner 參數。

當我們使用 Visual C++ 來設計對話盒時，大部份時候我們並不會自己指定視窗類別名稱。如果沒有指定視窗類別名稱，當 CreateDialogIndirectParam 呼叫 CreateWindowEx 時，會傳入內建的對話盒視窗類別名稱。對話盒類別是一個 “system global” 視窗類別，但卻不像其它 “system global” 視窗類別一樣有一個優雅的名稱（如 ListBox 或 Edit 等等）。這個類別沒有名稱，只有一個代碼：#32770，那也就是此一類別的 atom 值。由於這是一個 “system global” 視窗類別，微軟也爲它提供了一個視窗函式，放在 User32.DLL 之中，其視窗函式名爲 DefDlgProc，下一節我們將詳加討論之。

5. 如果面板的 help ID 不爲 0，SetWindowContextHelpId 函式會被呼叫，並傳入父視窗代碼及 help ID：

```
BOOL SetWindowContextHelpId(HWND hwnd, DWORD dwContextHelpId);
```

如果使用者在對話盒中要求 Help，視窗函式會收到一個 WM_HELP 訊息，其 lParam 參數指向一個 HELPINFO 結構，結構中的 dwContextId 欄位將擁有對話盒的 help ID。

6. 如果 DS_SETFONT 被指定，父視窗會收到一個 WM_SETFONT 訊息。

至此，父視窗已經產生出來並初始化了。CreateDialogIndirectParam 會再根據每一個 DLGITEMTEMPLATEEX 結構，產生一個子視窗：

7. 每一個子視窗的產生，都是靠 DLGITEMTEMPLATEEX 結構以及呼叫 CreateWindowEx 函式來完成的。每一個子視窗的擴充風格[4]、類別名稱、標題文字、視窗風格以及 ID 資訊都已記錄在記憶體中。x 座標、y 座標、寬度和高度首先會被轉換爲以圖素爲單位（原本是所謂的對話盒單位）。至於

[4] CreateDialogIndirectParam 函式當欲產生子視窗時，總是會包含 WS_EX_NOPARENTNOTIFY 延伸視窗型態。

CreateDialogIndirectParam 所獲得的 hinst 值以及才剛被產生的父視窗代碼，將分別做爲 CreateWindowEx 的 hInstance 和 hwndParent 參數。

8. 如果子視窗不能產生，CreateDialogIndirectParam 函式會摧毀父視窗和所有已經產生出來的子視窗，並傳回 NULL。然而，藉由 DS_NOFAILCREATE 的指定，你可以改變這種預設行爲。假如指定了 DS_NOFAILCREATE 給父視窗，則 CreateDialogIndirectParam 函式會繼續產生其餘子視窗並傳回其視窗代碼給父視窗，再傳回給呼叫者。
9. 如果子視窗有一個 help ID，而且不是 0，那麼 SetWindowContextHelpId 會被呼叫，並傳入子視窗代碼和 help ID。如果使用者在子視窗中要求 help 功能，對話盒視窗函式會收到一個 WM_HELP 訊息，其 lParam 參數指向一個 HELPINFO 結構，結構中的 dwContextId 欄位表現出子視窗的 help ID。
10. WM_SETFONT 訊息會被送往子視窗，告訴子視窗應使用何種字型來顯示文字。

現在，在父視窗及其所有子視窗都已產生之後，CreateDialogIndirectParam 送出一個 WM_INITDIALOG 訊息給父視窗。而此訊息的 lParam 參數值就是 CreateDialogIndirectParam 的 lParamInit 參數值。

我並未在這裡討論 CreateDialogIndirectParam 函式的 pfnDlgProc 參數，那是下一節的主題。

對話盒類別的視窗函式

就像先前所提的，每一個行程初始化時，Windows 會爲它註冊「對話盒」這麼一個 "system global" 視窗類別，此類別的視窗函式被微軟實作於 User32.DLL 中，函式名稱叫做 DefDlgProc：

```
LRESULT DefDlgProc(HWND hwndDlg, UNIT uMsg, WPARAM wParam, LPARAM lParam);
```

這個函式符合標準視窗函式的規格，因爲它真的就是個視窗函式。此視窗函式的存在是

爲了讓對話盒的處理更容易些，就像其它對話盒函式一樣。

當我們設計一個對話盒，我們必須先產生一個對話盒函式，型式如下：

```
BOOL DlgProc (HWND hwnd, UNIT uMsg, WPARAM wParam, LPARAM lParam);
```

這個函式看起來很像視窗函式，但其傳回值是 BOOL 而不是 LRESULT。這個區別非常重要。微軟在設計對話盒機制時，他們想要幫助程式開發人員簡化一些事情的處理動作。當父視窗收到視窗訊息，會先進入 DefDlgProc（對話盒類別的視窗函式），然後 DefDlgProc 呼叫我們自己的對話盒函式。

爲了讓事情簡單些，應用程式中的所有對話盒函式都必須傳回 TRUE 或 FALSE，表示我們是否已經處理過這個訊息。如果我們處理了這個訊息（傳回 TRUE），DefDlgProc 將不會再做其它處理，而是直接回返至系統。如果我們的對話盒函式並沒有處理這個訊息（傳回 FALSE），DefDlgProc 會執行此訊息的預設處理動作，而大部份情況是呼叫 DefWindowProc 來處理。

理論上這似乎是個很好的方法。然而，許多視窗訊息都期望在被處理過後有一個有意義的傳回值。例如 WM_CTLCOLOREDIT 訊息希望獲得一個畫刷代碼。如果交由我們自己的對話盒函式來處理 WM_CTLCOLOREDIT，我們該如何表示這畫刷代碼呢？又，一個對話盒函式應該如何回應 WM_QUERYENDSESSION 訊息呢？

對於某些這樣的訊息，微軟修改內建的 DefDlgProc 函式，明白檢查某些訊息，並且不管應用程式的對話盒函式針對此訊息傳回什麼樣的值，DefDlgProc 函式都回返至系統。然而當 Windows API 持續發展下去，微軟知道，應用程式的對話盒函式需要一個有效的方法來表示它處理訊息後的結果。所以，微軟爲每一個產生出來的對話盒增加了 4 個額外位元組（extra bytes）。你可以使用 GetWindowLong 和 SetWindowLong 函式，傳入 DWL_MSGRESULT 識別字，來取得或設定這 4 個位元組的值。所以，應用程式的對話盒函式可以藉由以下動作，指定其執行結果：

```
SetWindowLong(hwndDlg, DWL_MSGRESULT, lResult);
```

並且能夠傳回 TRUE 或 FALSE 來表示訊息是否已被處理。爲了處理上更方便些，WindowsX.H 標頭檔中定義了一個巨集如下：

```
#define SetDlgMsgResult(hwnd, msg, result) (( \
        (msg) == WM_CTLCOLORMSGBOX       || \
        (msg) == WM_CTLCOLOREDIT         || \
        (msg) == WM_CTLCOLORLISTBOX      || \
        (msg) == WM_CTLCOLORBTN          || \
        (msg) == WM_CTLCOLORDLG          || \
        (msg) == WM_CTLCOLORSCROLLBAR    || \
        (msg) == WM_CTLCOLORSTATIC       || \
        (msg) == WM_COMPAREITEM          || \
        (msg) == WM_VKEYTOITEM           || \
        (msg) == WM_CHARTOITEM           || \
        (msg) == WM_QUERYDRAGICON        || \
        (msg) == WM_INITDIALOG              \
    ) ? (BOOL)(result)
      :(SetWindowLong((hwnd),DWL_MSGRESULT,(LPARAM)(LRESULT)(result)),TRUE))
```

這個巨集一次執行了兩個動作。如果訊息是巨集中所列的 12 種訊息之一的話，result 就用來表示訊息已被處理，它同時也就是應用程式的對話盒函式的回返值。但對其它訊息來說，result 值會被存到視窗的額外位元組（extra bytes）中，且傳回一個 TRUE 值，表示此訊息已被處理過。

爲了更清楚解釋這裡所發生的事情，以下是 DefDlgProc 函式虛擬碼（pseudo-code）的大略架構：

```
LRESULT DefDlgProc (HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{

   BOOL fProcessedByApp = FALSE;

   // Assume that the application's dialog box
   // procedure doesn't process the message.
   SetWindowLong(hwndDlg, DWL_MSGRESULT, 0);

   // Get the address of the application's dialog box procedure.
   DLGPROC pfnDlgProc = (DLGPROC) GetWindowLong(hwndDlg, DWL_DLGPROC);
   if (pfnDlgProc != NULL) {
```

```
   // Call the application's dialog box procedure.
   fProcessedByApp = pfnDlgProc(hwnd, uMsg, wParam, lParam);
}

switch (uMsg) {
   case WM_CTLCOLORMSGBOX:
   case WM_CTLCOLOREDIT:
   case WM_CTLCOLORLISTBOX:
   case WM_CTLCOLORBTN:
   case WM_CTLCOLORDLG:
   case WM_CTLCOLORSCROLLBAR:
   case WM_CTLCOLORSTATIC:
   case WM_COMPAREITEM:
   case WM_VKEYTOITEM:
   case WM_CHARTOITEM:
   case WM_INITDIALOG:
   case WM_QUERYDRAGICON:

    // If uMsg is any of the preceding messages, return whatever
    // the application's dialog box procedure returned.
    // NOTE: Even though fProcessedByApp is declared as a BOOL, it can
    //      contain a 32-bit value. This is necessary when an
    //      HBRUSH is returned from a WM_CTLCOLOR* message.
    return((LRESULT) fProcessedByApp);
}

if (fProcessedByApp) {

   // If the application's dialog box procedure processed the message,
   // return whatever the application's dialog box procedure stored
   // in the dialog window's DWL_MSGRESULT extra byte value.
   return((LRESULT) GetWindowLong(hwndDlg, DWL_MSGRESULT));
}

// If we get here, the application's dialog box procedure did not process
// the message so we should do it using a regular switch statement.
switch (uMsg) {

   case DM_GETDEFID:
      // Process a DM_GETDEFID message.
      break;

   case DM_SETDEFID:
      // Process a DM_SETDEFID message.
      break;
```

```
      case DM_REPOSITION:
         // Process a DM_REPOSITION message.
         break;

      case WM_*:
         // Process various other messages.
         break;

      default:
         lResult = DefWindowProc(hwndDlg, uMsg, wParam, lParam);
         break;
   }
   return(lResult);
}
```

我唯一要說明的是額外位元組 DWL_DLGPROC 是如何初始化爲你的「應用程式對話盒函式」的位址。答案很簡單，當你呼叫 CreateDialogParamIndirect，你的對話盒函式位址便指定給 pfnDlgProc 參數了，而 CreateDialogParamIndirect 在產生父視窗之後，便執行了以下動作：

```
SetWindowLong(hwndParent, DWL_DLGPROC, pfnDlgProc);
```

然後，DefDlgProc 便會將所有送往父視窗視窗函式的訊息，統統交給我們的對話盒函式來處理。這也意味我們的對話盒函式不會收到 WM_CREATE 和 WM_NCCREATE。

由於這種架構，許多程式員常常會犯一個錯誤，以下片段取自應用程式的對話盒函式：

```
case WM_SOMEMSG:
   SetDlgMsgResult(hwnd, uMsg, 5);
   SetWindowText(hwnd, "New window text");
   return(TRUE);
```

這段碼會設定 DWL_MSGRESULT 視窗位元組爲 5，表示訊息處理結果爲 5。但之後卻呼叫了 SetWindowText，於是強制送出一個 WM_SETTEXT 訊息給 DefDlgProc。當 DefDlgProc 函式被遞迴呼叫，它執行以下動作：

```
// Assume that the application's dialog box
// procedure doesn't process the message,
SetWindowLong(hwndDlg, DWL_MSGRESULT, 0);
```

於是把 5 改爲 0。WM_SETTEXT 被處理過後，SetWindowText 會傳回 TRUE 給 DefDlgProc，於是 DefDlgProc 看到結果爲 0，而不是我們期望的 5。解決這個問題的方法是調換執行順序：

```
case WM_SOMEMSG:
   SetWindowText(hwnd, "New window text");
   SetDlgMsgResult(hwnd, uMsg, 5);
   return(TRUE);
```

Modeless 對話盒的鍵盤控制

由 CreateDialogParamIndirect 所產生的對話盒，是所謂的 Modeless 對話盒。也就是說，它會產生一個父視窗和一堆子視窗，並且在這些視窗都被產生出來之後，CreateDialogIndirectParam 回返，程式從回返處繼續執行。

對話盒的好處之一是，你可以利用一些助憶碼，以鍵盤控制對話盒。當對話盒呈現在使用者眼前時，他可以直接使用鍵盤（如 Tab 或 Shift+Tab 鍵）來控制輸入焦點在不同控制元件之間移轉。他也可以按下【Enter】鍵來選取所謂的 “default pushbutton”，或是按下【Esc】鍵來跳離對話盒。此外，他也可以使用方向鍵和助憶碼來改變鍵盤焦點。早在 1980 年，Windows 設計之初，對鍵盤並沒有太好的支援。鍵盤的支援是事後添加上去的，微軟在這上面花了許多功夫。對話盒的鍵盤控制係藉著 IsDialogMessage 函式來完成：

```
BOOL IsDialogMessage(HWND hwndDlg, PMSG lpmsg);
```

這個函式通常在你的執行緒的訊息迴路中被呼叫：

```
int WINAPI WinMain(...) {

   Msg msg;

   // Create the dialog box parent window and all its child windows.
   HWND hwndDlg = CreateDialogParam(...);
   .
   .
   .
   while (GetMessage(&msg, NULL, 0, 0)) {
```

```
      // After a message is pulled from the queue, IsDialogMessage checks to see
      // if it is a keyboard  message  destined for the dialog box or any of its
      // children. If it  is  navigational  key,  IsDialogMessage  performs  the
      // navigation and return TRUE.  For anything else,  IsDialogMessage
      // returns FALSE, and the message is processed normally.
      if (!IsDialogMessage(hwndDlg, &msg)) {

         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
   .
   .
   .
   return(0);
}
```

如果你沒有修改你的執行緒的訊息迴路，使它呼叫 IsDialogMessage，你的對話盒將不能夠支援鍵盤控制。當然，使用者可以使用滑鼠。

怎樣製作一個 Modal 對話盒

最常使用的對話盒型式莫過於 modal 對話盒。所謂 modal 對話盒是，使用者必須先結束此對話盒，才能繼續和程式的其它視窗溝通。【開啓舊檔】對話盒是 modal 對話盒的最佳範例，當【開啓舊檔】對話盒尚未出現時，使用者可以使用鍵盤或滑鼠在程式視窗中自由工作，一旦【開啓舊檔】對話盒出現，使用者便無法在此程式的其它子視窗工作，也不能夠切換視窗。他必須要離開此一對話盒，才可以繼續操作目前的程式。

我們可以利用 DialogBoxParam[5] 來產生一個 modal 對話盒：

[5] WIN32 API 已不再支援 CreateDialog 和 DialogBox 兩函式了。但你仍可繼續使用它，因爲 WinUser.H 表頭檔中有一些相關的巨集定義。

```
int DialogBoxParam(HINSTANCE hinst, LPCSTR lpTemplateName, HWND hWndOwner,
   DLGPROC pfnDlgProc, LPARAM lParamInit);
```

如果你有 DialogBoxParam 的原始碼，你會發現它是怎麼運作的（再一次，我省略了一些錯誤檢查碼）：

```
int DialogBoxParam(HINSTANCE hinst, LPCSTR lpTemplateName,
   HWND hWndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit) {

   if (IsWindow(hwndOwner)) {

      // The Owner window should be disabled so that it no longer
      // receives keystrokes or mouse input.

      HWND hwndActive = hwndOwner;
      while (GetWindowStyle(hwndActive) & WS_CHILD) {

         // If the caller passed the handle of a child window.
         // walk up until we find  popup or overlapped window.
         hwndActive = GetParent(hwndActive);
      }

      EnableWindow(hwndActive, FALSE);
   }

   // Create the Modeless dialog box (parent window and child window).
   HWND hwndDlg = CreateDialogParam(hinst, lpTemplateName,
      hwndOwner, pfnDlgProc, lParamInit);

   // The undocumented DWL_ENDDIALOGCALLED extra byte value is
   // initialized to FALSE by the system when the window is created.
   // See the following pseudo-code for the EndDialog function for more details.
   SetWindowWord(hwndDlg, DWL_ENDDIALOGCALLED, FALSE);

   // Stay in a message loop until EndDialog is called (the undocumented
   // DWL_ENDDIALOGCALLED extra bytes value is set by EndDialog).
   // This message loop makes this dialog box modal.
   while(!GetWindowWord(hwndDlg, DWL_ENDDIALOGCALLED)) {
      GetMessage(&msg, NULL, 0, 0);

      // If EndDialog has not been called for this window,
      // continue this message loop.
      if (!IsDialogMessage(hwndDlg, &msg)) {
         TranslateMessage(&msg);
```

```
        DispatchMessage(&msg);
      }
   }

   // Loop ended because EndDialog was called.

   // Get the value passed to EndDialog (the undocumented
   // DWL_DLGRESULT extra bytes value is set by EndDialog).
   int nResult = GetWindowLong(hwndDlg, DWL_DLGRESULT);

   // Destroy the parent window and all its child windows.
   DestroyWindow(hwndDlg);

   // Return the value passed to EndDialog back to
   // the thread that invoked the modal dialog box.
   return(nResult);
}
```

由這些程式碼，你可以清楚看出 modal 對話盒和 modaless 對話盒並沒有太大的差別。事實上只有兩件事不同。第一，DialogBoxParam 會抑制（disable）其擁有者（owner，也是一個視窗）的活動。對【開啓舊檔】對話盒而言，其擁有者 -- 通常就是程式的主視窗 -- 將不再收到任何硬體如鍵盤或滑鼠的訊息；不過倒是可以收到其它訊息。第二個不同是 DialogBoxParam 內有其自己的訊息迴路。由於有訊息迴路中有 GetMessage，所以 DialogBoxParam 不會立刻回返。你一定也注意到了，訊息迴路中呼叫了 IsDialogMessage，這可以確保鍵盤在 modal 對話盒中正常運作。

爲了終止訊息迴路，對話盒函式中必須呼叫 EndDialog：

```
BOOL EndDialog(HWND hwnd, int nResult);
```

這個函式需要一個對話盒視窗代碼，以及一個 nResult，此值將被 DialogBoxParam 當做傳回值。EndDialog 會重新將父視窗致能（enable），並設定對話盒額外位元組（extra bytes）中的一些未公開數值。DialogBoxParam 的訊息迴路一看到這些額外位元組被改變，就終止訊息迴路，將 EndDialog 的 nResult 參數傳回給 DialogBoxParam 的呼叫者。

以下是 EndDialogBox 的虛擬碼（pseudo-code）：

```
BOOL EndDialog(HWND hwnd, int nResult) {

   HWND hwndOwner = GetParent(hwnd);

   if (IsWindow(hwndOwner))
      EnableWindow(hwndOwner, TRUE);

   // Set the undocumented DWL_ENDDIALOGCALLED and DWL_DLGRESULT extra byte values
   // so that the modal dialog box ends and the desired return value is stored
   // away so that it can be retrieved inside of the DialogBoxParam function.
   SetWindowLong(hwnd, DWL_ENDDIALOGCALLED, TRUE);
   SetWindowLong(hwnd, DWL_DLGRESULT, nResult);

   // Force the message loop inside DialogBoxParam to wake
   // up and see that the dialog box is terminating.
   PostMessage(hwnd, WM_NULL, 0, 0);

   return(TRUE);  // EndDialog was successful
}
```

動態的 Modal 對話盒

DialogBoxIndirectParam 也能用來產生一個 modal 對話盒：

```
int DialogBoxIndirectParam(HINSTANCE hinst, PCDLGTEMPLATE hTemplate,
   HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
```

前一節所說的 DialogBoxParam 函式假定對話盒面板在 EXE 檔的資源之中，而這裡的 DialogBoxIndirectParam 是一個更低階函式，需要一個指標指向對話盒面板的記憶體位址。這表示對話盒面板不需要一定得在資源之中，可以被動態地產生於記憶體中。此函式將在第三章的 DlgDyn.03 範例中介紹。

以對話盒作為主視窗

Clock 範例程式（Clock.EXE）的程式碼在**程式列表 2.1 ~ 2.3** 中，示範如何將一個對話盒當作程式主視窗。這項技術我最喜歡，因爲它能省下我不少的時間。運作這項技術你就不再需要註冊視窗類別，不再需要呼叫 CreateWindowEx，更不需要自己建立一個訊息迴路。本書幾乎所有範例程式都使用這項技術。

本書的大部份程式執行時都會產生一個視窗。這個視窗是程式與使用者之間的一個溝通介面。對我而言，最容易的作法當然是以 Visual C++ 中的對話盒編輯器來設計。舉個例，我想要寫一個小程式用來顯示時間。當這個程式執行起來，畫面如圖 2.3。

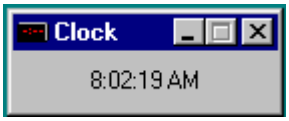


圖 2.3 Clock 程式

此程式所需用到的檔案，列在表 2.2 中。

表 2.2 建立 Clock 程式所需的檔案

檔案	說明
Clock.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Clock.RC	內含主視窗的對話盒面板及圖示。
Resource.H	內含 Clock.RC 檔中所有資源的 ID。
Clock.ICO	主視窗圖示。
Clock.MAK	Visual C++ 的 MAK 檔。

爲了建立此程式，首先我在 Visual C++ 中產生一個新專案，然後進入對話盒編輯器中，設計圖 2.3 的對話盒（如圖 2.4 所示）。

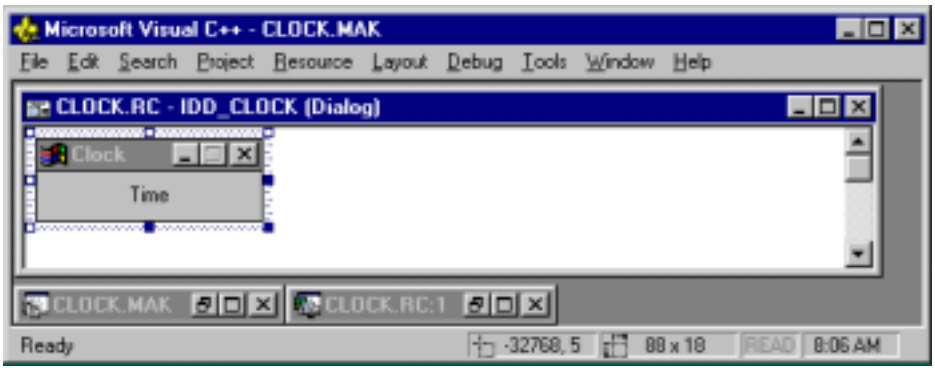


圖 2.4　Clock 程式的對話盒

然後，我必須寫一些碼使程式有效運作。我一共需要五個函式。首先我必須寫一個對話盒函式如下：

```
BOOL WINAPI Clock_DlgProc (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{

   switch (uMsg) {

      // Standard Window's messages
      adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, Clock_OnInitDialog);
      adgHANDLE_DLGMSG(hwnd, WM_TIMER,      Clock_OnTimer);
      adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    Clock_OnCommand);
   }
   return(FALSE);              // We didn't process the message.
}
```

這是一個標準的對話盒函式。請注意這裡我使用我自己寫的 adgHANDLE_DLGMSG 巨集來代替 WindowsX.H 檔中的 HANDLE_MSG 訊息剖析器函式。這個巨集被定義在 Win95ADG.H 表頭檔中，我將在本書附錄 A 討論之。

```
// The normal HANDLE_MSG macro in WINDOWSX.H does not work properly for dialog
// boxes because DlgProc's return a BOOL instead of an LRESULT (like
// WndProcs). This adgHANDLE_DLGMSG macro corrects the problem:
#define adgHANDLE_DLGMSG(hwnd, message, fn)                        \
   case (message): return (SetDlgMsgResult(hwnd, uMsg,             \
      HANDLE_##message((hwnd), (wParam), (lParam), (fn))))
```

回想一下，對話盒函式的傳回值和視窗函式的傳回值是不一樣的。由於這個差異，所以你不應該在對話盒函式中使用 HANDLE_MSG 巨集。我的 adgHANDLE_DLGMSG 巨集使用了 SetDlgMsgResult 巨集（也定義在 WindowsX.H 中），爲的是使傳回值能配合對話盒函式。注意，如果對話盒函式最後傳回的是 FALSE，將不會去呼叫 DefWindowProc。這是因爲此函式並非視窗函式，其傳回值只不過表示訊息是否已被處理過而已。

接下來我們必須建立三個函式，用來處理視窗訊息：Clock_OnInitDialog 產生一個週期 1 秒的計時器，Clock_OnTimer 每秒更新一次 IDC_TIME 控制元件（那是一個 static 控制元件），Clock_OnCommand 檢查 IDCANCEL。如果 Clock_OnCommand 沒有檢查 IDCANCEL，對話盒將永遠不能關閉。

最後一個函式必須寫在 WinMain 函式中。由於我們以對話盒作爲主視窗，所以程式的 WinMain 相當簡單，以下便是其全部程式碼：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   // Create the application window.
   DialogBox(hinstExe, MAKEINTRESOURCE(IDD_CLOCK), NULL, Clock_DlgProc));

   return(0);
}
```

我簡直不敢相信一個這麼簡單的函式卻能做那麼多的工作。唯一一個呼叫動作 DialogBox 產生了主視窗和所有的子視窗。然後，執行緒進入它自己的訊息迴路（包含在 DialogBoxParam 函式內）。一旦對話盒關閉，訊息迴路便終止，於是 DialogBoxParam 回返，我們的程式隨之終止。我們不需要註冊任何視窗類別，也不需要建立任何訊息迴路。

Clock.ico

程式列表 2.1　Clock.C

```
#0001  /******************************************************************
#0002  Module name: Clock.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates using a dialog box for an application's main window.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include "resource.h"
#0014
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017
#0018
#0019  BOOL Clock_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0020
#0021     SetTimer(hwnd, 1, 1000, NULL);             // 1 second intervals
#0022
#0023     FORWARD_WM_TIMER(hwnd, 1, SendMessage);     // Force an initial update
#0024
#0025     adgSETDLGICONS(hwnd, IDI_CLOCK, IDI_CLOCK);
#0026     return(TRUE);                              // Accept default focus window.
#0027  }
#0028
#0029
#0030  ////////////////////////////////////////////////////////////////
#0031
#0032
#0033  void Clock_OnTimer (HWND hwnd, UINT id) {
#0034
#0035     // Show current time
#0036     TCHAR szTime[100];
#0037     GetTimeFormat(LOCALE_USER_DEFAULT, 0, NULL, NULL,
#0038        szTime, adgARRAY_SIZE(szTime));
#0039     SetDlgItemText(hwnd, IDC_TIME, szTime);
#0040  }
#0041
#0042
#0043  ////////////////////////////////////////////////////////////////
#0044
```

```
#0045
#0046  void Clock_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0047
#0048     switch (id) {
#0049        case IDCANCEL:            // Allows dialog box to close
#0050           EndDialog(hwnd, id);
#0051           break;
#0052     }
#0053  }
#0054
#0055
#0056  ////////////////////////////////////////////////////////////////
#0057
#0058
#0059  BOOL WINAPI Clock_DlgProc (HWND hwnd, UINT uMsg,
#0060     WPARAM wParam, LPARAM lParam) {
#0061
#0062     switch (uMsg) {
#0063
#0064        // Standard Window's messages
#0065        adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, Clock_OnInitDialog);
#0066        adgHANDLE_DLGMSG(hwnd, WM_TIMER,     Clock_OnTimer);
#0067        adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    Clock_OnCommand);
#0068     }
#0069     return(FALSE);               // We didn't process the message.
#0070  }
#0071
#0072
#0073  ////////////////////////////////////////////////////////////////
#0074
#0075
#0076  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0077     LPSTR lpszCmdLine, int nCmdShow) {
#0078
#0079     adgWARNIFUNICODEUNDERWIN95();
#0080     adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_CLOCK),
#0081        NULL, Clock_DlgProc));
#0082
#0083     return(0);
#0084  }
#0085
#0086
#0087  ////////////////////////////// End of File //////////////////////
```

程式列表 2.2　Clock.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  ////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  ////////////////////////////////////////////////////////////////
#0044  //
```

```
#0045  // Dialog
#0046  //
#0047
#0048  IDD_CLOCK DIALOG DISCARDABLE  0x8000, 5, 88, 18
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | 0x800
#0050  CAPTION "Clock"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      CTEXT          "Time",IDC_TIME,0,4,88,12,SS_NOPREFIX
#0054  END
#0055
#0056
#0057  /////////////////////////////////////////////////////////////////
#0058  //
#0059  // Icon
#0060  //
#0061
#0062  IDI_CLOCK             ICON    DISCARDABLE     "Clock.ico"
#0063
#0064  #ifndef APSTUDIO_INVOKED
#0065  /////////////////////////////////////////////////////////////////
#0066  //
#0067  // Generated from the TEXTINCLUDE 3 resource.
#0068  //
#0069
#0070
#0071  /////////////////////////////////////////////////////////////////
#0072  #endif    // not APSTUDIO_INVOKED
```

程式列表 2.3　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by Clock.rc
#0004  //
#0005  #define IDC_TIME                    100
#0006  #define IDD_CLOCK                   101
#0007  #define IDI_ICON1                   102
#0008  #define IDI_CLOCK                   102
#0009
#0010  // Next default values for new objects
#0011  //
#0012  #ifdef APSTUDIO_INVOKED
#0013  #ifndef APSTUDIO_READONLY_SYMBOLS
#0014  #define _APS_NEXT_RESOURCE_VALUE     103
```

```
#0015  #define _APS_NEXT_COMMAND_VALUE        40001
#0016  #define _APS_NEXT_CONTROL_VALUE        1000
#0017  #define _APS_NEXT_SYMED_VALUE          101
#0018  #endif
#0019  #endif
```

對話盒 Tab 鍵的走訪順序

我即將在這一節中介紹的這項技術，帶出來的一個主要觀念就是：對話盒與一般視窗並沒有什麼不同。我將探討對話盒的 Tab 次序，以及「如何在控制元件已經被產生並顯現之後，再增加或移除控制元件」。

讓我們從 Tab 的工作順序開始。每當產生一個視窗，系統便會將此視窗加入系統內部的一個樹狀結構中，這個樹狀結構稱爲 window manager's list。當 CreateDialogBoxIndirectParam 產生子視窗，它會一個接著一個地將子視窗安插到此樹狀結構中。只要是影響到子視窗的動作，Windows 都會去參考 window manager's list。例如，如果父視窗被隱藏，其所有子視窗也會隱藏；如果父視窗被摧毀，其所有子視窗也會被摧毀。

如果我製作了如圖 **2.5** 的對話盒，資源編譯器會產生對話盒面板如下：

```
IDD_TABSTOPS DIALOG DISCARDABLE  0, 0, 248, 106
STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Tabstops"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Checkbox &1",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,8,8,56,10
    CONTROL         "Checkbox &2",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,8,20,56,10
    CONTROL         "Checkbox &3",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,8,32,56,10
    PUSHBUTTON      "&Add dialog control",IDC_ADDDLGCNTL,8,60,64,14
    LTEXT       "Placeholder for control parent dialog box\n\nThis window is
destroyed when the ""Add dialog control"" button is pressed.",
                    IDC_CTRLPRNT,80,8,104,92
    DEFPUSHBUTTON   "OK",IDOK,192,4,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,192,21,50,14
END
```

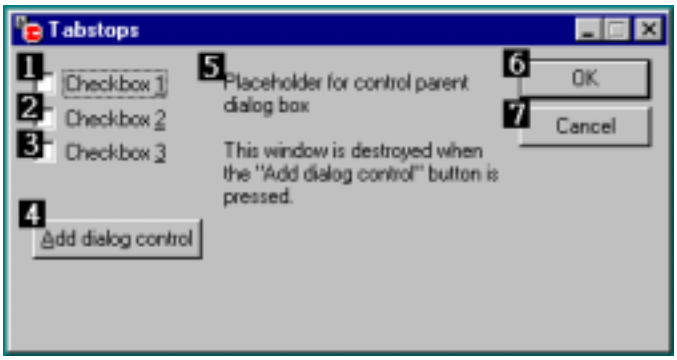


圖 2.5　一個對話盒面板

對話盒中的各個控制元件的敘述句順序極爲重要，CreateDialogIndirect 會依序呼叫 CreateWindowEx 來產生對應的子視窗。這表示，本例首先會產生三個 checkboxes，緊跟著是【Add dialog control】鈕，然後是 static 控制元件、【OK】鈕和【Cancel】。依此順序，按下 Tab 鍵將依序走訪每一個 checkbox，然後到【Add dialog control】鈕。再按一下 Tab 鍵，鍵盤焦點會轉移到【OK】鈕 -- 因爲 static 元件不會得到鍵盤焦點。如果焦點位於【Cancel】鈕，再按一次 Tab 鍵，焦點會移至第一個 checkbox。

當一個控制元件得到了鍵盤焦點，使用者如果按下 Tab 鍵，IsDialogMessage 函式會呼叫 GetNextDlgTabItem 以決定哪一個視窗將獲得輸入焦點：

```
HWND GetNextDlgTabItem(HWND hwndDlg, HWND hwndCtl, BOOL fPrevious);
```

這個函式搜尋 window manager's list，尋找對話盒的子視窗。更明確地說，它是從目前擁有鍵盤焦點的視窗開始，循著 list 向前搜尋（跳過任何隱藏的、被除能的、或 static 控制元件），搜尋那些擁有 WS_TABSTOP 型態子的視窗。如果你改變 window manager's list 內的視窗順序，你同時也就改變了 Tab 的走訪順序。

當 IsDialogMessage 發現使用者按下了 Tab 鍵，便會去呼叫 GetNextDlgTabItem 函式：

```
SetFocus(GetNextDlgTabItem(hwndDlg, GetFocus(), FALSE);
```

通常，GetNextDlgTabItem 只會檢查目前對話盒的子視窗。可是，如果一個或多個子視窗設定了新的視窗擴充風格 WS_EX_CONTROLPARENT，上述行爲會被改變。當 GetNextDlgTabItem 檢查每個子視窗，看看它們是否設定 WS_TABSTOP 風格時，它也會檢查它們是否設立 WS_EX_CONTROLPARENT 風格。如果 WS_EX_CONTROLPARENT 爲 on，則 GetNextDlgTabItem 會繼續往下搜尋，看看其子視窗是否也有設立 WS_EX_CONTROLPARENT。這些子視窗算起來是對話盒的孫子輩了。如果有任何一個「孫」視窗設立了 WS_TAPSTOP，GetNextDlgTabItem 會將鍵盤焦點移轉給它，並傳回其視窗代碼。

如果沒有任何一個「孫」視窗設立 WS_TABSTOP，GetNextDlgTabItem 會跳回上一層，繼續搜尋對話盒目前的子視窗。而如果有一個「孫」視窗設立了 WS_EX_CONTROLPARENT，GetNextDlgTabItem 會繼續下到另一層到搜尋，看看是否有「曾孫」視窗設立 WS_TABSTOP。

Tabstop 範例程式

Tabstop 程式（Tabstops.EXE）的原始碼顯示於程式列表 **2.4 ~ 2.6** 中，示範如何控制對話盒中的 Tab 走訪順序，以及如何改變這個順序。它同時也示範如何在對話盒中增加和移除一個已產生並顯示的控制元件。當你執行這個程式，會顯示圖 **2.6** 的對話盒。

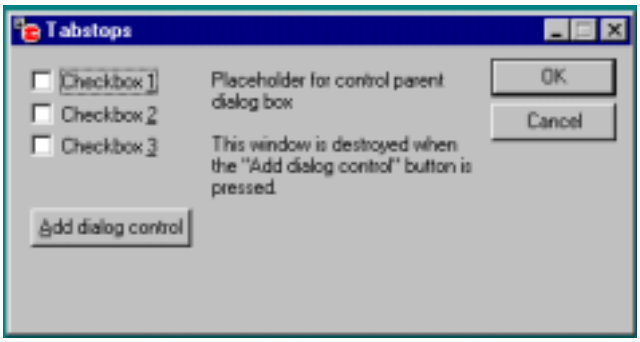


圖 2.6　Tabstop 的對話盒

在這個對話盒中，你可以輕易使用 Tab 鍵來移動控制元件的焦點。請驗證 Tab 順序是否與對話盒面板中的 CONTROL 敘述句順序相同。你也可以使用助憶鍵來改變焦點，例如按下 Alt+A 會將焦點移至【Add dialog control】鈕，並壓下它。

注意對話盒中的文字 "Placeholder for control…"，那是一個 static 控制元件。static 控制元件不會獲得輸入焦點，因此當焦點落在【Add dialog control】鈕，再按下 Tab 鍵，焦點會跳過 static 控制元件，移至【OK】鈕。

此程式所需使用到的檔案，列在表 **2.3** 中。

表 2.3 建立 TapStops 程式所需的檔案

檔案	說明
TapStops.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
TapStops.RC	內含主視窗的對話盒面板，及其圖示。
Resource.H	內含 TapStops.RC 檔中所有資源的 ID。
TapStops.ICO	主視窗圖示。
TapStops.MAK	Visual C++ 的 MAK 檔。

欲改變 Tab 順序，請選按圖 **2.6** 的【Add dialog control】鈕。如果你這麼做，TabStops_OnCommand 會收到 WM_COMMAND 訊息並以一個迷你對話盒取代原本的 placeholder static 控制元件。這個迷你對話盒顯示在圖 **2.7** 中。

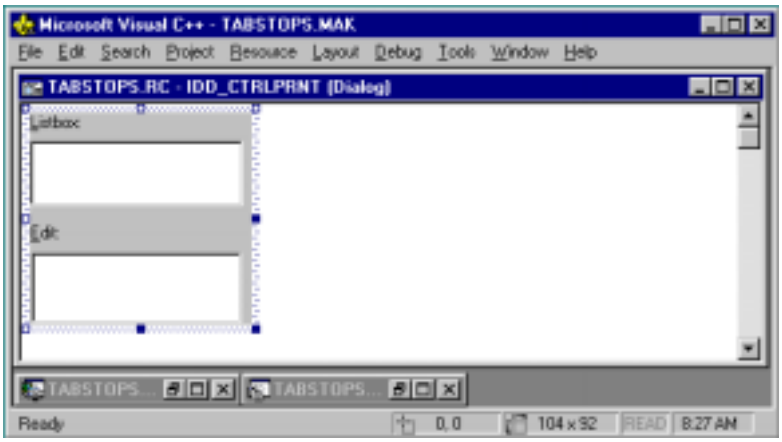


圖 2.7　當使用者選擇了【Add dialog control】鈕，placeholder static 控制元件會以此迷你視窗代替之。

此視窗的文字描述如下：

```
IDD_CTRLPRNT DIALOG DISCARDABLE  0, 0, 104, 92
STYLE WS_CHILD | WS_VISIBLE
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT          "&Listbox:",IDC_STATIC,0,0,29,8
    LISTBOX        IDC_LIST1,0,12,100,28,LBS_SORT | LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT          "&Edit:",IDC_STATIC,0,48,16,8
    EDITTEXT       IDC_EDIT1,0,60,100,32,ES_MULTILINE | ES_AUTOHSCROLL
END
```

注意，這個對話盒並沒有指定邊框型態（WS_BORDER 或 WS_DLGFRAME 等等），不過它指定了 WS_CHILD。在以此取代 placeholder static 控制元件之後，程式出現了新的視窗。

我們所做的，只不過是以一個視窗取代另一個視窗。但這個新視窗是一個擁有四個子視窗的父視窗，要做到正確的替換動作，我們必須改變 window manager's list 中的視窗位置，才能使 Tab 鍵正確工作。現在我要來探討視窗替換工作是如何完成的。

當使用者按下【Add dialog control】鈕 ，TabStops_OnCommand 會收到這個訊息並加以處理。首先，它找出 placeholder static 控制元件在 window manager's list 的位置，並取其視窗大小：

```
// Determine the location and size of the existing static child window.
hwndCtlOld = GetDlgItem(hwnd, IDC_CTRLPRNT);
GetWindowRect(hwndCtlOld, &rc);
MapWindowRect(HWND_DESKTOP, hwnd, &rc);   // Macro in windowsx.h
```

然後，產生迷你對話盒（對話盒面板在 EXE 檔的資源之中）。請注意，主對話盒的視窗代碼被傳給 CreateDialogParam 做爲參數，爲的是讓主對話盒成爲這個新對話盒的父視窗：

```
// Create the child dialog box window from the resource's template.
hwndCtlNew = CreateDialog(GetWindowInstance(hwnd),
   MAKEINTRESOURCE(IDD_CTRLPRNT), hwnd, NULL);
```

現在這個新對話盒需要重新定位並調整大小，以使它能夠正好放入原來 placeholder static 控制元件的位置。我們利用 SetWindowPos 來完成這項任務：

```
// Position and size the dialog box child window so that it is in the
// exact same location as the existing static child window.
// NOTE: The dialog box child is placed immediately after the existing
//       static window in the z-order so that the controls within it are properly
//       located in the tabbing order.
SetWindowPos(hwndCtlNew, hwndCtlOld, rc.left, rc.top,
   rc.right - rc.left, rc.bottom - rc.top, SWP_SHOWWINDOW);
```

此函式除了指定 x 和 y 座標外，也允許我們指定其 z-order。SetWindowPos 的第二個參數表示一個視窗代碼，而這個視窗代碼指的是你的替換對象的視窗代碼，本例爲 placeholder static 控制元件的視窗代碼。這表示新產生的迷你對話盒將插入至 window manager's list 中，緊跟在 placeholder static 控制元件之後。

只將這個新對話盒放置到 window manager's list 中，並不足以確保 Tab 鍵的工作順序是正

確的。爲了確保迷你對話盒中的控制元件也融入 tab 鍵的正確順序之中，我們必須爲新對話盒加入 WS_EX_CONTROLPARENT 風格，這可藉由以下函式來完成：

```
// Make sure that the window has the WS_EX_CONTROLPARENT style so
// that IsDialogMessage considers the dialog box's children
// as the user navigates with the keyboard.
ModifyWndStyle(hwndCtlNew, 0, WS_EX_CONTROLPARENT, TRUE);
```

現在我們已經產生出新對話盒並且將它定位好了，所以不再需要 placeholder static 控制元件。我們所以摧毀它：

```
// Destroy the existing static child window that was only used
// for placement anyway.
DestroyWindow(hwndCtlOld);
```

最後，由於我們加入了新對話盒，所以我們應該將【Add dialog control】鈕除能（disable），並將【OK】鈕設爲預設按鈕：

```
// Disable the "Add dialog control" button since we have just
// done this and we do not allow it to be done a second time.
EnableWindow(hwndCtl, FALSE);

// We also must tell the "Add dialog control" button to redraw
// itself in the not default pushbutton state and to set the
// OK button as the new default pushbutton.
Button_SetState(hwndCtl,
   Button_GetState(hwndCtl) & ~BS_DEFPUSHBUTTON);
SendMessage(hwnd, DM_SETDEFID, IDOK, 0);
```

將按鈕除能（disable）會產生一個問題。如果一個擁有鍵盤焦點的視窗被除能，這個視窗就不再能夠獲得焦點。系統會移除這個視窗的焦點，但卻不知道會給哪一個視窗。所以，最終情況是，所有視窗都失去焦點了！而一旦失去了焦點，使用者將不再能夠使用鍵盤來工作。爲了確保程式不會有這種狀況發生，我明確地將焦點設定給對話盒的子視窗（也可能是「孫」視窗）。我使用的是 SetFocus 和 GetNextDlgTabItem 函式：

```
// The button had the focus and has now been disabled so we must
// explicitly set focus to a different window.
SetFocus(GetNextDlgTabItem(hwnd, hwndCtl, FALSE));
```

TabStops.ICO

程式列表 2.4　TabStops.C

```
#0001  /****************************************************************
#0002  Module name: TabStops.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates dialog box tab ordering and adding/deleting controls.
#0006  ****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <Windows.h>
#0011  #include <Windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include "Resource.h"
#0014
#0015
#0016  //////////////////////////////////////////////////////////////////
#0017
#0018
#0019  void ModifyWndStyle (HWND hwnd, DWORD dwRemove, DWORD dwAdd,
#0020     BOOL fExtendedStyles) {
#0021
#0022     int nOffset = fExtendedStyles ? GWL_EXSTYLE : GWL_STYLE;
#0023     SetWindowLong(hwnd, nOffset,
#0024        (GetWindowLong(hwnd, nOffset) & ~dwRemove) | dwAdd);
#0025  }
#0026
#0027
#0028  //////////////////////////////////////////////////////////////////
#0029
#0030
#0031  BOOL DlgTabs_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0032
#0033     adgSETDLGICONS(hwnd, IDI_TABSTOPS, IDI_TABSTOPS);
#0034     return(TRUE);                // Accept default focus window.
#0035  }
#0036
#0037
#0038  //////////////////////////////////////////////////////////////////
#0039
#0040
#0041  void DlgTabs_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0042
#0043     HWND hwndCtlOld, hwndCtlNew;
#0044     RECT rc;
```

```
#0045
#0046    switch (id) {
#0047
#0048       case IDOK:
#0049       case IDCANCEL:            // Allows dialog box to close
#0050          EndDialog(hwnd, id);
#0051          break;
#0052
#0053       case IDC_ADDDLGCNTL:
#0054
#0055          // Determine the location and size of the existing static child window.
#0056          hwndCtlOld = GetDlgItem(hwnd, IDC_CTRLPRNT);
#0057          adgASSERT(IsWindow(hwndCtlOld));
#0058          GetWindowRect(hwndCtlOld, &rc);
#0059          MapWindowRect(HWND_DESKTOP, hwnd, &rc);   // Macro in windowsx.h
#0060
#0061          // Create the child dialog box window from the resource's template.
#0062          // NOTE: I pass NULL for the dialog box procedure address below
#0063          // because this is a sample application. DefDlgProc will do all the
#0064          // processing itself. Normally, you would pass the address of a
#0065          // dialog box procedure to handle messages for the child dialog.
#0066          hwndCtlNew = CreateDialog(GetWindowInstance(hwnd),
#0067             MAKEINTRESOURCE(IDD_CTRLPRNT), hwnd, NULL);
#0068          adgASSERT(IsWindow(hwndCtlNew));
#0069
#0070          // Position and size the dialog box child window so that it is in the
#0071          // exact same location as the existing static child window. NOTE: The
#0072          // dialog box child is placed immediately after the existing static
#0073          // window in the z-order so that the controls within it are properly
#0074          // located in the tabbing order.
#0075          SetWindowPos(hwndCtlNew, hwndCtlOld, rc.left, rc.top,
#0076             rc.right - rc.left, rc.bottom - rc.top, SWP_SHOWWINDOW);
#0077
#0078          // Make sure that the window has the WS_EX_CONTROLPARENT style so
#0079          // that IsDialogMessage considers the dialog box's children
#0080          // as the user navigates with the keyboard.
#0081          ModifyWndStyle(hwndCtlNew, 0, WS_EX_CONTROLPARENT, TRUE);
#0082
#0083          // Destroy the existing static child window that was only used
#0084          // for placement anyway.
#0085          DestroyWindow(hwndCtlOld);
#0086
#0087          // Disable the "Add dialog control" button since we have just
#0088          // done this and we do not allow it to be done a second time.
#0089          EnableWindow(hwndCtl, FALSE);
#0090
```

```
#0091         // We also must tell the "Add dialog control" button to redraw
#0092         // itself in the not default pushbutton state and to set the
#0093         // OK button as the new default pushbutton.
#0094         Button_SetStyle(hwndCtl,
#0095            GetWindowStyle(hwndCtl) & ~BS_DEFPUSHBUTTON | BS_PUSHBUTTON,
#0096               TRUE);
#0097         SendMessage(hwnd, DM_SETDEFID, IDOK, 0);
#0098
#0099         // The button had the focus and has now been disabled so we must
#0100         // explicitly set focus to a different window.
#0101         SetFocus(GetNextDlgTabItem(hwnd, hwndCtl, FALSE));
#0102         break;
#0103    }
#0104 }
#0105
#0106
#0107 ////////////////////////////////////////////////////////////////
#0108
#0109
#0110 BOOL WINAPI DlgTabs_DlgProc (HWND hwnd, UINT uMsg,
#0111    WPARAM wParam, LPARAM lParam) {
#0112
#0113    switch (uMsg) {
#0114
#0115       // Standard Window's messages
#0116       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG,  DlgTabs_OnInitDialog);
#0117       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    DlgTabs_OnCommand);
#0118    }
#0119    return(FALSE);              // We didn't process the message.
#0120 }
#0121
#0122
#0123 ////////////////////////////////////////////////////////////////
#0124
#0125
#0126 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0127    LPSTR lpszCmdLine, int nCmdShow) {
#0128
#0129    adgWARNIFUNICODEUNDERWIN95();
#0130    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_TABSTOPS),
#0131       NULL, DlgTabs_DlgProc));
#0132
#0133    return(0);
#0134 }
#0135
#0136 ////////////////////////////// End of File ///////////////////
```

程式列表 2.5　TabStops.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  ////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  ////////////////////////////////////////////////////////////////
#0044  //
```

```
#0045  // Dialog
#0046  //
#0047
#0048  IDD_TABSTOPS DIALOG DISCARDABLE  -32768, 5, 248, 106
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Tabstops"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      CONTROL        "Checkbox &1",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
#0054                     WS_TABSTOP,8,8,56,10
#0055      CONTROL        "Checkbox &2",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
#0056                     WS_TABSTOP,8,20,56,10
#0057      CONTROL        "Checkbox &3",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
#0058                     WS_TABSTOP,8,32,56,10
#0059      PUSHBUTTON      "&Add dialog control",IDC_ADDDLGCNTL,8,60,64,14
#0060      LTEXT          "Placeholder for control parent dialog box\n\nThis window is
#0061  destroyed when the ""Add dialog control"" button is pressed.",
#0062                    IDC_CTRLPRNT,80,8,104,92
#0063      DEFPUSHBUTTON   "OK",IDOK,192,4,50,14
#0064      PUSHBUTTON      "Cancel",IDCANCEL,192,21,50,14
#0065  END
#0066
#0067  IDD_CTRLPRNT DIALOG DISCARDABLE  0, 0, 104, 92
#0068  STYLE WS_CHILD | WS_VISIBLE
#0069  FONT 8, "MS Sans Serif"
#0070  BEGIN
#0071      LTEXT          "&Listbox:",IDC_STATIC,0,0,29,8
#0072      LISTBOX        IDC_LIST1,0,12,100,28,LBS_SORT | LBS_NOINTEGRALHEIGHT |
#0073                     WS_VSCROLL | WS_TABSTOP
#0074      LTEXT          "&Edit:",IDC_STATIC,0,48,16,8
#0075      EDITTEXT       IDC_EDIT1,0,60,100,32,ES_MULTILINE | ES_AUTOHSCROLL
#0076  END
#0077
#0078
#0079  /////////////////////////////////////////////////////////////////
#0080  //
#0081  // Icon
#0082  //
#0083
#0084  IDI_TABSTOPS           ICON    DISCARDABLE     "TabStops.ico"
#0085
#0086  #ifndef APSTUDIO_INVOKED
#0087  /////////////////////////////////////////////////////////////////
#0088  //
#0089  // Generated from the TEXTINCLUDE 3 resource.
#0090  //
```

```
#0091
#0092
#0093  /////////////////////////////////////////////////////////////////
#0094  #endif    // not APSTUDIO_INVOKED
```

程式列表 2.6　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by TabStops.rc
#0004  //
#0005  #define IDC_STATIC                   -1
#0006  #define IDC_CTRLPRNT                 101
#0007  #define IDD_TABSTOPS                 102
#0008  #define IDD_CTRLPRNT                 103
#0009  #define IDI_TABSTOPS                 104
#0010  #define IDC_CHECK1                   1000
#0011  #define IDC_CHECK2                   1001
#0012  #define IDC_CHECK3                   1002
#0013  #define IDC_LIST1                    1004
#0014  #define IDC_EDIT1                    1005
#0015  #define IDC_ADDDLGCNTL                1006
#0016
#0017  // Next default values for new objects
#0018  //
#0019  #ifdef APSTUDIO_INVOKED
#0020  #ifndef APSTUDIO_READONLY_SYMBOLS
#0021  #define _APS_NEXT_RESOURCE_VALUE       106
#0022  #define _APS_NEXT_COMMAND_VALUE        40001
#0023  #define _APS_NEXT_CONTROL_VALUE        1007
#0024  #define _APS_NEXT_SYMED_VALUE          101
#0025  #endif
#0026  #endif
```

第３章

對話盒的應用技巧

第２章中我曾介紹過一些系統提供的對話盒函式，使我們很容易產生視窗和子視窗。本章我將介紹一些有用的對話盒操控技巧，使你的程式功能更強大。每一個技巧我都以一個實例來說明，以便讓你有更清楚的認識。

擴大 / 縮小對話盒的技術

使用者學習一套新軟體需要花上不少時間。爲了幫助使用者學習，通常軟體一開始都會隱藏一些進階功能。只有比較熟練的使用者，才會用到這些進階功能。擴大 / 縮小對話盒這項技巧允許你將程式中的某些進階功能隱藏起來。例如 Windows 95 的【Choose Profile】對話盒（圖 **3.1**）就是使用了擴大 / 縮小對話盒技巧。

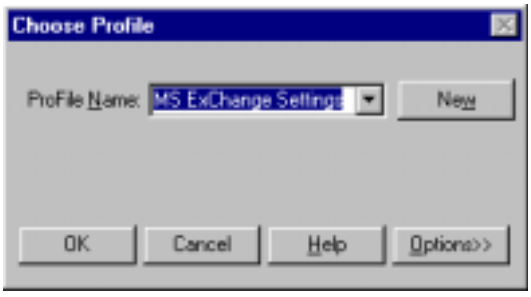


圖 3.1　Windows 95 的 Choose Profile 對話盒

這個對話盒顯示出使用者最常用到以及必須用到的選項。當使用者可按下【Options >>】鈕，對話盒會顯現出較進階的功能，而且對話盒會擴張開來顯示這些進階選項，如圖 3.2 所示。這個對話盒並不允許使用者將它恢復成原來的面貌（因爲【Options】鈕已經被除能了）。一般的對話盒除了提供擴展選項，同時也提供恢復（縮小）功能。如果你建立一個可擴展亦可縮小的對話盒，你可以自己決定如何設計它的操控模式。

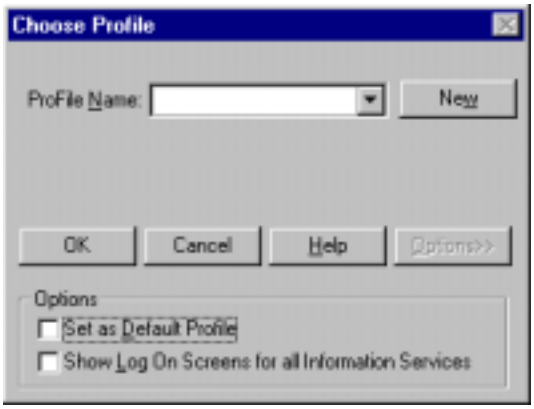


圖 3.2　擴展後的 Choose Profile 對話盒

Dialog Expand 程式實例

Dialog Expand 程式（DlgXpnd.EXE）的原始碼顯示於程式列表 3.3 ~ 3.5 中，示範如何建立一個可擴展可縮小的對話盒。當你執行這個程式，會顯示圖 3.3 的對話盒。這是一個尚未被擴展開來的對話盒。如果你想看到更多選項，可以按下【Options】鈕，對話盒便會擴展並顯示出一些進階選項，如圖 3.4。現在，使用者可以使用這些進階功能。如果使用者再按下【Options】鈕，便能恢復視窗的原本面貌。

此程式的所有檔案列於表 3.1 中。

圖 3.3　Dialog Expand 程式的對話盒

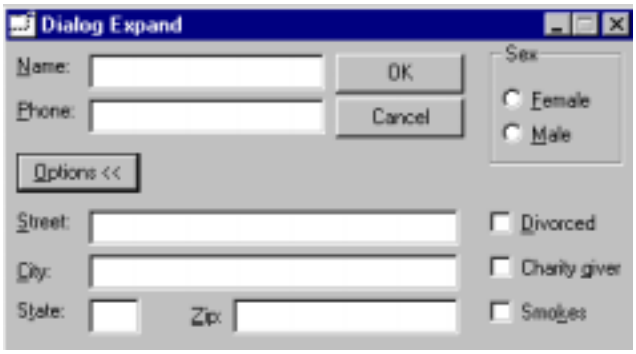


圖 3.4　擴展後的 Dialog Expand 對話盒

表 3.1　建立 DlgXpnd 程式所需的檔案

檔案	說明
DlgXpnd.C	內含 WinMain 函式、對話盒函式、訊息剖析函式。
ExpndBox.C	內含一堆函式，讓你很容易產生一個可擴展可縮小的對話盒。你可以把這個檔案拿到其他應用程式中使用。
ExpndBox.H	內含 ExpndBox.C 檔中所有函式的宣告。
DlgXpnd.RC	內含主視窗的對話盒面板（template）及其圖示（icon）。
Resource.H	內含 DlgXpnd.RC 檔中所有資源的 ID。
DlgXpnd.ICO	主視窗圖示。
DlgXpnd.MAK	Visual C++ 的 MAK 檔。

設計對話盒

要產生一個可擴展可縮小的對話盒，首先就是利用 Visual C++ 的對話盒編輯器來設計其外觀。你必須設計一個對話盒使它包含所有可能出現的控制元件。對話盒左上部份應該是一直都會出現的控制元件們，這一部份我把它稱爲預設區（default area），因爲無論如何這塊區域都是可見的。當對話盒展開，就會曝露出預設區下側以及右側的控制元件。

當所有控制元件都已安置好後，再加上一個 static 控制元件，我稱之爲預設盒子（default box）；;請改變其視窗風格，使它成爲一個黑色四方框，用以幫助我們辨認。這一個控制元件會被 DlgXpnd.C 中的函式所使用，因此它的編號必須是獨一無二的。

請把預設盒子（defult box）安置在對話盒的左上角，令它包含所有在預設狀態下可見的控制元件。**圖 3.5** 顯示對話盒及其預設盒子的最後結果。你可以看到，預設盒子的右邊線正位於【OK】鈕和【Cancel】鈕的右側，預設盒子的底線正位於【Option】鈕的下側。

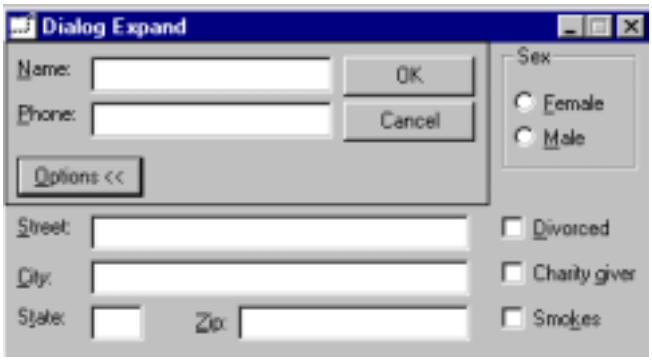


圖 3.5　對話盒及其預設盒子的最後結果

ExpandBox 函式

ExpandBox 函式負責對話盒的擴張和縮小。此一函式定義在 ExpndBox.C 檔中，函式原型如下：

```
ExpandBox(BOOL fExpand, HWND hwnd, int nIDDefaultBox);
```

第一個參數 fExpand，表示要擴張或是縮小。如果你想要擴張，應傳入 TRUE。第二個參數 hwnd，表示對話盒代碼。第三個參數 nIDDefaultBox，表示預設盒子（風格爲 SS_BLACKFRAME 之 static 視窗）的 ID。

當視窗被縮小，預設區以外的所有控制元件都必須被除能（disable）。這可以藉由以下迴路完成：

```
// Retrieve coordinates for the default child window.
 GetWindowRect(hwndDefaultBox, &rcDefaultBox);

 // Enable/disable all child windows outside of the default box.
 hwndChild = GetFirstChild(hwnd);
 for (; hwndChild != NULL; hwndChild = GetNextSibling(hwndChild)) {

    // Get rectangle occupied by child window in screen coordinates.
    GetWindowRect(hwndChild, &rcChild);

    if (!IntersectRect(&rcIntersection, &rcChild, &rcDefaultBox))
       EnableWindow(hwndChild, fExpand);
 }
```

針對對話盒中的每一個子視窗，我們都必須判斷它是否與預設盒子（一個矩形）有交集。如果沒有交集，子視窗可能處於致能（enable）狀態，也可以處於除能（disable）狀態 -- 視 fExpand 參數而定。將被切除之視窗上的控制元件除能，是非常重要的一件事，因爲如果使用者按下 Tab 鍵轉移鍵盤焦點，Windows 會尋找下一個有設立 WS_TABSTOP 風格且處於致能狀態的控制元件。雖然縮小後的對話盒會切除預設區以外的控制元件，但它們仍處於致能狀態，其控制元件仍有可能獲得鍵盤焦點，而使用者卻看不見。爲確保讓使用者僅能走訪預設區中的控制元件，所以 ExpandBox 函式將預設區以外的所有控

制元件都予以除能（disable）。

在這些控制元件被致能或除能後，ExpandBox 會根據 fExpand 參數來擴張或縮小對話盒。如果對話盒必須縮小，ExpandBox 會查詢預設盒子的位置來決定縮小後對話盒的大小。然而如果對話盒要擴張，ExpandBox 卻不易得知擴張後的對話盒大小。所以 ExpandBox 第一次必須以某種方式記錄對話盒萎縮前的尺寸。這些初始大小會被記錄在萎縮對話盒的額外位元組 GWL_USERDATA 中（寬度放在高位元組，高度放在低位元組）。ExpandBox 將對話盒縮小的動作如下：

```
GetWindowRect(hwnd, &rcWnd);
if (GetWindowLong(hwnd, GWL_USERDATA) == 0) {

   // This is the first time we are being called to shrink the dialog
   // box. The dialog box is currently in its expanded size and we must
   // save the expanded width and height so that it can be restored
   // later when the dialog box is expanded.
   SetWindowLong(hwndDefaultBox, GWL_USERDATA,
      MAKELONG(rcWnd.right - rcWnd.left, rcWnd.bottom - rcWnd.top));

   // We also hide the default box here so that it is not visible
   ShowWindow(hwndDefaultBox, SW_HIDE);
}

// Shrink the dialog box so that it encompasses everything from the top,
// left up to and including the default box.
SetWindowPos(hwnd, NULL, 0, 0,
   rcDefaultBox.right - rcWnd.left, rcDefaultBox.bottom - rcWnd.top,
   SWP_NOZORDER | SWP_NOMOVE);
```

程式碼首先檢查是否這欲縮小之對話盒的 GWL_USERDATA 值是否爲 0。如果是 0，表示這是第一次做萎縮動作，必須記錄萎縮前的對話盒尺寸。如果這是第一次萎縮，預設盒子還必須被隱藏起來，使用者才不會在螢幕上看到它。程式碼的最後一行，是爲了讓對話盒的大小和預設盒子一樣大。

擴張對話盒的動作比較簡單，只是從預設盒子的額外位元組 GWL_USERDATA 中取得原來對話盒大小並呼叫 SetWindowPos 函式：

```
// Expand the dialog box by restoring it to its original size.
DWORD dwDims = GetWindowLong(hwndDefaultBox, GWL_USERDATA);
SetWindowPos(hwnd, NULL, 0, 0,
   LOWORD(dwDims), HIWORD(dwDims), SWP_NOZORDER | SWP_NOMOVE);

// Make sure that the entire dialog box is visible on the user's screen.
SendMessage(hwnd, DM_REPOSITION, 0, 0);
```

最後呼叫 SendMessage，爲的是確保使用者能在螢幕上看到全部對話盒。想想看，假如呈萎縮狀態的對話盒位於螢幕右下角，而使用者這時候按下【Options】鈕，會發生什麼情況？當對話盒擴張開來，會有一些控制元件看不到（超越螢幕之外了），使用者必須移動對話盒才能夠看到完整的面貌。我們送出一個 DM_REPOSITION 訊息給對話盒，便可以由程式自動調整對話盒在螢幕上的位置，讓使用者清楚看到。

ExpandBox 函式可以在兩個地方被呼叫：第一個是在 DlgXpnd_OnInitDialog 函式中 -- 用以縮小對話盒。記住，這個對話盒第一次呈現是在萎縮狀態，所以程式一開始就呼叫 ExpandBox（並令 fExpand 爲 FALSE）是必要的。呼叫此函式同時也記錄了對話盒的大小，以便日後的擴張動作。

第二個呼叫出現在 DlgXpnd_OnCommand 函式中 -- 當使用者按下對話盒的【Options】鈕，程式便呼叫 ExpandBox。DlgXpnd_OnCommand 會先判斷目前是萎縮狀態還是擴張狀態，再呼叫 ExpandBox 以改變狀態。

DlgXpnd.ICO

程式列表 3.1　DlgXpnd.C

```
#0001  /******************************************************************
#0002  Module name: DlgXpnd.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates Expanding/Collapsing dialog box.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
```

```
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include "ExpndBox.h"
#0014  #include "resource.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  BOOL DlgXpnd_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0021
#0022     // During initialization, before any windows are shown, shrink the dialog
#0023     // box so that only the default portion is shown.
#0024     ExpandBox(FALSE, hwnd, IDC_DEFAULTBOX);
#0025
#0026     adgSETDLGICONS(hwnd, IDI_DLGXPND, IDI_DLGXPND);
#0027     return(TRUE);               // Accept default focus window.
#0028  }
#0029
#0030
#0031  ////////////////////////////////////////////////////////////////
#0032
#0033
#0034  void DlgXpnd_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0035
#0036     BOOL fExpanded;
#0037     int nLen;
#0038     TCHAR szText[20];
#0039
#0040     switch (id) {
#0041
#0042        case IDOK:
#0043        case IDCANCEL:            // Allows dialog box to close
#0044           EndDialog(hwnd, id);
#0045           break;
#0046
#0047        case IDC_OPTIONS:
#0048           if (codeNotify != BN_CLICKED)
#0049              break;
#0050
#0051           // Determine if the dialog is currently expanded.
#0052           nLen = Button_GetText(hwndCtl, szText, adgARRAY_SIZE(szText));
#0053           fExpanded = (szText[nLen - 1] == __TEXT('<'));
#0054
#0055           // User selected "Options >>" button, expand/shrink the dialog box.
```

```
#0056          ExpandBox(!fExpanded, hwnd, IDC_DEFAULTBOX);
#0057
#0058          // Set the focus to the desired control.
#0059          SetFocus(GetNextDlgTabItem(hwnd, hwndCtl, 0));
#0060
#0061          // Change the options button
#0062          lstrcpy(&szText[nLen - 2], fExpanded ? __TEXT(">>") : __TEXT("<<"));
#0063          Button_SetText(hwndCtl, szText);
#0064          break;
#0065       }
#0066    }
#0067
#0068
#0069    ////////////////////////////////////////////////////////////////
#0070
#0071
#0072    BOOL WINAPI DlgXpnd_DlgProc (HWND hwnd, UINT uMsg,
#0073       WPARAM wParam, LPARAM lParam) {
#0074
#0075       switch (uMsg) {
#0076
#0077          // Standard Window's messages
#0078          adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, DlgXpnd_OnInitDialog);
#0079          adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    DlgXpnd_OnCommand);
#0080       }
#0081       return(FALSE);              // We didn't process the message.
#0082    }
#0083
#0084
#0085    ////////////////////////////////////////////////////////////////
#0086
#0087
#0088    int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0089       LPSTR lpszCmdLine, int nCmdShow) {
#0090
#0091       adgWARNIFUNICODEUNDERWIN95();
#0092       adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_DLGXPAND),
#0093          NULL, DlgXpnd_DlgProc));
#0094
#0095       return(0);
#0096    }
#0097
#0098
#0099    ////////////////////////////////// End of File ///////////////////////
```

程式列表 3.2　ExpndBox.C

```
#0001  /************************************************************
#0002  Module name: ExpndBox.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Expanding/shrinking dialog box toolkit functions.
#0006  ************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include "ExpndBox.h"
#0014  #pragma warning(disable: 4001)   /* Single line comment */
#0015
#0016
#0017  //////////////////////////////////////////////////////////////
#0018
#0019
#0020  void ExpandBox (BOOL fExpand, HWND hwnd, int nIDDefaultBox) {
#0021
#0022    RECT rcWnd, rcDefaultBox, rcChild, rcIntersection;
#0023    HWND hwndChild, hwndDefaultBox = GetDlgItem(hwnd, nIDDefaultBox);
#0024
#0025    // Retrieve coordinates for the default child window.
#0026    GetWindowRect(hwndDefaultBox, &rcDefaultBox);
#0027
#0028    // Enable/disable all child windows outside of the default box.
#0029    hwndChild = GetFirstChild(hwnd);
#0030    for (; hwndChild != NULL; hwndChild = GetNextSibling(hwndChild)) {
#0031
#0032       // Get rectangle occupied by child window in screen coordinates.
#0033       GetWindowRect(hwndChild, &rcChild);
#0034
#0035       if (!IntersectRect(&rcIntersection, &rcChild, &rcDefaultBox))
#0036          EnableWindow(hwndChild, fExpand);
#0037    }
#0038
#0039    if (!fExpand) {
#0040       GetWindowRect(hwnd, &rcWnd);
#0041       if (GetWindowLong(hwnd, GWL_USERDATA) == 0) {
#0042
#0043          // This is the first time we are being called to shrink the dialog
#0044          // box. The dialog box is currently in its expanded size and we must
```

```
#0045         // save the expanded width and height so that it can be restored
#0046         // later when the dialog box is expanded.
#0047         SetWindowLong(hwndDefaultBox, GWL_USERDATA,
#0048            MAKELONG(rcWnd.right - rcWnd.left, rcWnd.bottom - rcWnd.top));
#0049
#0050         // We also hide the default box here so that it is not visible
#0051         ShowWindow(hwndDefaultBox, SW_HIDE);
#0052       }
#0053
#0054       // Shrink the dialog box so that it encompasses everything from the top,
#0055       // left up to and including the default box.
#0056       SetWindowPos(hwnd, NULL, 0, 0,
#0057          rcDefaultBox.right - rcWnd.left, rcDefaultBox.bottom - rcWnd.top,
#0058          SWP_NOZORDER | SWP_NOMOVE);
#0059    } else {
#0060
#0061       // Expand the dialog box by restoring it to its original size.
#0062       DWORD dwDims = GetWindowLong(hwndDefaultBox, GWL_USERDATA);
#0063       SetWindowPos(hwnd, NULL, 0, 0,
#0064          LOWORD(dwDims), HIWORD(dwDims), SWP_NOZORDER | SWP_NOMOVE);
#0065
#0066       // Make sure that the entire dialog box is visible on the user's screen.
#0067       SendMessage(hwnd, DM_REPOSITION, 0, 0);
#0068    }
#0069 }
#0070
#0071
#0072 ////////////////////////////// End of File ////////////////////
```

程式列表 3.3　ExpndBox.H

```
#0001 /************************************************************
#0002 Module name: ExpndBox.h
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Expanding/shrinking dialog box toolkit functions.
#0006 ************************************************************/
#0007
#0008
#0009 void ExpandBox (BOOL fExpand, HWND hwnd, int nIDDefaultBox);
#0010
#0011
#0012 ////////////////////////////// End of File ////////////////////
```

程式列表　3.4 DlgXpnd.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  ////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  ////////////////////////////////////////////////////////////////
#0044  //
```

```
#0045  // Dialog
#0046  //
#0047
#0048  IDD_DLGXPAND DIALOG DISCARDABLE  -32768, 5, 244, 110
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Dialog Expand"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      LTEXT          "&Name:",IDC_STATIC,4,4,22,8
#0054      EDITTEXT       IDC_NAME,32,4,92,13,ES_AUTOHSCROLL
#0055      LTEXT          "&Phone:",IDC_STATIC,4,20,24,8
#0056      EDITTEXT       IDC_PHONE,32,20,92,13,ES_AUTOHSCROLL
#0057      PUSHBUTTON     "&Options >>",IDC_OPTIONS,4,40,48,14
#0058      LTEXT          "&Street:",IDC_STATIC,4,60,23,8
#0059      EDITTEXT       IDC_STREET,32,60,144,13,ES_AUTOHSCROLL
#0060      LTEXT          "&City:",IDC_STATIC,4,78,16,8
#0061      EDITTEXT       IDC_CITY,32,76,144,13,ES_AUTOHSCROLL
#0062      LTEXT          "S&tate:",IDC_STATIC,4,92,21,8
#0063      EDITTEXT       IDC_STATE,32,92,20,13,ES_AUTOHSCROLL
#0064      LTEXT          "&Zip:",IDC_STATIC,72,94,14,8
#0065      EDITTEXT       IDC_ZIP,88,92,88,13,ES_AUTOHSCROLL
#0066      GROUPBOX       "Sex",IDC_STATIC,188,0,52,44
#0067      CONTROL        "&Female",IDC_FEMALE,"Button",BS_AUTORADIOBUTTON |
#0068                     WS_TABSTOP,192,16,35,10
#0069      CONTROL        "&Male",IDC_MALE,"Button",BS_AUTORADIOBUTTON |
#0070                     WS_TABSTOP,192,28,27,10
#0071      CONTROL        "&Divorced",IDC_DIVORCED,"Button",BS_AUTOCHECKBOX |
#0072                     WS_TABSTOP,188,60,41,10
#0073      CONTROL        "Charity &giver",IDC_CHARITYGIVER,"Button",
#0074                     BS_AUTOCHECKBOX | WS_TABSTOP,188,76,52,10
#0075      CONTROL        "Smo&kes",IDC_SMOKES,"Button",BS_AUTOCHECKBOX |
#0076                     WS_TABSTOP,188,92,37,10
#0077      CONTROL        "",IDC_DEFAULTBOX,"Static",SS_BLACKFRAME,0,0,184,56
#0078      DEFPUSHBUTTON   "OK",IDOK,128,4,50,14
#0079      PUSHBUTTON     "Cancel",IDCANCEL,128,20,50,14
#0080  END
#0081
#0082
#0083  ////////////////////////////////////////////////////////////////
#0084  //
#0085  // Icon
#0086  //
#0087
#0088  IDI_DLGXPND           ICON    DISCARDABLE     "DlgXpnd.ico"
#0089
#0090  #ifndef APSTUDIO_INVOKED
```

```
#0091  //////////////////////////////////////////////////////////////
#0092  //
#0093  // Generated from the TEXTINCLUDE 3 resource.
#0094  //
#0095
#0096
#0097  //////////////////////////////////////////////////////////////
#0098  #endif    // not APSTUDIO_INVOKED
```

程式列表 3.5　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by Touch.rc
#0004  //
#0005  #define IDD_TOUCH                   103
#0006  #define IDI_TOUCH                   104
#0007  #define IDC_UPDATECREATETIME          1000
#0008  #define IDC_UPDATELASTACCESSTIME       1001
#0009  #define IDC_UPDATEMODIFIEDTIME        1002
#0010  #define IDC_SYSTEMTIME               1003
#0011
#0012  // Next default values for new objects
#0013  //
#0014  #ifdef APSTUDIO_INVOKED
#0015  #ifndef APSTUDIO_READONLY_SYMBOLS
#0016  #define _APS_NEXT_RESOURCE_VALUE       106
#0017  #define _APS_NEXT_COMMAND_VALUE        40001
#0018  #define _APS_NEXT_CONTROL_VALUE        1006
#0019  #define _APS_NEXT_SYMED_VALUE         101
#0020  #endif
#0021  #endif
```

Modalless 對話盒技術（譯註：不是 modeless）

當你要使用 modal 對話盒，你可以呼叫 DialogBoxParam 函式產生之，然後對話盒中的控制元件就會被一一初始化。聽起來很簡單，但是對話盒如何知道其控制元件的初始值呢？每當使用者改變對話盒中的控制元件的內容，然後按下【OK】鈕，對話盒會讀取控制元件的內容並以某種方式告知程式。程式與對話盒之間的溝通非常不方便。通常，其溝通步驟如下：

1. 程式會定義一個資料結構來表示對話盒中所要顯示的資料。
2. 程式會配置一塊記憶體 （可能從堆疊中配置，作爲區域變數）來存放此資料結構內容，且將它初始化。
3. 程式呼叫 DialogBoxParam 並傳入初始化後的結構的位址，做爲函式的最後一個參數（lParamInit）。
4. 當對話盒收到 WM_INITDIALOG 訊息，其 lParam 參數即表示上述的結構位址。Dlg_OnInitDialog 可以存取此結構內容，並適當地初始化對話盒中的控制元件。
5. 在回返之前，Dlg_OnInitDialog 必須將此資料結構的位址記錄到對話盒的額外位元組 GWL_USERDATA 中，稍後步驟 6 會用到它。

這時候，使用者已經可以看見對話盒了，其中的所有控制元件也都已經初始化。當使用者改變對話盒的內容並按下【OK】鈕，對話盒必須將這些改變傳給程式：

6. 當 Dlg_OnCommand 得知【OK】鈕已按下，它會取得資料結構的位址。這個位址已被步驟 5 記錄在對話盒的額外位元組 GWL_USERDATA 之中。
7. 程式取得控制元件中的現值，並設定資料結構的欄位內容，以反應控制元件的現值。
8. 最後，Dlg_OnCommand 呼叫 EndDialog 摧毀對話盒，使程式可以繼續完成其餘工作。

對話盒被解散後，DialogBoxParam 會返回，於是我們可以從資料結構中得知使用者所做的設定。在早期的 Windows 程式設計中，這個程序甚至更繁複，但爲了使用 modal 對話盒，我們又不得不做。那個時候，沒有 DialogBoxParam 函式，也沒有額外位元組 GWL_USERDATA，我們必須將結構宣告爲一個全域變數，才能夠被程式和對話盒存取。

今天，即使 Windows API 已經發展了一段時間，這些步驟仍然相當冗長。如果有比較好的技術出現，無疑會受到相當的歡迎。在這一節中我要介紹一項我自己的技術，稱爲

modalless 對話盒，它可以使對話盒的溝通問題容易一些。

Modalless 對話盒是介於 modal 對話盒和 modeless 對話盒之間的一項產物。對作業系統來說，它是一個 modeless 對話盒，但對使用者而言，卻又很像一個 modal 對話盒。就程式設計者的觀點來看，此對話盒的產生和摧毀都像是個 modeless 對話盒，但其顯示和隱藏卻又像是個 modal 對話盒。

Modal 對話盒的真正問題在於當使用者按下【OK】鈕之後，其控制元件的內容以及對話盒的摧毀動作。如果我們產生一個 modeless 對話盒並且只有在使用者按下【OK】鈕之後才隱藏它，對話盒並未被摧毀，而程式也可以透過一些 Windows API 來查詢控制元件的內容。

Modalless 對話盒程式

Modalless 對話盒程式（DlgMdlls.EXE）的原始碼顯示於**程式列表 3.6 ~ 3.11**，示範如何產生並使用一個 modalless 對話盒。當你執行這個程式，主視窗如**圖 3.6** 所示。

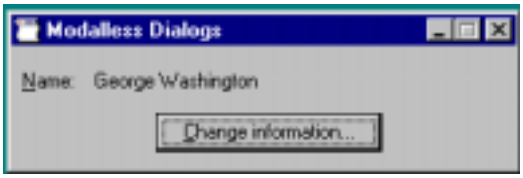


圖 3.6　Modalless 對話盒程式

這個視窗目前顯示了 George Washington 的名稱。然而你沒看到的還有一個 modeless 對話盒，它在程式剛開始執行時就已產生了（只是沒有設定 WS_VISIBLE 風格，所以你看不見）—也就是說，這個對話盒被隱藏了！

現在，如果使用者想把 George Washington 改變爲另一個名字，必須按下【Change information…】鈕，於是獲得**圖 3.7** 的對話盒。這個對話盒是一個 modalless 對話盒，但

對使用者來說，它卻像是個 modal 對話盒。使用者可以輸入不同的名字，然後按下【OK】或【Cancel】。假如使用者選擇了【OK】，主視窗便會更新對話盒中的名稱。如果選擇了【Cancel】，除了摧毀隱藏的 modalless 對話盒，不會做其它動作。

此程式所需用到的檔案，列在表 **3.2** 中。

圖 3.7 Change Information modalless 對話盒

表 3.2　DlgMdlls 程式所需檔案

檔案	說明
DlgMdlls.C	內含 WinMain 函式、對話盒函式、訊息剖析函式。
DlgMdlls.C	內含針對 modalless 對話盒而設計的訊息剖析函式。
Modalles.C	內含一堆函式，讓你更方便建構 modalless 對話盒。你可以在你自己的程式中使用這些函式。
Modalles.H	內含 Modalles.C 檔中所有函式的宣告。
DlgMdlls.RC	內含主視窗的對話盒面板（template）及圖示（icon）。
Resource.H	內含 DlgMdlls.RC 檔中所有資源的 ID。
DlgMdlls.ICO	主視窗圖示（icon）。
DlgMdlls.MAK	Visual C++ 的 MAK 檔。

當 DlgMdlls.EXE 開始執行，會產生一個 modal 對話盒。這個對話盒是程式的主視窗。當主視窗收到 WM_INITDIALOG 訊息，DlgMdlls_OnInitDialog 函式便去呼叫 Modalless_CreateDlg 函式。後者會呼叫 CreateDialogParam 以產生一個隱藏的 modalless 對話盒，並傳回其父視窗代碼。這個代碼會被存放在主視窗的 GWL_USERDATA 欄位中：

```
HWND hwndMdllsDlg = Modalless_CreateDlg(GetWindowInstance(hwnd),
```

```
   MAKEINTRESOURCE(IDD_DLGMDLSDLG), hwnd, MdllsDlg_DlgProc, 0);
   adgASSERT(IsWindow(hwndMdllsDlg));

   // Save handle to modalless dialog box so it can be shown later
   SetWindowLong(hwnd, GWL_USERDATA, (LONG) hwndMdllsDlg);
```

當使用者關閉這個程式，modalless 對話盒會被 Modalless_DestroyDlg 函式（其內部呼叫 DestroyWindow）摧毀：

```
void DlgMdlls_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{

   HWND hwndMdllsDlg = NULL;
    TCHAR szBuf[100];

    switch (id) {
       case IDCANCEL:            // Allows dialog box to close
          // If the main window is dying, destroy the modalless dialog box. Of
          // course, the system would do this for us automatically, but its
          // better for us to explicitly do this instead.
          Modalless_DestroyDlg((HWND) GetWindowLong(hwnd, GWL_USERDATA));
          EndDialog(hwnd, id);
          break;
       .
       .
       .
```

一旦 modalless 對話盒產生出來，其處理函式和 modal 對話盒的處理函式類似。**表 3.3** 說明了 modal 和 modalless 對話盒所使用的相同功能的函式。

表 3.3　modal 和 modalless 對話盒所用到的相同功能的函式

動作	Modal 對話盒 函式	Modalless 對話盒 函式
顯示對話盒	DialogBoxParam	Modalless_ShowDlg
摧毀對話盒	EndDialog	Modalless_EndDlg

當使用者按下【Change Information】鈕，程式會執行以下的碼，其中包含對

DlgMdlls_OnCommand 函式的呼叫：

```
// Get the handle to the modalless dialog box from the extra bytes.
hwndMdllsDlg = (HWND) GetWindowLong(hwnd, GWL_USERDATA);

// Initialize the edit control inside the modalless dialog box with
// the value from the main application window's static control.
GetDlgItemText(hwnd, IDC_NAME, szBuf, adgARRAY_SIZE(szBuf));
SetDlgItemText(hwndMdllsDlg, IDC_NAME, szBuf);

// Display the modalless dialog box to the user.
if (IDOK == Modalless_ShowDlg(hwndMdllsDlg)) {

   // If the user pressed the OK button, examine the modalless dialog
   // box's controls to get the user's modified values and update the
   // main window's data fields.
   GetDlgItemText(hwndMdllsDlg, IDC_NAME, szBuf, adgARRAY_SIZE(szBuf));
   SetDlgItemText(hwnd, IDC_NAME, szBuf);
}
```

程式首先從主視窗的 GWL_USERDATA 欄位中取得 modalless 對話盒代碼，然後以現有的(current)資訊來初始化 modalless 對話盒中的控制元件，然後呼叫 Modalless_ShowDlg 顯示 modalless 對話盒。如果使用者按下【OK】鈕，Modalless_ShowDlg 傳回 IDOK，而上面那段碼會設定主視窗的控制元件內容，以反映使用者的修改值。

讓我們詳細探討 Modalless_ShowDlg。這個函式需要 modalless 對話盒的視窗代碼。它藉著「將 modalless 對話盒之父視窗除能（disable）、顯示 modalless 對話盒、開始一個訊息迴路」的行爲，使自己看起來像一個 modal 對話盒。它的訊息迴路如下：

```
// Execute a message loop until the "ModallessResult" property becomes
// associated with the modalless dialog box. This happens when the dialog
// box function calls the EndModalLessDlg function.
do {

   // Wait for a message and process it.
   GetMessage(&msg, NULL, 0, 0);
   if (!IsDialogMessage(hwnd, &msg)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
```

```
   }

   // Get the value of "ModallessResult" property. If property does not
   // exist, GetProp returns zero.
   nResult = (int) GetProp(hwnd, g_szModallessResult);

   // Try to remove the property. If RemoveProp returns NULL, the property
   // is not associated with the window and the message loop must continue.
   // The debug version of Windows 95 warns that the line below removes
   // a property that doesn't exist.  This is by design.
} while (RemoveProp(hwnd, g_szModallessResult) == NULL);
```

訊息迴路不會結束 -- 直到 modalless 對話盒函式呼叫了 Modalless_EndDlg。這個函式會將一個 property 和一個 modalless 對話盒產生關連。GetMessage 訊息迴路會偵測是否這個 property 存在並取其內容。此內容即對話盒函式要傳回給「Modalless_ShowDlg 呼叫者」的回返值。

在訊息迴路中，GetProp 函式用來檢查是否 Modalless_EndDlg 函式已被呼叫過。如果 GetProp 函式傳回 0，表示視窗尚未設定 property，要不就是傳遞給 Modalless_EndDlg 的 nResult 參數是 0；我們可以使用 RemoveProp 函式來判定是哪一種情形。如果 RemoveProp 不能移除一個指名的 property，它會傳回 NULL，於是表示 Modalless_EndDlg 函式尚未被呼叫。如果移除了 property，接著就要結束訊息迴路、重新將父視窗致能（enable）、隱藏 modalless 對話盒、最後再把 GetProp 函式傳回的結果（nResult）傳回給程式。

```
if (IsWindow(hwndOwner)) {

   // Allow the owner window to receive keyboard and mouse input.
   EnableWindow(hwndOwner, TRUE);
}

// Hide the modalless dialog box and return the result.
ShowWindow(hwnd, SW_HIDE);
return(nResult);
```

現在，讓我們討論 modalless 對話盒何時做初始化動作。使用 Modalless_CreateDlg 函式來產生對話盒時，會送出 WM_INITDIALOG 訊息給對話盒函式，通告它做初始化動作。

當 Modalless_ShowDlg 呼叫 ShowWindow 以顯示對話盒時，系統會送出 WM_SHOWWINDOW 訊息給 modalless 對話盒函式。這正是對話盒函式在視窗顯示之前的初使化好時機。一般而言，每當顯示一個對話盒，鍵盤焦點通常會落在對話盒的第一個子視窗上，這是一般使用者所期望的：

```
void MdllsDlg_OnShowWindow (HWND hwnd, BOOL fShow, UINT status) {

   if (fShow) {

      // The modalless dialog box is becoming visible. Make the first
      // WS_TABSTOP child window have the focus.
      SetFocus(GetNextDlgTabItem(hwnd, NULL, FALSE));
   }
}
```

當使用者按下【OK】或【Cancel】鈕，MdllsDlg_OnCommand 函式會去呼叫 Modalless_EndDlg 函式：

```
BOOL WINAPI Modalless_EndDlg (HWND hwnd, int nResult) {

   // Setting the property also tells the mesasge loop to terminate.
   return(SetProp(hwnd, g_szModallessResult, (HANDLE) nResult));
}
```

這個函式將「視窗的一個 property」和「modalless 對話盒的回返值」產生關連。第一個參數是對話盒代碼，第二個參數是傳回值，將傳回給 Modalless_EndDlg 的呼叫者。如果 property 的設定沒有成功，Modalless_EndDlg 的傳回值將是 FALSE，而對話盒也就不會被摧毀。對 Modal 對話盒而言，Modalless_EndDlg 和 EndDialog 的動作極爲相似。

程式列表 3.6　DlgMdlls.C

DlgMdlls.ICO

```
#0001  /************************************************************
#0002  Module name: DlgMdlls.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Application to demonstrate modalless dialog boxes.
#0006  ************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include "resource.h"
#0014  #include "Modalles.h"
#0015  #pragma warning(disable: 4001)   /* Single line comment */
#0016
#0017
#0018  ////////////////////////////////////////////////////////////
#0019
#0020
#0021  // Dialog procedure for modalless dialog (contained in MdllsDlg.c)
#0022  extern BOOL WINAPI MdllsDlg_DlgProc (HWND hwnd, UINT uMsg,
#0023    WPARAM wParam, LPARAM lParam);
#0024
#0025
#0026  ////////////////////////////////////////////////////////////
#0027
#0028
#0029  BOOL DlgMdlls_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0030
#0031    HWND hwndMdllsDlg = Modalless_CreateDlg(GetWindowInstance(hwnd),
#0032       MAKEINTRESOURCE(IDD_DLGMDLSDLG), hwnd, MdllsDlg_DlgProc, 0);
#0033    adgASSERT(IsWindow(hwndMdllsDlg));
#0034
#0035    // Save handle to modalless dialog box so it can be shown later
#0036    SetWindowLong(hwnd, GWL_USERDATA, (LONG) hwndMdllsDlg);
#0037
#0038    adgSETDLGICONS(hwnd, IDI_DLGMDLS, IDI_DLGMDLS);
#0039    return(TRUE);               // Accept default focus window.
#0040  }
#0041
#0042
#0043  ////////////////////////////////////////////////////////////
#0044
#0045
#0046  void DlgMdlls_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
```

```
#0047
#0048    HWND hwndMdllsDlg = NULL;
#0049    TCHAR szBuf[100];
#0050
#0051    switch (id) {
#0052
#0053       case IDCANCEL:            // Allows dialog box to close
#0054          // If the main window is dying, destroy the modalless dialog box. Of
#0055          // course, the system would do this for us automatically, but its
#0056          // better for us to explicitly do this instead.
#0057          Modalless_DestroyDlg((HWND) GetWindowLong(hwnd, GWL_USERDATA));
#0058          EndDialog(hwnd, id);
#0059          break;
#0060
#0061       case IDC_CHANGEINFO:
#0062
#0063          // Get the handle to the modalless dialog box from the extra bytes.
#0064          hwndMdllsDlg = (HWND) GetWindowLong(hwnd, GWL_USERDATA);
#0065
#0066          // Initialize the edit control inside the modalless dialog box with
#0067          // the value from the main application window's static control.
#0068          GetDlgItemText(hwnd, IDC_NAME, szBuf, adgARRAY_SIZE(szBuf));
#0069          SetDlgItemText(hwndMdllsDlg, IDC_NAME, szBuf);
#0070
#0071          // Display the modalless dialog box to the user.
#0072          if (IDOK == Modalless_ShowDlg(hwndMdllsDlg)) {
#0073
#0074             // If the user pressed the OK button, examine the modalless dialog
#0075             // box's controls to get the user's modified values and update the
#0076             // main window's data fields.
#0077             GetDlgItemText(hwndMdllsDlg, IDC_NAME, szBuf, adgARRAY_SIZE(szBuf));
#0078             SetDlgItemText(hwnd, IDC_NAME, szBuf);
#0079          }
#0080          break;
#0081    }
#0082 }
#0083
#0084
#0085 ////////////////////////////////////////////////////////////////////
#0086
#0087
#0088 BOOL WINAPI DlgMdlls_DlgProc (HWND hwnd, UINT uMsg,
#0089    WPARAM wParam, LPARAM lParam) {
#0090
#0091    switch (uMsg) {
#0092
```

```
#0093       // Standard Window's messages
#0094       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, DlgMdlls_OnInitDialog);
#0095       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    DlgMdlls_OnCommand);
#0096    }
#0097    return(FALSE);              // We didn't process the message.
#0098 }
#0099
#0100
#0101 ////////////////////////////////////////////////////////////
#0102
#0103
#0104 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0105    LPSTR lpszCmdLine, int nCmdShow) {
#0106
#0107    adgWARNIFUNICODEUNDERWIN95();
#0108    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_DLGMDLS),
#0109       NULL, DlgMdlls_DlgProc));
#0110
#0111    return(0);
#0112 }
#0113
#0114
#0115 ////////////////////////////// End of File //////////////////////
```

程式列表 3.7　MdllsDlg.C

```
#0001 /******************************************************************
#0002 Module name: MdllsDlg.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Modalless dialog box.
#0006 ******************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)   /* Single line comment */
#0013 #include "resource.h"
#0014 #include "modalles.h"
#0015
#0016
#0017 ////////////////////////////////////////////////////////////////
#0018
#0019
```

```
#0020  void MdllsDlg_OnShowWindow (HWND hwnd, BOOL fShow, UINT status) {
#0021
#0022     if (fShow) {
#0023
#0024        // The modalless dialog box is becoming visible. Make the first
#0025        // WS_TABSTOP child window have the focus.
#0026        SetFocus(GetNextDlgTabItem(hwnd, NULL, FALSE));
#0027     }
#0028  }
#0029
#0030
#0031  ////////////////////////////////////////////////////////////////
#0032
#0033
#0034  BOOL MdllsDlg_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0035
#0036     return(TRUE);               // Accept default focus window.
#0037  }
#0038
#0039
#0040  ////////////////////////////////////////////////////////////////
#0041
#0042
#0043  void MdllsDlg_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0044
#0045     switch (id) {
#0046
#0047        case IDOK:
#0048        case IDCANCEL:            // Allows dialog box to close
#0049           Modalless_EndDlg(hwnd, id);
#0050           break;
#0051     }
#0052  }
#0053
#0054
#0055  ////////////////////////////////////////////////////////////////
#0056
#0057
#0058  BOOL WINAPI MdllsDlg_DlgProc (HWND hwnd, UINT uMsg,
#0059     WPARAM wParam, LPARAM lParam) {
#0060
#0061     switch (uMsg) {
#0062
#0063        // Standard Window's messages
#0064        adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, MdllsDlg_OnInitDialog);
#0065        adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    MdllsDlg_OnCommand);
```

```
#0066
#0067       // NOTE: Normally, if a dialog box procedure processes a message it
#0068       // returns TRUE and DefDlgProc does not do additional processing.
#0069       // However, for the WM_SHOWWINDOW message, we need to do some processing
#0070       // AND we also need the system to do its default processing (which
#0071       // shows/hides the window). So, this message must have special handling.
#0072       case WM_SHOWWINDOW:
#0073          HANDLE_WM_SHOWWINDOW(hwnd, wParam, lParam, MdllsDlg_OnShowWindow);
#0074          return(FALSE);          // Return FALSE to DefDlgProc.
#0075    }
#0076    return(FALSE);               // We didn't process the message.
#0077 }
#0078
#0079
#0080 ////////////////////////////// End of File ///////////////////////
```

程式列表 3.8　Modalles.C

```
#0001 /******************************************************************
#0002 Module name: Modalles.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Modalless dialog box toolkit functions.
#0006 ******************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"         /* See Appendix A for details */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)    /* Single line comment */
#0013 #include "modalles.h"
#0014
#0015
#0016 ////////////////////////////////////////////////////////////////////
#0017
#0018
#0019 // Internal property string used to associate the dialog box's result value.
#0020 static TCHAR g_szModallessResult[] = __TEXT("ModallessResult");
#0021
#0022
#0023 ////////////////////////////////////////////////////////////////////
#0024
#0025
#0026 HWND WINAPI Modalless_CreateDlgA (HINSTANCE hinst, LPCSTR szTemplateName,
#0027    HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit) {
```

```
#0028
#0029    return(CreateDialogParamA(hinst, szTemplateName, hwndOwner,
#0030       pfnDlgProc, lParamInit));
#0031 }
#0032
#0033
#0034 ////////////////////////////////////////////////////////////////
#0035
#0036
#0037 HWND WINAPI Modalless_CreateDlgW (HINSTANCE hinst, LPCWSTR szTemplateName,
#0038    HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit) {
#0039
#0040    return(CreateDialogParamW(hinst, szTemplateName, hwndOwner,
#0041       pfnDlgProc, lParamInit));
#0042 }
#0043
#0044
#0045 ////////////////////////////////////////////////////////////////
#0046
#0047
#0048 BOOL WINAPI Modalless_DestroyDlg (HWND hwnd) {
#0049
#0050    return(DestroyWindow(hwnd));
#0051 }
#0052
#0053
#0054 ////////////////////////////////////////////////////////////////
#0055
#0056
#0057 int WINAPI Modalless_ShowDlg (HWND hwnd) {
#0058
#0059    HWND hwndOwner = GetParent(hwnd);
#0060    int nResult = 0;
#0061    MSG msg;
#0062
#0063    if (IsWindow(hwndOwner)) {
#0064
#0065       // Stop the owner window from receiving keyboard or mouse input.
#0066       EnableWindow(hwndOwner, FALSE);
#0067    }
#0068
#0069    // Display the modalless dialog box by sending a WM_SHOWWINDOW message to
#0070    // the dialog box function.
#0071    ShowWindow(hwnd, SW_SHOW);
#0072
#0073    // Execute a message loop until the "ModallessResult" property becomes
```

```
#0074    // associated with the modalless dialog box. This happens when the dialog
#0075    // box function calls the EndModalLessDlg function.
#0076    do {
#0077
#0078       // Wait for a message and process it.
#0079       GetMessage(&msg, NULL, 0, 0);
#0080       if (!IsDialogMessage(hwnd, &msg)) {
#0081          TranslateMessage(&msg);
#0082          DispatchMessage(&msg);
#0083       }
#0084
#0085       // Get the value of "ModallessResult" property. If property does not
#0086       // exist, GetProp returns zero.
#0087       nResult = (int) GetProp(hwnd, g_szModallessResult);
#0088
#0089       // Try to remove the property. If RemoveProp returns NULL, the property
#0090       // is not associated with the window and the message loop must continue.
#0091       // The debug version of Windows 95 warns that the line below removes
#0092       // a property that doesn't exist.  This is by design.
#0093    } while (RemoveProp(hwnd, g_szModallessResult) == NULL);
#0094
#0095    if (IsWindow(hwndOwner)) {
#0096
#0097       // Allow the owner window to receive keyboard and mouse input.
#0098       EnableWindow(hwndOwner, TRUE);
#0099    }
#0100
#0101    // Hide the modalless dialog box and return the result.
#0102    ShowWindow(hwnd, SW_HIDE);
#0103    return(nResult);
#0104 }
#0105
#0106
#0107 ////////////////////////////////////////////////////////////////
#0108
#0109
#0110 BOOL WINAPI Modalless_EndDlg (HWND hwnd, int nResult) {
#0111
#0112    // Setting the property also tells the mesasge loop to terminate.
#0113    return(SetProp(hwnd, g_szModallessResult, (HANDLE) nResult));
#0114 }
#0115
#0116
#0117 ////////////////////////////// End of File ///////////////////
```

程式列表 3.9　Modalles.H

```
#0001  /****************************************************************
#0002  Module name: Modalles.h
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Modalless dialog box toolkit functions.
#0006  ****************************************************************/
#0007
#0008
#0009  HWND WINAPI Modalless_CreateDlgA (HINSTANCE hinst, LPCSTR szTemplateName,
#0010    HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
#0011
#0012  HWND WINAPI Modalless_CreateDlgW (HINSTANCE hinst, LPCWSTR szTemplateName,
#0013    HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
#0014
#0015  #ifdef UNICODE
#0016  #define Modalless_CreateDlg Modalless_CreateDlgW
#0017  #else
#0018  #define Modalless_CreateDlg Modalless_CreateDlgA
#0019  #endif // !UNICODE
#0020
#0021
#0022  ////////////////////////////////////////////////////////////////
#0023
#0024
#0025  int  WINAPI Modalless_ShowDlg (HWND hwnd);
#0026  BOOL WINAPI Modalless_EndDlg (HWND hwnd, int nResult);
#0027  BOOL WINAPI Modalless_DestroyDlg (HWND hwnd);
#0028
#0029
#0030  ////////////////////////////////// End of File ///////////////////
```

程式列表 3.10　DlgMdlls.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
```

```
#0010  #include "windows.h"
#0011
#0012  /////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  /////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  /////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  /////////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_DLGMDLS DIALOG DISCARDABLE  -32768, 5, 196, 44
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Modalless Dialogs"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      LTEXT          "&Name:",IDC_STATIC,4,8,22,8
#0054      LTEXT          "George Washington",IDC_NAME,32,8,160,8
#0055      DEFPUSHBUTTON   "&Change information...",IDC_CHANGEINFO,56,24,88,14
```

```
#0056  END
#0057
#0058  IDD_DLGMDLSDLG DIALOG DISCARDABLE  0, 0, 186, 42
#0059  STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
#0060  CAPTION "Modalless Dialog Box"
#0061  FONT 8, "MS Sans Serif"
#0062  BEGIN
#0063      LTEXT          "&Name:",IDC_STATIC,4,4,22,8
#0064      EDITTEXT       IDC_NAME,28,4,152,13,ES_AUTOHSCROLL
#0065      DEFPUSHBUTTON   "OK",IDOK,40,24,50,14
#0066      PUSHBUTTON     "Cancel",IDCANCEL,108,24,50,14
#0067  END
#0068
#0069
#0070  //////////////////////////////////////////////////////////////////
#0071  //
#0072  // Icon
#0073  //
#0074
#0075  IDI_DLGMDLS           ICON    DISCARDABLE     "DlgMDlls.ico"
#0076
#0077  #ifndef APSTUDIO_INVOKED
#0078  //////////////////////////////////////////////////////////////////
#0079  //
#0080  // Generated from the TEXTINCLUDE 3 resource.
#0081  //
#0082
#0083
#0084  //////////////////////////////////////////////////////////////////
#0085  #endif    // not APSTUDIO_INVOKED
```

程式列表 3.11　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by DlgMdlls.rc
#0004  //
#0005  #define IDD_DLGMDLS                 101
#0006  #define IDI_DLGMDLS                 102
#0007  #define IDD_DLGMDLSDLG               113
#0008  #define IDC_NAME                    1000
#0009  #define IDC_CHANGEINFO               1001
#0010  #define IDC_STATIC                  -1
#0011
#0012  // Next default values for new objects
```

```
#0013  //
#0014  #ifdef APSTUDIO_INVOKED
#0015  #ifndef APSTUDIO_READONLY_SYMBOLS
#0016  #define _APS_NEXT_RESOURCE_VALUE       103
#0017  #define _APS_NEXT_COMMAND_VALUE        40001
#0018  #define _APS_NEXT_CONTROL_VALUE        1002
#0019  #define _APS_NEXT_SYMED_VALUE          101
#0020  #endif
#0021  #endif
```

對話盒的動態技術

有時候我們需要在程式執行時期動態製造出一個對話盒。舉個例，一個資料庫程式可能允許使用者設計一個表格（form），用來將資料加進資料庫中。另一個例子是 Visual C++ 的對話盒編輯器，允許使用者製造一個對話盒，並且不需經過任何編譯動作就可以測試此對話盒。本節的對話盒動態技術告訴你如何在執行期間建立對話盒。

WIN32 API 提供兩個函式，在程式執行時期產生對話盒。DialogBoxIndirectParam 用來產生一個 modal 對話盒：

```
int DialogBoxIndirectParam(HINSTANCE hinst, LPCDLGTEMPLATE hDlgTemplate,
  HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
```

CreateDialogIndirectParam 用來產生一個 modeless 對話盒：

```
HWND CreateDialogIndirectParam(HINSTANCE hinst, LPCDLGTEMPLATE lpDlgTemplate,
                      HWND hwndOwner, DLGPROC pfnDlgProc, LPARAM lParamInit);
```

動態對話盒的產生是先在記憶體中建立一個對話盒面板（template），再將此記憶體區塊的位址傳給上述兩個函式，做爲 lpDlgTemplate 參數。

函式回返之後，你可以釋放記憶體區塊。第二章談到設計對話盒外觀的那一節中，我曾經描述過對話盒面板的二進位碼（binary image）看起來長什麼樣子。我曾經在那裡討論過如何使用 Visual C++ 來設計對話盒，再使用資源編譯器來編譯資源描述檔（.RC）並與可執行檔結合在一起。

經過那樣的討論，我們可以很清楚知道如何在執行時期建立一個對話盒。我們所要做的是配置一塊記憶體並放進一個 DLGTEMPLATEEX 結構、一個可有可無的 FONTINFOEX 結構、以及零個或多個 DLGITEMTEMPLATEEX 結構（每一個子視窗需要一個）。你可能需要回頭參考第二章以喚回你的記憶。一旦記憶體區塊被初始化，就可以去呼叫上述對話盒函式，產生出對話盒。

動態對話盒程式實例

動態對話盒程式（DlgDyn.EXE）的原始碼在**程式列表 3.12 ~ 3.17** 中，示範如何在程式執行期間動態產生對話盒。當你執行這個程式，主視窗如**圖 3.8**。

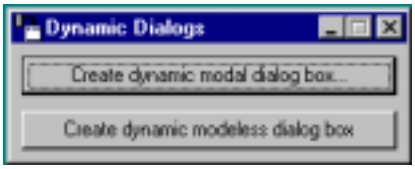


圖 3.8　動態對話盒程式

當你選擇其中的一個項目，記憶體中會產生一個動態對話盒，然後顯示到螢幕上。假如你選擇的是【Create dynamic modal dialog box】，會產生一個 modal 對話盒如**圖 3.9**，如果你選擇的是【Create dynamic modeless dialog box】，會產生一個 modeless 對話盒如**圖 3.10**。如果產生的是 modeless 對話盒，你會發現【Create dynamic modal dialog box】按鈕會改變爲【Destroy dynamic modeless dialog box】，你可以使用此按鈕來移除 modeless 對話盒。

建立此程式所需用到的檔案，列於**表 3.4** 中。

圖 3.9　動態產生之 modal 對話盒

圖 3.10　動態產生之 modeless 對話盒

表 3.4　DlgDyn 程式所需檔案

檔案	說明
DlgDyn.C	內含 WinMain 函式、對話盒函式、訊息剖析函式。
DlgDlg.C	內含動態產生之 modal 對話盒和 modeless 對話盒的對話盒函式，及其訊息剖析函式。
DlgTmplt.C	內含在記憶體中建構動態對話盒面板(template)的相關函式。
DlgTmplt.H	內含 DlgTmplt.C 檔中所有函式的宣告。
DlgDyn.RC	內含主視窗的對話盒面板（template）及其圖示。
Resource.H	內含 DlgDyn.RC 檔中所有資源的 ID。
DlgDyn.ICO	主視窗圖示（icon）。
DlgDyn.MAK	Visual C++ 的 MAK 檔。

當你選擇【Create Dynamic modal dialog box】，DlgDyn_OnCommand 函式（在 DlgDyn.C 檔中）會執行以下動作：

```
case IDC_MODAL:
   pDlgTemplate = BuildDynamicDlgBox(FALSE);
   adgASSERT(pDlgTemplate != NULL);
   adgVERIFY(-1 != DialogBoxIndirect(GetWindowInstance(hwnd),
      (LPDLGTEMPLATE) pDlgTemplate, hwnd, DynDlg_DlgProc));
   DlgTemplate_Free(pDlgTemplate);
   break;
```

BuildDynamicDlgBox 函式（在 DynDlg.C 檔中）負責在記憶體中產生對話盒板（template）的大部份工作，它會配置一塊記憶體並初始化 DLGTEMPLATEEX 結構、FONTINFOEX 結構、和一串 DLGITEMTEMPLATEEX 結構。BuildDynamicDlgBox 存取這些資料結構是爲了描述對話盒和其所有子視窗的屬性。BuildDynamicDlgBox 建立的面板既適用於 modal 對話盒也適用於 modeless 對話盒。假如你想要這個面板是 modal 對話盒，請傳 FALSE 給 BuildDynamicDlgBox，如果你想要這個面板是 modeless 對話盒，則傳 TRUE 給 BuildDynamicDlgBox。

如果 BuildDynamicDlgBox 成功地執行完畢，會傳回面板的記憶體位址，此記憶體位址會傳回給 DialogBoxIndirect，用來剖析記憶體內容並產生視窗和其子視窗，接下來就進入函式內部的 GetMessage 訊息迴路，等待對話盒函式呼叫 EndDialog 結束對話盒。對話盒結束後便呼叫 DlgTemplate_Free（在 DlgTmplt.C 中）釋放此記憶體區塊。

這個程式所產生的 modeless 對話盒和之前所產生的幾乎一樣，但是它以 CreateDialogIndirect 替代 DialogBoxIndirect 以產生對話盒。並且，這個 modeless 對話盒代碼存放在主視窗的額外位元組 GWL_USRDATA 之中，當使用者按下【Destroy dynamic modeless dialog box】按鈕，modeless 對話盒便會被摧毀。

製作對話盒面板（Dialog Box Template）

BuildDynamicDlgBox 用了 DlgTmplt.C 檔中的許多函式來製作對話盒面板，唯一的參數是 fModeless，那是一個布林（Boolean）值，表示要製作的是 modal 對話盒還是 modeless 對話盒的面板。BuildDynamicDlgBox 首先呼叫 DlgTemplate_Create 函式（在 DlgTmplt.C 檔中）：

```
// Create the dynamic dialog box header information.
if (fModeless) {

   pDlgTemplate = DlgTemplate_Create(
      WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | WS_VISIBLE |
      WS_POPUP | DS_SETFONT, 0, 0, 6, 26, 216, 72, __TEXT(""), __TEXT(""),
      __TEXT("Dynamic Modeless Demo"), 12, FW_NORMAL, TRUE,
      __TEXT("Courier New"));
} else {

   pDlgTemplate = DlgTemplate_Create(
      WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | WS_VISIBLE |
      WS_POPUP, 0, 0, 6, 26, 216, 72, __TEXT(""), __TEXT(""),
      __TEXT("Dynamic Modal Demo"), 0, 0, 0, NULL);
}
```

DlgTemplate_Create 會配置一個足以放置 DLGTEMPLATEEX 結構、FONTINFOEX 結構（如果有指定 DS_SETFONT 風格）、視窗選單、類別、標題欄字串的記憶體區塊，並傳回區塊位址。

由於指定了 DS_SETFONT 風格，當產生一個 modeless 對話盒，最後四個參數（12, FW_NORMAL, TRUE, 和 Courier New 字型）將表示對話盒產生時所使用的字型。但是 DS_SETFONT 風格不能指定給 modal 對話盒，所以 DlgTemplate_Create 會忽略最後四個參數。對話盒中欲顯示出來的控制元件，其個數並不會指定給 DlgTemplate_Create；通常其初值爲 0，每次呼叫 DlgTemplate_AddControl（用以增加控制元件）才累加 1。

每當一個 DLGTEMPLATEEX 結構被放進記憶體區塊中，就相當於附加一個控制元件。BuildDynamicDlgBox 使用下面這個資料結構來產生那些控制元件：

```
// The table that describes the dialog box controls follow.
#define MAXDLGITEMDATA (10)

static struct {
   short   x, y, cx, cy;
   DWORD   dwStyle, dwExStyle, dwHelpId;
   int     id;
   LPCTSTR szClass, szText;
   WORD    wExtraDataCount;
   PBYTE   pbExtraData;
} DynDlgBoxData[] = {

   { 4, 4, 32, 12, SS_LEFT, 0, 0, IDC_STATIC,
      __TEXT("STATIC"), __TEXT("&Edit:"), 0, NULL },

   { 40, 4, 64, 12, ES_LEFT | WS_BORDER | WS_TABSTOP | WS_GROUP, 0, 0,
      IDC_EDITBOX, __TEXT("EDIT"), __TEXT(""), 0, NULL },
   .
   .
   .
   // End of list marker (szClass must be null)
   { 0, 0, 0, 0, 0, 0, 0, 0, NULL, NULL, 0, NULL }
};
```

每次欲增加一個控制元件，便呼叫 DlgTemplate_AddControl 一次，新的 DLGITEMTEMPLATEEX 結構會被附加到記憶體區塊尾端：

```
// Add each of the controls in the DynamicDlgBoxData array.
for (x = 0; DynDlgBoxData[x].szClass != NULL; x++) {

   // Do not add an OK button for a modeless dialog box.
   if (fModeless) {
      if (DynDlgBoxData[x].id == IDOK)
         continue;
   }

   fCtrlAddedOK = DlgTemplate_AddControl(&pDlgTemplate,
      DynDlgBoxData[x].x,
      DynDlgBoxData[x].y,
      DynDlgBoxData[x].cx,
      DynDlgBoxData[x].cy,
      DynDlgBoxData[x].id,
      DynDlgBoxData[x].dwHelpId,
```

```
      DynDlgBoxData[x].dwStyle | WS_VISIBLE,
      DynDlgBoxData[x].dwExStyle,
      DynDlgBoxData[x].szClass,
      DynDlgBoxData[x].szText,
      DynDlgBoxData[x].wExtraDataCount,
      DynDlgBoxData[x].pbExtraData);

   if (!fCtrlAddedOK)
      break;
}
```

由於 modeless 對話盒通常不會有【OK】鈕，所以之前的程式碼並沒有呼叫 DlgTemplate_AddControl 來產生此按鈕。

對話盒面板之記憶體區塊管理

用以管理對話盒面板記憶體區塊的函式主要有 DlgTemplate_CreateA，DlgTemplate_CreateW、DlgTemplate_AddControlA、DlgTemplate_AddControlW、DlgTemplate_Free 等函式。

DlgTemplate_CreateA 和 DlgTemplate_AddControlA 很容易瞭解，所以我先說明它們。這些函式接受 ANSI 字串，並將之轉換爲 Unicode 字串，然後再去呼叫對應的 Unicode 函式 -- 那是真正執行任務的 Unicode 版函式。而 ANSI 版函式只不過是個轉換層（thunking layers）。還記得嗎，Windows NT 和 Windows 95 都期望所有的資源都使用 Unicode，所以在 Unicode 函式中完成真正的工作是比較理想的。

DlgTemplate_CreateW 配置一塊記憶體。這塊記憶體必須足以容納 DLGTEMPLATEEX 結構及其三個字串欄位：選單、類別和標題。如果這個對話盒面板指定了 DS_SETFONT 風格，記憶體區塊還必須足以容納 FONTINFOEX 結構，及其 typeface 字串欄位。

記憶體區塊被配置妥當後，DlgTemplate_CreateW 會依據傳入的參數值填入此記憶體區塊的各個欄位中，並將 cDlgItems 的值設爲 0，然後將記憶體區塊的位址傳回給其呼叫者。

DlgTemplate_AddControlW 用來添加一個新的控制元件，放到由 DlgTemplate_CreateW 所

產生的記憶體區塊的尾端。函式首先計算需要多少個位元組才能夠容納DLGITEMTEMPLATEEX 結構和其三個不定長欄位：類別名稱、標題、初始化資料位元組。在將先前配置的記憶體區塊擴大以容納我們想加入的控制元件之前，我們必須先知道有多少個位元組已被使用。C-Runtime 的 _msize 函式可以傳回目前記憶體區塊的大小。現在，DlgTemplate_AddControlW 可以呼叫 C-Runtime 的 realloc 函式來擴大記憶體區塊，使其容納新加入的控制元件：

```
// Increase the size of the memory block to include the new dialog item.
nBytesBeforeAddingControl = _msize(*ppDlgTemplate);
pDlgTemplateNew = realloc(*ppDlgTemplate,
   nBytesBeforeAddingControl + nBytesForNewControl);
if (pDlgTemplateNew == NULL)
   return(FALSE);
```

由於 realloc 函式不需要傳回相同的記憶體位址，所以 DlgTemplate_AddControlW 必須將這個新位址告知其呼叫者。這也就是爲什麼 DlgTemplate_AddControlW 的第一個參數是一個「面板指標的指標」，而不僅僅是一個「面板指標」而已。新的記憶體位址會被傳回來更新這個指標，以至於現在指標指向新的面板。

當記憶體區塊大到足以容納加入的控制元件時，新的 DLGITEMTEMPLATEEX 結構會被加入區塊之中（放在已被使用過的位元組之後）。如果有一個新的控制元件被加入，DLGTEMPLATEEX 中的 cDlgItems 欄位（記錄著有多少個控制元件）也會隨之增加。

一旦程式將控制元件加到記憶體中的對話盒面板，DialogBoxIndirectParam 和 CreateDialogIndirectParam 兩函式便能夠使用這記憶體中的對話盒面板來產生對話盒。

DlyDyn.ICO

程式列表 3.12　DlgDyn.C

```
#0001  /******************************************************************
#0002  Module name: DlgDyn.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates dynamic modeless and modal dialog boxes.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include "resource.h"
#0014  #include "dlgtmplt.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  // Function (contained in dyndlg.c) that demonstrates how to create a dynamic
#0021  // dialog box template.
#0022  extern PDLGTEMPLATEEX BuildDynamicDlgBox (BOOL fModeless);
#0023
#0024  // Dialog box procedure (contained in dyndlg.c) for both the modal and modeless
#0025  // dynamic dialog boxes.
#0026  extern BOOL WINAPI DynDlg_DlgProc (HWND hwnd, UINT uMsg,
#0027    WPARAM wParam, LPARAM lParam);
#0028
#0029
#0030  ////////////////////////////////////////////////////////////////
#0031
#0032
#0033  // When a modeless dialog box is invoked from a modal dialog box, the modeless
#0034  // dialog box's keyboard interface will not function. This is because the
#0035  // GetMessage loop inside DialogBox does not call IsDialogMessage for the
#0036  // modeless dialog box. In order to get the modeless dialog box's keyboard
#0037  // interface working, I install a message filter (WH_MSGFILTER) hook. I store
#0038  // the dialog's window handle in the variable below so that the message filter
#0039  // hook function has access to the modeless dialog box.
#0040  HWND g_hwndModeless = NULL;
#0041
#0042  // This is hook handle that identifies the WH_MSGFITLER hook used to get
#0043  // keyboard processing to the modeless dialog box.
#0044  HHOOK g_hhookMsgFilter = NULL;
```

```
#0045
#0046
#0047 ////////////////////////////////////////////////////////////////
#0048
#0049
#0050 BOOL DlgDyn_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0051
#0052    adgSETDLGICONS(hwnd, IDI_DLGDYN, IDI_DLGDYN);
#0053    return(TRUE);               // Accept default focus window.
#0054 }
#0055
#0056
#0057 ////////////////////////////////////////////////////////////////
#0058
#0059
#0060 void DlgDyn_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0061
#0062    PDLGTEMPLATEEX pDlgTemplate;
#0063
#0064    switch (id) {
#0065
#0066       case IDOK:
#0067       case IDCANCEL:            // Allows dialog box to close
#0068          EndDialog(hwnd, id);
#0069          break;
#0070
#0071       case IDC_MODAL:
#0072          pDlgTemplate = BuildDynamicDlgBox(FALSE);
#0073          adgASSERT(pDlgTemplate != NULL);
#0074          adgVERIFY(-1 != DialogBoxIndirect(GetWindowInstance(hwnd),
#0075             (LPDLGTEMPLATE) pDlgTemplate, hwnd, DynDlg_DlgProc));
#0076          DlgTemplate_Free(pDlgTemplate);
#0077          break;
#0078
#0079       case IDC_MODELESS:
#0080          if (IsWindow(g_hwndModeless)) {
#0081
#0082             // The modeless window is already up, let's destroy it.
#0083             DestroyWindow(g_hwndModeless);
#0084             g_hwndModeless = NULL;
#0085          } else {
#0086
#0087             // The modeless window is not up, let's create it.
#0088             pDlgTemplate = BuildDynamicDlgBox(TRUE);
#0089             adgASSERT(pDlgTemplate != NULL);
#0090             g_hwndModeless = CreateDialogIndirect(GetWindowInstance(hwnd),
```

```
#0091              (LPDLGTEMPLATE) pDlgTemplate, hwnd, DynDlg_DlgProc);
#0092            adgASSERT(IsWindow(g_hwndModeless));
#0093            DlgTemplate_Free(pDlgTemplate);
#0094         }
#0095
#0096         // Toggle the text in the button
#0097         SetWindowText(hwndCtl, IsWindow(g_hwndModeless) ?
#0098            __TEXT("Destroy dynamic modeless dialog box") :
#0099            __TEXT("Create dynamic modeless dialog box"));
#0100         break;
#0101    }
#0102 }
#0103
#0104
#0105 ////////////////////////////////////////////////////////////////
#0106
#0107
#0108 BOOL WINAPI DlgDyn_DlgProc (HWND hwnd, UINT uMsg,
#0109    WPARAM wParam, LPARAM lParam) {
#0110
#0111    switch (uMsg) {
#0112
#0113       // Standard Window's messages
#0114       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, DlgDyn_OnInitDialog);
#0115       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    DlgDyn_OnCommand);
#0116    }
#0117    return(FALSE);              // We didn't process the message.
#0118 }
#0119
#0120
#0121 ////////////////////////////////////////////////////////////////
#0122
#0123
#0124 // This message filter hook function exists to enable the keyboard interface
#0125 // for the modeless dialog.
#0126 LRESULT WINAPI DlgDyn_MsgFilterProc (int nCode, WPARAM wParam, LPARAM lParam) {
#0127
#0128    // This variable prevents infinite recursion caused by the call to
#0129    // IsDialogMessage below (IsDialogMessage calls any installed message
#0130    // filter hooks as its first order of business).
#0131    static BOOL fRecurse = FALSE;
#0132    LRESULT lResult;
#0133
#0134    // If we are called recursively due to IsDialogMessage, we simply return.
#0135    if (fRecurse) {
#0136
```

```
#0137       // Returning FALSE tells the system that we want it to do its normal
#0138       // processing for this message.  Ie., let IsDialogMessage do its stuff.
#0139       // NOTE: We are not calling CallNextHookEx because we don't want the
#0140       // same message to go through the same hook chain twice (although this
#0141       // hook function will see it twice).
#0142       return(FALSE);
#0143    }
#0144
#0145    // Here we unhook our hook and then rehook ourselves to guarantee that our
#0146    // hook procedure is at the front of the hook chain. This is necessary
#0147    // because any hook procedure before us in the hook chain would see each
#0148    // message twice. By installing ourselves at the front of the chain we get
#0149    // the message first the second time it goes through the hook chain and
#0150    // can simply stop the message from continuing through the hook chain when
#0151    // we are recursing because of IsDialogMessage.
#0152    adgVERIFY(UnhookWindowsHookEx(g_hhookMsgFilter));
#0153    g_hhookMsgFilter = SetWindowsHookEx(WH_MSGFILTER, DlgDyn_MsgFilterProc,
#0154       NULL, GetCurrentThreadId());
#0155    adgASSERT(g_hhookMsgFilter != NULL);
#0156
#0157    // Call any other hooks in the chain.  lResult is TRUE if the next hook
#0158    // in the chain doesn't want any additional processing for the message.
#0159    lResult = CallNextHookEx(g_hhookMsgFilter, nCode, wParam, lParam);
#0160
#0161    // If the next hook in the chain wants to allow additional processing
#0162    if (!lResult) {
#0163
#0164       // Call IsDialogMessage, setting the static variable fRecurse to TRUE
#0165       // to avoid infinite recursion.  If IsDialogMessage returns TRUE, the
#0166       // message was processed and we DO NOT want normal processing for this
#0167       // message.
#0168       if (IsWindow(g_hwndModeless)) {
#0169          fRecurse = TRUE;
#0170          lResult = IsDialogMessage(g_hwndModeless, (PMSG) lParam);
#0171          fRecurse = FALSE;
#0172       }
#0173    }
#0174    return(lResult);
#0175 }
#0176
#0177
#0178 ////////////////////////////////////////////////////////////////
#0179
#0180
#0181 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0182    LPSTR lpszCmdLine, int nCmdShow) {
```

```
#0183
#0184    adgWARNIFUNICODEUNDERWIN95();
#0185
#0186    // Set message filter hook to enable keyboard interface processing
#0187    // for the dynamic modeless dialog box.
#0188    g_hhookMsgFilter = SetWindowsHookEx(WH_MSGFILTER, DlgDyn_MsgFilterProc,
#0189        NULL, GetCurrentThreadId());
#0190    adgASSERT(g_hhookMsgFilter != NULL);
#0191
#0192    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_DLGDYN),
#0193       NULL, DlgDyn_DlgProc));
#0194
#0195    // Clean up by unhooking our message filter hook
#0196    adgVERIFY(UnhookWindowsHookEx(g_hhookMsgFilter));
#0197    return(0);
#0198  }
#0199
#0200
#0201  ////////////////////////////////// End of File ////////////////////
```

程式列表 3.13　DynDlg.C

```
#0001  /******************************************************************
#0002  Module name: DynDlg.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Dynamic dialog box functions.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include "DlgTmplt.h"
#0014  #include "Resource.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////////
#0018
#0019
#0020  // Below is the table that describes the dialog box controls.
#0021  static struct {
#0022     short   x, y, cx, cy;
#0023     DWORD   dwStyle, dwExStyle, dwHelpId;
```

```
#0024    int    id;
#0025    LPCTSTR szClass, szText;
#0026    WORD   wExtraDataCount;
#0027    PBYTE  pbExtraData;
#0028 } DynDlgBoxData[] = {
#0029
#0030    { 4, 4, 32, 12, SS_LEFT, 0, 0, IDC_STATIC,
#0031       __TEXT("STATIC"), __TEXT("&Edit:"), 0, NULL },
#0032
#0033    { 40, 4, 64, 12, ES_LEFT | WS_BORDER | WS_TABSTOP | WS_GROUP, 0, 0,
#0034       IDC_EDITBOX, __TEXT("EDIT"), __TEXT(""), 0, NULL },
#0035
#0036    { 4, 18, 40,  8, SS_LEFT, 0, 0, IDC_STATIC,
#0037       __TEXT("STATIC"), __TEXT("&Listbox:"), 0, NULL },
#0038
#0039    { 4, 28, 76, 41, LBS_NOTIFY | LBS_SORT | LBS_STANDARD |
#0040       WS_BORDER | WS_VSCROLL | WS_TABSTOP | WS_GROUP, 0, 0,
#0041       IDC_LISTBOX, __TEXT("LISTBOX"), __TEXT(""), 0, NULL },
#0042
#0043    { 132,  4, 36, 12, BS_DEFPUSHBUTTON | WS_TABSTOP | WS_GROUP, 0, 0,
#0044       IDOK, __TEXT("BUTTON"), __TEXT("OK"), 0, NULL },
#0045
#0046    { 176,  4, 36, 12, BS_PUSHBUTTON | WS_TABSTOP | WS_GROUP, 0, 0,
#0047       IDCANCEL, __TEXT("BUTTON"), __TEXT("Cancel"), 0, NULL },
#0048
#0049    { 88, 24, 56, 44, BS_GROUPBOX | WS_GROUP, 0, 0, IDC_STATIC,
#0050       __TEXT("BUTTON"), __TEXT("Group box"), 0, NULL },
#0051
#0052    { 92, 36, 48, 12, BS_AUTORADIOBUTTON | WS_TABSTOP | WS_GROUP, 0, 0,
#0053       IDC_RADIO1, __TEXT("BUTTON"), __TEXT("Radio &1"), 0, NULL },
#0054
#0055    { 92, 48, 48, 12, BS_AUTORADIOBUTTON | WS_TABSTOP, 0, 0,
#0056       IDC_RADIO2, __TEXT("BUTTON"), __TEXT("Radio &2"), 0, NULL },
#0057
#0058    { 152, 32, 60, 12, BS_AUTOCHECKBOX | WS_TABSTOP | WS_GROUP, 0, 0,
#0059       IDC_CHECKBOX, __TEXT("BUTTON"), __TEXT("C&heck box"), 0, NULL },
#0060
#0061    // End of list marker (szClass must be null)
#0062    { 0, 0, 0, 0, 0, 0, 0, 0, NULL, NULL, 0, NULL }
#0063 };
#0064
#0065
#0066 ////////////////////////////////////////////////////////////
#0067
#0068
#0069 PDLGTEMPLATEEX BuildDynamicDlgBox (BOOL fModeless) {
```

```
#0070
#0071    PDLGTEMPLATEEX pDlgTemplate;
#0072    int x;
#0073    BOOL fCtrlAddedOK;
#0074
#0075    // Create the dynamic dialog box header information.
#0076    if (fModeless) {
#0077      pDlgTemplate = DlgTemplate_Create(
#0078         WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | WS_VISIBLE |
#0079         WS_POPUP | DS_SETFONT, 0, 0, 6, 26, 216, 72, __TEXT(""), __TEXT(""),
#0080         __TEXT("Dynamic Modeless Demo"),
#0081         12, FW_NORMAL, TRUE, __TEXT("Courier New"));
#0082    } else {
#0083       pDlgTemplate = DlgTemplate_Create(
#0084         WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | WS_VISIBLE |
#0085         WS_POPUP, 0, 0, 6, 26, 216, 72, __TEXT(""), __TEXT(""),
#0086         __TEXT("Dynamic Modal Demo"),
#0087         0, 0, 0, NULL);
#0088    }
#0089
#0090    if (pDlgTemplate == NULL)
#0091       return(NULL);
#0092
#0093    // Add each of the controls in the DynamicDlgBoxData array.
#0094    for (x = 0; DynDlgBoxData[x].szClass != NULL; x++) {
#0095
#0096       // Do not add an OK button for a modeless dialog box.
#0097       if (fModeless) {
#0098          if (DynDlgBoxData[x].id == IDOK)
#0099             continue;
#0100       }
#0101
#0102       fCtrlAddedOK = DlgTemplate_AddControl(&pDlgTemplate,
#0103          DynDlgBoxData[x].x,
#0104          DynDlgBoxData[x].y,
#0105          DynDlgBoxData[x].cx,
#0106          DynDlgBoxData[x].cy,
#0107          DynDlgBoxData[x].id,
#0108          DynDlgBoxData[x].dwHelpId,
#0109          DynDlgBoxData[x].dwStyle | WS_VISIBLE,
#0110          DynDlgBoxData[x].dwExStyle,
#0111          DynDlgBoxData[x].szClass,
#0112          DynDlgBoxData[x].szText,
#0113          DynDlgBoxData[x].wExtraDataCount,
#0114          DynDlgBoxData[x].pbExtraData);
#0115
```

```
#0116       if (!fCtrlAddedOK)
#0117          break;
#0118    }
#0119
#0120    // If insufficient memory, free what we have and return NULL to caller.
#0121    if (!fCtrlAddedOK) {
#0122       DlgTemplate_Free(pDlgTemplate);
#0123       pDlgTemplate = NULL;
#0124    }
#0125
#0126    // Return the address of the dynamic dialog box information.
#0127    return(pDlgTemplate);
#0128 }
#0129
#0130
#0131 ////////////////////////////////////////////////////////////////
#0132
#0133
#0134 BOOL DynDlg_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0135
#0136    return(TRUE);
#0137 }
#0138
#0139
#0140 ////////////////////////////////////////////////////////////////
#0141
#0142 void DynDlg_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0143
#0144    switch (id) {
#0145
#0146       case IDOK:
#0147       case IDCANCEL:            // Allows dialog box to close
#0148
#0149          // The processing in here to close the dialog box differs depending on
#0150          // whether the dialog box is modal or modeless.
#0151
#0152          // First we must determine if the dialog box is modal or modeless.
#0153          // If there is an OK button, the dialog box is modal.
#0154          if (IsWindow(GetDlgItem(hwnd, IDOK))) {
#0155             // The dialog box is modal
#0156             EndDialog(hwnd, id); // Close the dialog box
#0157          } else {
#0158             // The dialog box is modeless
#0159             if (id == IDCANCEL) {
#0160                // Don't destroy the dialog box if the user pressed Enter.
#0161
```

```
#0162              // Simulate the user clicking the "Destroy dynamic modeless
#0163              // dialog box" button in the parent window.
#0164              FORWARD_WM_COMMAND(GetParent(hwnd), IDC_MODELESS,
#0165                 GetDlgItem(GetParent(hwnd), IDC_MODELESS),
#0166                 BN_CLICKED, PostMessage);
#0167           }
#0168        }
#0169        break;
#0170     }
#0171  }
#0172
#0173
#0174  ////////////////////////////////////////////////////////////////
#0175
#0176
#0177  BOOL WINAPI DynDlg_DlgProc (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
#0178
#0179     switch (uMsg) {
#0180
#0181        // Standard Window's messages
#0182        adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, DynDlg_OnInitDialog);
#0183        adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    DynDlg_OnCommand);
#0184     }
#0185     return(FALSE);              // We didn't process the message.
#0186  }
#0187
#0188
#0189  ////////////////////////// End of File ///////////////////////////////
```

程式列表 3.14　DlgTmplt.C

```
#0001  /******************************************************************
#0002  Module name: DlgTmplt.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Dynamic Dialog Box template toolkit functions.
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include <malloc.h>
#0014  #include "DlgTmplt.h"
```

```
#0015
#0016
#0017 ////////////////////////////////////////////////////////////
#0018
#0019
#0020 #define NEXTDWORDBOUNDARY(p) ((PBYTE) ((DWORD) ((3 + (PBYTE) (p))) & ~3))
#0021 #define LENINWORDS(ByteLen)  ((ByteLen + sizeof(WORD)  - 1) & ~1)
#0022 #define LENINDWORDS(ByteLen) ((ByteLen + sizeof(DWORD) - 1) & ~3)
#0023
#0024
#0025 ////////////////////////////////////////////////////////////
#0026
#0027
#0028 // Macro to calculate number of bytes needed for a (unicode) string,
#0029 // including the zero terminating character.
#0030 #define BYTESFORSTRING(sz) (sizeof(WCHAR) * (1 + wcslen(sz)))
#0031
#0032
#0033 ////////////////////////////////////////////////////////////
#0034
#0035
#0036 BOOL ANSIToUnicode (PCSTR szStrA, PWSTR *pszStrW) {
#0037
#0038    int nLenOfWideCharStr;
#0039    BOOL fOK = FALSE;            // Assume failure
#0040
#0041    *pszStrW = NULL;             // Assume failure
#0042    if (szStrA == NULL) {
#0043
#0044       // It is OK to have a NULL string.
#0045       return(TRUE);
#0046    }
#0047
#0048    // Get the number of bytes needed for the wide version of the string.
#0049    nLenOfWideCharStr = MultiByteToWideChar(CP_ACP, 0, szStrA, -1, NULL, 0);
#0050
#0051    // Allocate memory to accomodate the size of the wide character string.
#0052    *pszStrW = malloc(nLenOfWideCharStr * sizeof(WCHAR));
#0053
#0054    if (*pszStrW != NULL) {
#0055
#0056       // Convert the multi-byte string to a wide-character string.
#0057       MultiByteToWideChar(CP_ACP, 0, szStrA, -1, *pszStrW, nLenOfWideCharStr);
#0058       fOK = TRUE;
#0059    }
#0060    return(fOK);
```

```
#0061  }
#0062
#0063
#0064  ////////////////////////////////////////////////////////////
#0065
#0066
#0067  PDLGTEMPLATEEX DlgTemplate_CreateW (
#0068     DWORD  dwStyle,
#0069     DWORD  dwExStyle,
#0070     DWORD  dwHelpId,
#0071     short  x,  short y,          // In dialog-box units
#0072     short  cx, short cy,         // In dialog-box units
#0073     PCWSTR szMenuName,           // "" if no menu
#0074     PCWSTR szClassName,          // "" if standard dialog box class
#0075     PCWSTR szCaptionText,
#0076     short  nPointSize,           // Only used if DS_SETFONT specified
#0077     short  nWeight,              // Only used if DS_SETFONT specified
#0078     short  fItalic,              // Only used if DS_SETFONT specified
#0079     PCWSTR szTypeFace) {         // Only used if DS_SETFONT specified
#0080
#0081     int          nBytesForNewTemplate;
#0082     int          nMenuNameLen, nClassNameLen, nCaptionTextLen, nTypeFaceLen;
#0083     PBYTE        pbDlgTemplate, pbDlgTypeFace;
#0084     PDLGTEMPLATEEX pDlgTemplate;
#0085     PFONTINFOEX   pFontInfo;
#0086
#0087     // Calculate number of bytes required by following fields:
#0088     nMenuNameLen    = BYTESFORSTRING(szMenuName);
#0089     nClassNameLen   = BYTESFORSTRING(szClassName);
#0090     nCaptionTextLen = BYTESFORSTRING(szCaptionText);
#0091
#0092     // Block must be large enough to contain the following:
#0093     nBytesForNewTemplate =
#0094        adgMEMBEROFFSET(DLGTEMPLATEEX, bStartOfStrings) +
#0095        nMenuNameLen +         // # bytes: menu name.
#0096        nClassNameLen +        // # bytes: dialog class name.
#0097        nCaptionTextLen;       // # bytes: dialog box caption.
#0098
#0099     if (dwStyle & DS_SETFONT) {
#0100
#0101        // Dialog box uses font other than System font.
#0102
#0103        // Calculate # of bytes required for typeface name.
#0104        nTypeFaceLen = BYTESFORSTRING(szTypeFace);
#0105
#0106        // Block must be large enough to include font information.
```

```
#0107       nBytesForNewTemplate +=
#0108          adgMEMBEROFFSET(FONTINFOEX, bStartOfStrings) +
#0109          nTypeFaceLen;       // # bytes for font typeface name.
#0110
#0111    } else {
#0112
#0113       // Dialog box uses the System font.
#0114       nTypeFaceLen = 0;
#0115
#0116       // Block length does not change.
#0117    }
#0118
#0119    // Allocate an even number of DWORDs for alignment.
#0120    nBytesForNewTemplate = LENINDWORDS(nBytesForNewTemplate);
#0121
#0122    // Allocate block of memory for dialog template.
#0123    pDlgTemplate = (PDLGTEMPLATEEX) malloc(nBytesForNewTemplate);
#0124    if (pDlgTemplate == NULL)
#0125       return(NULL);
#0126
#0127    // Set the members of the DLGTEMPLATE structure.
#0128    pDlgTemplate->wDlgVer    = 1;        // Always 1
#0129    pDlgTemplate->wSignature  = 0xFFFF;    // Always 0xFFFF
#0130    pDlgTemplate->dwHelpID    = dwHelpId;
#0131    pDlgTemplate->dwStyle     = dwStyle;
#0132    pDlgTemplate->dwExStyle   = dwExStyle;
#0133    pDlgTemplate->cDlgItems   = 0;        // AddControl increments this
#0134    pDlgTemplate->x          = x;
#0135    pDlgTemplate->y          = y;
#0136    pDlgTemplate->cx         = cx;
#0137    pDlgTemplate->cy         = cy;
#0138
#0139
#0140    // pbDlgTemplate points to start of variable part of DLGTEMPLATEEX.
#0141    pbDlgTemplate = (PBYTE) pDlgTemplate +
#0142       adgMEMBEROFFSET(DLGTEMPLATEEX, bStartOfStrings);
#0143
#0144    // Append the menu name, class name, and caption text to block.
#0145    CopyMemory(pbDlgTemplate, szMenuName, nMenuNameLen);
#0146    pbDlgTemplate += nMenuNameLen;
#0147    CopyMemory(pbDlgTemplate, szClassName, nClassNameLen);
#0148    pbDlgTemplate += nClassNameLen;
#0149    CopyMemory(pbDlgTemplate, szCaptionText, nCaptionTextLen);
#0150    pbDlgTemplate += nCaptionTextLen;
#0151
#0152    if (dwStyle & DS_SETFONT) {
```

```
#0153
#0154       // Dialog box uses font other than system font.
#0155
#0156       // pFontInfo points to start of FONTINFOEX structure.
#0157       pFontInfo = (PFONTINFOEX) pbDlgTemplate;
#0158
#0159       // Set the members of the FONTINFO structure.
#0160       pFontInfo->nPointSize = nPointSize;
#0161       pFontInfo->nWeight    = nWeight;
#0162       pFontInfo->fItalic    = fItalic;
#0163
#0164       // szTypeFace points to start of the variable part of FONTINFOEX.
#0165       pbDlgTypeFace = (PBYTE) pFontInfo +
#0166          adgMEMBEROFFSET(FONTINFOEX, bStartOfStrings);
#0167
#0168       // Append the typeface name to the block.
#0169       CopyMemory(pbDlgTypeFace, szTypeFace, nTypeFaceLen);
#0170    }
#0171
#0172    return(pDlgTemplate);
#0173 }
#0174
#0175
#0176 //////////////////////////////////////////////////////////////
#0177
#0178
#0179 PDLGTEMPLATEEX DlgTemplate_CreateA (
#0180    DWORD  dwStyle,
#0181    DWORD  dwExStyle,
#0182    DWORD  dwHelpId,
#0183    short   x,  short y,          // In dialog-box units
#0184    short   cx, short cy,         // In dialog-box units
#0185    PCSTR  szMenuName,            // "" if no menu
#0186    PCSTR  szClassName,           // "" if standard dialog box class
#0187    PCSTR  szCaptionText,
#0188    short  nPointSize,            // Only used if DS_SETFONT specified
#0189    short  nWeight,               // Only used if DS_SETFONT specified
#0190    short  fItalic,               // Only used if DS_SETFONT specified
#0191    PCSTR  szTypeFace) {          // Only used if DS_SETFONT specified
#0192
#0193    PDLGTEMPLATEEX pDlgTemplate = NULL;
#0194    PWSTR szMenuNameW = NULL, szClassNameW = NULL;
#0195    PWSTR szCaptionTextW = NULL, szTypeFaceW = NULL;
#0196
#0197    // Allocate temporary Unicode buffers and convert ANSI strings to Unicode.
#0198    if (ANSIToUnicode(szMenuName,    &szMenuNameW)    &&
```

```
#0199       ANSIToUnicode(szClassName,   &szClassNameW)   &&
#0200       ANSIToUnicode(szCaptionText, &szCaptionTextW) &&
#0201       ANSIToUnicode(szTypeFace,    &szTypeFaceW)) {
#0202
#0203      pDlgTemplate = DlgTemplate_CreateW(dwStyle, dwExStyle,
#0204         dwHelpId, x, y, cx, cy, szMenuNameW, szClassNameW,
#0205         szCaptionTextW, nPointSize, nWeight, fItalic, szTypeFaceW);
#0206    }
#0207
#0208    // Destroy temporary Unicode buffers.
#0209    // NOTE: ANSI C states that it is OK to pass NULL to free.
#0210    free(szTypeFaceW);
#0211    free(szCaptionTextW);
#0212    free(szClassNameW);
#0213    free(szMenuNameW);
#0214
#0215    return(pDlgTemplate);
#0216  }
#0217
#0218
#0219  ////////////////////////////////////////////////////////////////
#0220
#0221
#0222  BOOL DlgTemplate_AddControlW (
#0223    PDLGTEMPLATEEX* ppDlgTemplate, // Returned from Create or AddControl
#0224    short  x,  short y,          // In dialog-box units
#0225    short  cx, short cy,         // In dialog-box units
#0226    DWORD  id,
#0227    DWORD  dwHelpId,
#0228    DWORD  dwStyle,             // WS_CHILD is automatically added
#0229    DWORD  dwExStyle,
#0230    PCWSTR szClass,
#0231    PCWSTR szText,
#0232    WORD   wExtraDataCount,
#0233    PBYTE  pbExtraData) {        // Number of additional data bytes
#0234
#0235    PDLGTEMPLATEEX pDlgTemplateNew; // If the memory block expands.
#0236    int nBytesForNewControl;
#0237    int nBytesBeforeAddingControl;
#0238    PDLGITEMTEMPLATEEX pDlgItemTemplate;
#0239    PBYTE pbDlgItemTemplate;
#0240
#0241    // Calculate number of bytes required by following fields:
#0242    int nClassLen = BYTESFORSTRING(szClass);
#0243    int nTextLen  = BYTESFORSTRING(szText);
#0244
```

```
#0245    // Block must be increased by to contain the following:
#0246    nBytesForNewControl =
#0247       adgMEMBEROFFSET(DLGITEMTEMPLATEEX, bStartOfStrings) +
#0248       nClassLen +              // # bytes for control class
#0249       nTextLen +               // # bytes for control text
#0250       sizeof(WORD);             // 1 byte  for size in bytes of CreateParams
#0251                               // data
#0252
#0253    // Alignment padding to CreateParams data
#0254    nBytesForNewControl = LENINDWORDS(nBytesForNewControl);
#0255    nBytesForNewControl += wExtraDataCount; // # bytes for CreateParams data
#0256
#0257    // Alignment padding to next DLGITEMTEMPLATEEX structure
#0258    nBytesForNewControl = LENINDWORDS(nBytesForNewControl);
#0259
#0260    // Increase the size of the memory block to include the new dialog item.
#0261    nBytesBeforeAddingControl = _msize(*ppDlgTemplate);
#0262    pDlgTemplateNew = realloc(*ppDlgTemplate,
#0263       nBytesBeforeAddingControl + nBytesForNewControl);
#0264    if (pDlgTemplateNew == NULL)
#0265       return(FALSE);
#0266
#0267    // Tell the caller the new address of the dialog template.
#0268    *ppDlgTemplate = pDlgTemplateNew;
#0269
#0270    // Increment the number of controls in the template.
#0271    pDlgTemplateNew->cDlgItems++;
#0272
#0273    // pDlgItemTemplate points to the start of the DLGITEMTEMPLATE being added.
#0274    pDlgItemTemplate = (PDLGITEMTEMPLATEEX)
#0275       (((PBYTE) pDlgTemplateNew) + nBytesBeforeAddingControl);
#0276
#0277    // Set the members of the newly added DLGITEMTEMPLATE structure.
#0278    pDlgItemTemplate->x        = x;
#0279    pDlgItemTemplate->y        = y;
#0280    pDlgItemTemplate->cx       = cx;
#0281    pDlgItemTemplate->cy       = cy;
#0282    pDlgItemTemplate->id       = id;
#0283    pDlgItemTemplate->dwStyle   = dwStyle | WS_CHILD;
#0284    pDlgItemTemplate->dwExStyle = dwExStyle;
#0285    pDlgItemTemplate->dwHelpID  = dwHelpId;
#0286
#0287    // pbDlgTemplate points to the start of the variable part of the
#0288    // DLGITEMTEMPLATEEX.
#0289    pbDlgItemTemplate = (PBYTE) pDlgItemTemplate +
#0290       adgMEMBEROFFSET(DLGITEMTEMPLATEEX, bStartOfStrings);
```

```
#0291
#0292    // Append the strings for the control's class name and caption to the block.
#0293    CopyMemory(pbDlgItemTemplate, szClass, nClassLen);
#0294    pbDlgItemTemplate += nClassLen;
#0295    CopyMemory(pbDlgItemTemplate, szText, nTextLen);
#0296    pbDlgItemTemplate += nTextLen;
#0297
#0298    // Append the control's extra data byte information.
#0299    * (PWORD) pbDlgItemTemplate = wExtraDataCount;
#0300    pbDlgItemTemplate += sizeof(WORD);
#0301
#0302    // Append the control's CreateParams data bytes.
#0303    CopyMemory(pbDlgItemTemplate, pbExtraData, wExtraDataCount);
#0304
#0305    return(TRUE);
#0306  }
#0307
#0308
#0309  ////////////////////////////////////////////////////////////////
#0310
#0311
#0312  BOOL DlgTemplate_AddControlA (
#0313    PDLGTEMPLATEEX* ppDlgTemplate, // Returned from Create or AddControl
#0314    short  x,  short y,          // In dialog-box units
#0315    short  cx, short cy,         // In dialog-box units
#0316    DWORD  id,
#0317    DWORD  dwHelpId,
#0318    DWORD  dwStyle,             // WS_CHILD is automatically added
#0319    DWORD  dwExStyle,
#0320    PCSTR  szClass,
#0321    PCSTR  szText,
#0322    WORD   wExtraDataCount,      // Number of additional data bytes
#0323    PBYTE  pbExtraData) {        // Passed via CREATESTRUCT's pCreateParams
#0324
#0325    BOOL fOK = FALSE;            // Assume failure
#0326    PWSTR szClassW = NULL;
#0327    PWSTR szTextW = NULL;
#0328
#0329    // Allocate temporary Unicode buffers and convert ANSI strings to Unicode.
#0330    if (ANSIToUnicode(szClass, &szClassW) &&
#0331        ANSIToUnicode(szText, &szTextW)) {
#0332
#0333       fOK = DlgTemplate_AddControlW(ppDlgTemplate, x, y, cx, cy, id, dwHelpId,
#0334          dwStyle, dwExStyle, szClassW, szTextW, wExtraDataCount, pbExtraData);
#0335    }
#0336
```

```
#0337    // Destroy temporary Unicode buffers.
#0338    // NOTE: ANSI C states that it is OK to pass NULL to free.
#0339    free(szTextW);
#0340    free(szClassW);
#0341
#0342    return(fOK);
#0343  }
#0344
#0345
#0346  ////////////////////////////////////////////////////////////////
#0347
#0348
#0349  BOOL DlgTemplate_Free (PDLGTEMPLATEEX pDlgTemplate) {
#0350
#0351    free((PVOID) pDlgTemplate);
#0352    return(TRUE);
#0353  }
#0354
#0355
#0356  ////////////////////////// End of File ///////////////////////////
```

程式列表 3.15　DlgTmplt.H

```
#0001  /******************************************************************
#0002  Module name: DlgTmplt.h
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Dynamic Dialog Box template toolkit functions.
#0006  ******************************************************************/
#0007
#0008  typedef struct { // dlttex
#0009    WORD    wDlgVer;       // Always 1
#0010    WORD    wSignature;    // Always 0xFFFF
#0011    DWORD   dwHelpID;
#0012    DWORD   dwExStyle;     // Such as WS_EX_TOPMOST
#0013    DWORD   dwStyle;       // Such as WS_CAPTION
#0014    WORD    cDlgItems;
#0015    short   x;         // In pixels
#0016    short   y;         // In pixels
#0017    short   cx;        // In dialog box units
#0018    short   cy;        // In dialog box units
#0019    BYTE    bStartOfStrings;
#0020    // Zero-terminated Unicode string for the menu name;
#0021    // Zero-terminated Unicode string for the class name;
#0022    // Zero-terminated Unicode string for the window title;
```

```
#0023 } DLGTEMPLATEEX, *PDLGTEMPLATEEX;
#0024
#0025
#0026 typedef struct {
#0027    short   nPointSize;
#0028    short   nWeight;    // Such as FW_NORMAL
#0029    short   fItalic;    // TRUE or FALSE
#0030    BYTE    bStartOfStrings;
#0031    // Zero-terminated Unicode string for the font name;
#0032 } FONTINFOEX, *PFONTINFOEX;
#0033
#0034
#0035 typedef struct {
#0036    DWORD   dwHelpID;
#0037    DWORD   dwExStyle;   // Such as WS_EX_CONTROLPARENT
#0038    DWORD   dwStyle;    // Such as WS_TABSTOP
#0039    short   x;       // In dialog box units
#0040    short   y;       // In dialog box units
#0041    short   cx;      // In dialog box units
#0042    short   cy;      // In dialog box units
#0043    DWORD   id;
#0044    BYTE    bStartOfStrings;
#0045    // Zero-terminated Unicode string for the class name;
#0046    // Zero-terminated Unicode string for the control title;
#0047    // WORD    wExtraCount; // Usually 0
#0048    // wExtraCount bytes of raw data;
#0049  } DLGITEMTEMPLATEEX, *PDLGITEMTEMPLATEEX;
#0050
#0051 PDLGTEMPLATEEX DlgTemplate_CreateW (
#0052    DWORD  dwStyle,
#0053    DWORD  dwExStyle,
#0054    DWORD  dwHelpId,
#0055    short  x,  short y,          // In dialog-box units
#0056    short  cx, short cy,         // In dialog-box units
#0057    PCWSTR szMenuName,           // "" if no menu
#0058    PCWSTR szClassName,          // "" if standard dialog box class
#0059    PCWSTR szCaptionText,
#0060    short  nPointSize,           // Only used if DS_SETFONT specified
#0061    short  nWeight,              // Only used if DS_SETFONT specified
#0062    short  fItalic,              // Only used if DS_SETFONT specified
#0063    PCWSTR szTypeFace);          // Only used if DS_SETFONT specified
#0064
#0065
#0066 PDLGTEMPLATEEX DlgTemplate_CreateA (
#0067    DWORD  dwStyle,
#0068    DWORD  dwExStyle,
```

```
#0069    DWORD  dwHelpId,
#0070    short  x,  short y,          // In dialog-box units
#0071    short  cx, short cy,         // In dialog-box units
#0072    PCSTR  szMenuName,           // "" if no menu
#0073    PCSTR  szClassName,          // "" if standard dialog box class
#0074    PCSTR  szCaptionText,
#0075    short  nPointSize,           // Only used if DS_SETFONT specified
#0076    short  nWeight,              // Only used if DS_SETFONT specified
#0077    short  fItalic,              // Only used if DS_SETFONT specified
#0078    PCSTR  szTypeFace);          // Only used if DS_SETFONT specified
#0079
#0080  #ifdef UNICODE
#0081  #define DlgTemplate_Create DlgTemplate_CreateW
#0082  #else
#0083  #define DlgTemplate_Create DlgTemplate_CreateA
#0084  #endif // !UNICODE
#0085
#0086
#0087  //////////////////////////////////////////////////////////////
#0088
#0089
#0090  BOOL DlgTemplate_AddControlW (
#0091    PDLGTEMPLATEEX* ppDlgTemplate, // Returned from Create or AddControl
#0092    short  x,  short y,          // In dialog-box units
#0093    short  cx, short cy,         // In dialog-box units
#0094    DWORD  id,
#0095    DWORD  dwHelpId,
#0096    DWORD  dwStyle,              // WS_CHILD is automatically added
#0097    DWORD  dwExStyle,
#0098    PCWSTR szClass,
#0099    PCWSTR szText,
#0100    WORD   wExtraDataCount,      // Number of additional data bytes
#0101    PBYTE  pbExtraData);         // Passed via CREATESTRUCT's pCreateParams
#0102
#0103  BOOL DlgTemplate_AddControlA (
#0104    PDLGTEMPLATEEX* ppDlgTemplate, // Returned from Create or AddControl
#0105    short  x,  short y,          // In dialog-box units
#0106    short  cx, short cy,         // In dialog-box units
#0107    DWORD  id,
#0108    DWORD  dwHelpId,
#0109    DWORD  dwStyle,              // WS_CHILD is automatically added
#0110    DWORD  dwExStyle,
#0111    PCSTR  szClass,
#0112    PCSTR  szText,
#0113    WORD   wExtraDataCount,      // Number of additional data bytes
#0114    PBYTE  pbExtraData);         // Passed via CREATESTRUCT's pCreateParams
```

```
#0115
#0116  #ifdef UNICODE
#0117  #define DlgTemplate_AddControl DlgTemplate_AddControlW
#0118  #else
#0119  #define DlgTemplate_AddControl DlgTemplate_AddControlA
#0120  #endif // !UNICODE
#0121
#0122
#0123  //////////////////////////////////////////////////////////////////
#0124
#0125
#0126  BOOL DlgTemplate_Free (PVOID pvTemplate);
#0127
#0128
#0129  ////////////////////////// End of File ///////////////////////////////
```

程式列表 3.16　DlgDyn.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  //////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  //////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  //////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
```

```
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  ////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  ////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_DLGDYN DIALOG DISCARDABLE  -32768, 5, 156, 42
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Dynamic Dialogs"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      DEFPUSHBUTTON   "Create dynamic modal dialog box...",IDC_MODAL,4,4,148,
#0054                  14
#0055      PUSHBUTTON      "Create dynamic modeless dialog box",IDC_MODELESS,4,24,
#0056                  148,14
#0057  END
#0058
#0059
#0060  ////////////////////////////////////////////////////////////
#0061  //
#0062  // Icon
#0063  //
#0064
#0065  IDI_DLGDYN            ICON    DISCARDABLE     "DlgDyn.ico"
#0066
#0067  #ifndef APSTUDIO_INVOKED
#0068  ////////////////////////////////////////////////////////////
#0069  //
#0070  // Generated from the TEXTINCLUDE 3 resource.
#0071  //
#0072
#0073
```

```
#0074  //////////////////////////////////////////////////////////////
#0075  #endif    // not APSTUDIO_INVOKED
```

程式列表 3.17　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by DlgDyn.rc
#0004  //
#0005  #define IDD_DLGDYN                  101
#0006  #define IDC_EDITBOX                 101
#0007  #define IDI_DLGDYN                  102
#0008  #define IDC_LISTBOX                 102
#0009  #define IDC_RADIO1                  103
#0010  #define IDC_RADIO2                  104
#0011  #define IDC_CHECKBOX                105
#0012  #define IDC_MODAL                   1000
#0013  #define IDC_MODELESS                1001
#0014  #define IDC_STATIC                  -1
#0015
#0016  // Next default values for new objects
#0017  //
#0018  #ifdef APSTUDIO_INVOKED
#0019  #ifndef APSTUDIO_READONLY_SYMBOLS
#0020  #define _APS_NEXT_RESOURCE_VALUE      103
#0021  #define _APS_NEXT_COMMAND_VALUE       40001
#0022  #define _APS_NEXT_CONTROL_VALUE       1001
#0023  #define _APS_NEXT_SYMED_VALUE         106
#0024  #endif
#0025  #endif
```

對話盒的 “Layout” 技術

使用者常常會發現，他們所使用的對話盒太小，以至於不符合需求。某些情況下，對話盒的控制元件甚至無法完全顯示它們的文字。讓人奇怪的是，爲什麼【開啓舊檔】對話盒不能讓使用者自由改變其大小，以便能夠一次看到更多檔案和資料夾？當使用者將對話盒調整得更大之後，其上的所有控制元件相對地也必需被重新定位與調整大小，這樣做會使整個對話盒看起來更有效率嗎？

這種情況可能發生！在 Windows 95 桌面上，對著【我的電腦（My Computer）】按下滑鼠右鍵，出現一個突冒式（pop-up）選單，好，請選擇其中的尋找（Find...）選項，如圖 **3.11**。於是出現如圖 **3.12** 所示的對話盒。你可以把【Find : All Files】對話盒向右拉長，得到如圖 **3.13** 的畫面。

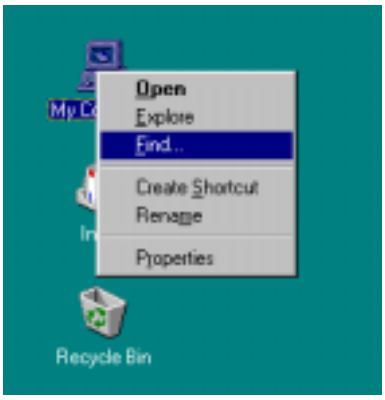


圖 3.11　選擇“尋找”（Find）選項

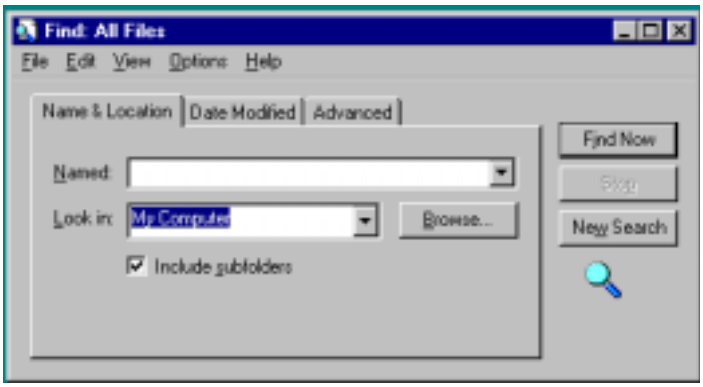


圖 3.12“尋找：所有檔案”對話盒

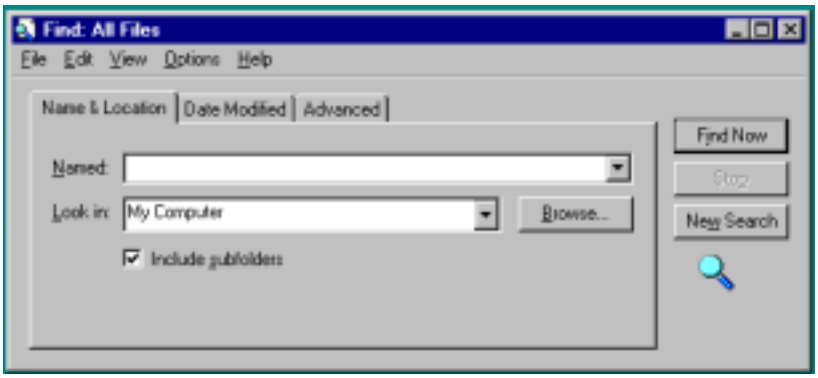


圖 3.13　調整大小後的“尋找：所有檔案”對話盒

能讓使用者改變對話盒的大小，程式將顯得很有彈性。如果對話盒內含的控制元件是 comboboxes、listboxes、edit controls，就比較容易處理視窗拉大或縮小的問題。

Windows 提供極少的函式來協助 Win32 程式員重新組織對話盒。當然，你可以總是爲你的對話盒設立 WS_THICKFRAME 風格，使它能夠被任意調整大小。但不幸地，你必須爲了這樣的對話盒，在其對應的視窗函式的 OnSize 訊息處理常式中寫上好一大段碼。只要對話盒的大小改變，這一段碼就必須重新安排其所有控制元件的位置和大小。寫這類程式碼需要很多時間，而其結果常常沒有彈性、難以維護並容易出錯。這裡我將介紹一種所謂的 Layout 技術，它提供一種資料驅動（data-driven）處理方式，藉由一小組規則的設定，而不是藉由程式碼的撰寫，使你能夠重新安排你的對話盒控制元件。

使用 “Layout” 技術

欲使用 Layout 技術，請讓你的對話盒的 OnSize 訊息處理常式將你的對話盒視窗代碼以及一串規則，當做參數傳遞給 Layout_ComputeLayout。後者會根據你所指定的規則排列你的對話盒子視窗。如果你設定的規則有問題，Layout 會發出一個 “assertion”，顯示錯誤狀況並離開程式。下面是使用 Layout 技術來完成 OnSize 訊息處理常式的一個簡單例子：

```
void SomeClass_OnSize (HWND hwnd, UINT state, int cx, int cy) {

   static RULE rules[] = {

      // Action    Act-on                      Relative-to              Offset
      // ------    ------                      -----------              ------

      {  lMOVE,    lTOP   (IDC_B2),            lBOTTOM   (IDC_B1),      +8  },
      {  lMOVE,    lRIGHT (IDC_B1),            lRIGHT    (lPARENT),     -8  },
      {  lMOVE,    lLEFT  (IDC_B2),            lLEFT     (IDC_B1),      +0  },
      {  lVCENTER, lGROUP (IDC_B1, IDC_B2),    lCHILD    (IPARENT),     +0  },
      {  lEND                                                 }

   };

   // Place the dialog's child controls according to the rules. If there is
   // a problem with the rules, Layout_ComputeLayout will assert.
   Layout_ComputeLayout(hwnd, rules);
}
```

當 SomeClass_OnSize 最初被呼叫時，Layout_ComputeLayout 會放置 IDC_B1 和 IDC_B2 控制元件至對話盒中，使對話盒看起如圖 **3.14**。然後，如果你垂直拉動對話盒，SomeClass_OnSize 會調整控制元件的位置，如圖 **3.15**。如果你水平拉動對話盒，會得到如圖 **3.16** 的結果。如果你從右下角拉大對話盒，得到的結果如圖 **3.17**。

圖 3.14　對話盒原本的樣子

圖 3.15　對話盒垂直方向拉長

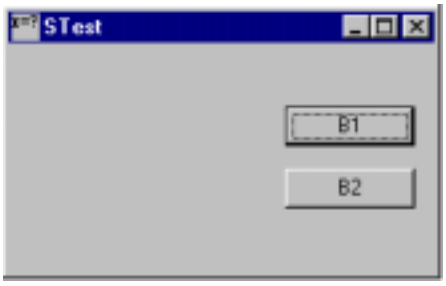


圖 3.16　對話盒水平方向拉長

圖 3.17　對話盒右下角拉長

欲得到這樣的結果，OnSize 處理常式必須傳遞一個陣列給 Layout_ComputeLayout。這個陣列中的每一項規則會透過一些巨集填入一個 RULE 結構中（定義在 Layout.H 表頭檔中）。每一筆規則的第一個欄位表示一個動作（action），可以是 lSTRETCH 或 lMOVE 或 lHCENTER 或 lVCENTER。最後一筆規則應該有一個 lEND 動作，用以標示結束。陣列中規則的排列順序並不會影響執行結果。

當你定義了一個規則，side 可以是控制元件的左緣、上緣、右緣或下緣；metric 要不是控制元件的一個邊，就是控制元件的寬度或高度；而 part 指的如果不是一個 metric，就是一群控制元件。每一筆規則都有一個 part 表示所影響的控制元件，另有一個 part 表示動作是相對於哪些控制元件。舉個例子，先前的程式碼中共有五筆規則，第一筆規則表示影響 IDC_B2 控制元件（第二個按鈕）的上緣，而相對於 IDC_B1 控制元件（第一個按鈕）的下緣，動作是 Lmove。執行結果是：移動 IDC_B2 控制元件至距離 IDC_B1 控制元件下緣 8 個「垂直對話盒單位」的位置上。這 8 個對話盒單位是根據每一規則之最後一個欄位的值 -- 此值用來自由調整兩個控制元件之間的距離。

所有能夠用來填寫 act-on 和 relative-to 欄位的巨集，都列於表 **3.5** 中。

表 3.5 Layout 所能使用的巨集

部份	說明
lLEFT(idc) lTOP(idc) lRIGHT(idc) lBOTTOM(idc)	指定控制元件 idc 的一個邊
lWIDTH(idc) lHEIGHT(idc)	指定此控制元件 idc 的寬度或高度
lWIDTHOF(idc, percent) lHEIGHTOF(idc, percent)	指定控制元件 idc 的寬度或高度的百分比。舉個例，lWIDTHOF(IDC_B1,33) 指定 IDC_B1 控制元件的寬度的 33%。
lCHILD(idc)	指定控制元件 idc。
lGROUP(idcFirst, idcLast)	指定 id 值落於 idcFirst 至 idcLast 之間的所有控制元件成爲一個群組（group）。

請注意前一個範例中的第二筆規則，其 relative-to 欄位表示在一個控制元件的右側，這個控制元件是 lPARENT。Lparent 表示一個特殊的值，用來參考父對話盒的 metrics。所以第二筆規則可以解讀爲「移動 IDC_B2 控制元件至距離父對話盒右緣 8 個對話盒單位的位置上」。不論使用者如何拉大此對話盒，IDC_B1 控制元件會一直置於距父對話盒 8 個對話盒單位的位置上。

第三筆規則是爲了調整使 IDC_B2 控制元件的左緣對齊 IDC_B1 控制元件的左緣。我們也可以指定 act-on 欄位爲 lLEFT(IDC_B1)，relative-to 欄位爲 lLEFT(IDC_B2)，效果相同，但這時候是 IDC_B1 的左緣對齊 IDC_B2 的左緣。

第四筆以及最後一筆項目是使 IDC_B1 和 IDC_B2 成爲一個群組，並將它們放置於 lPARENT 對話盒垂直高度的中心。欲將群組（group）置於對話盒中央，Layout 首先會將 IDC_B2 控制元件放在 IDC_B1 控制元件下方，然後計算這個群組（兩個控制元件）的矩形範圍，並將該矩形置於對話盒中央，再適當移動群組中的每一項控制元件。

Layout 的四個動作被定義在表 **3.6** 中。

表 3.6　Layout 所支援的動作巨集

動作	意義
lSTRETCH	以另一個控制元件的 metric 爲基準，改變一個控制元件的 metric。一次只有一個 metric 可以被改變，所以控制元件會被賦與新的寬度或高度，而不會搬移位置。如果你拉大控制元件的左緣，其右緣會固定不動；反之亦然。如果你改變一個控制元件的寬度，控制元件會得到一個新寬度，但左緣和右緣都不會變化。當你想調整一個控制元件的大小，而它卻又還有其他規則在身（例如「置於中央」等等），那麼改變其寬度和高度將是很有用的方法。
lMOVE	這項動作會維持控制元件的原本大小，但是有某一邊會相對於另一控制元件的一邊而移動，就像移動 A 控制元件的左邊去緊鄰 B 控制元件的右邊一樣。實際上，你可以以另一控制元件的寬度或高度爲基準，移動控制元件的某一邊。當你要以父對話盒的寬度或高度爲基準來放置你的控制元件，這是很有用的方法。舉個例子，lMOVE, lLEFT (IDC_B1), lWIDTHOF (lPARENT,33), 0　會移動 IDC_B1 控制元件，使其左側落在父對話盒寬度 1/3 的地方。

動作	意義
lHCENTER lVCENTER	指定一群控制元件放置在一個控制元件中央。如果你指定的是 lHCENTER，就會置於水平位置中央，如果你指定的是 lVCENTER，就會置於垂直位置中央。通常所謂置於中央是相對於 lPARENT 而言，但並不一定如此。事實上本章的 DlgSize 程式便是將一個群組（兩個按鈕）置於 edit 控制元件的中央。

現在你知道如何使用 Layout 技術來排列你的控制元件了，你可以嘗試修改稍後我將介紹的 DlgSize 程式中的規則，以獲取一些經驗。

Layout 演算法的實作

以下概略介紹 Layout 演算法：

1. 建立你想在對話盒中佈局的子控制元件的各個規則，在其中記錄每一個子視窗的已知尺寸（metrics）。
2. 走訪整個規則串列，將所有會受到規則影響的控制元件的大小記錄視爲未知。Layout 的目標就是希望藉由一些已知量來計算出未知量。重要的是，父對話盒（lPARENT）的所有六個尺寸總是已知。已知量，如父對話盒的尺寸和子控制元件的任何邊緣，都可以提供給「規則系統」做爲必要的參考資料。
3. 走訪整個規則串列，將偏移量的「對話盒單位」轉換爲圖素（pixel）單位，並將串列中的每一個規則標示爲未完成。
4. 重複走訪整個串列，試圖完成每一筆尚未被完成的規則，直到「完全走過一遍而仍然不能夠完成任何一筆規則」才結束，這表示我們已經不能再有什麼貢獻了。如果這時候還有任何一筆規則未能完成，或是有任何子控制元件的尺寸仍然未知，表示 Layout 失敗，它會 "asserts" 並離開。
5. 假如所有規則都順利完成，而且每個子控制元件的四邊都爲已知，我們便成功了。此時我們就可以使用 DeferWindowPos 函式將所有的控制元件搬移到它們的新位置上。

由於這是一本 Windows 程式設計書籍而不是一本介紹 Layout 演算法的書，我並不打算解釋 Layout 工作原理的細節。如果你對於 Layout 的工作原理感興趣，請你仔細閱讀並瞭解在 Layout.C 和 Layout.H 檔中程式碼的每一個動作與註解。當你對此已有相當瞭解之後，你便有能力規畫並除錯一組簡單的對話盒規則。

在探討 DlgSize 程式之前，讓我們再看看上述步驟之中與 Windows 有關的部份。演算法的第一個步驟會建立由此對話盒之所有子控制元件所構成的一個串列，這可以利用 GetFirstChild 和 GetNextSibling 巨集完成，它們定義在 WindowsX.H 表頭檔中。這些巨集分別使用 GetTopWindow 和 GetWindow 函式獲得 z-order 最上層的子視窗和其在 window list 中的下一個兄弟視窗。「列舉出所有子視窗」是一個很常見的動作，所以這些巨集對於你自己的應用程式或許也很有幫助。

在第三個步驟中，我們使用 adgMapDialogRect 函式來轉換偏移量（offset）-- 從對話盒單位轉換爲圖素（pixel）。由於我希望 Layout 可針對任何視窗（而不僅僅是對話盒）運作，所以我不能使用 MapDialogRect 函式，因爲 MapDialogRect 僅在對話盒中有效。adgMapDialogRect 首先檢查視窗類別的 GCW_ATOM 值，看看是否是 32770，這是 Windows 的對話盒類別的 atom 值。如果對象是一個對話盒，就呼叫 MapDialogRect。如果不是對話盒，我們必須模擬 quirky mapping 演算法，也就是 MapDialogRect 函式的內部動作。這個 MapDialogRect 函式首先計算一個字元的平均寬度（藉由一整個字串的寬度，該字串包含所有大寫字母與小寫字母，a～z 和 A～Z）[1]，然後將這字串的寬度除以字串長度（字元個數）以取得字元平均寬度（四捨五入）。將此字元平均寬度設定給 cxChar 變數，然後就可以這樣子將矩形大小從對話盒單位轉換爲圖素（pixels）單位：

```
SetRect(prc,
     prc->left  * cxChar / 4, prc->top    * cyChar / 8,
     prc->right * cxChar / 4, prc->bottom * cyChar / 8);
```

[1] Quirky Mapping 演算法適用於 Windows 的任何一種語言版本（英文版、法文版、中文版…）。

演算法的最後一個步驟中，Layout 使用了 BeginDeferWindowPos 函式建立起一個「所有將被移動之視窗」的串列。它使用 DeferWindowPos 函式，將位置資訊加入串列之中，然後呼叫 EndDeferWindowPos，同時將所有視窗移動至指定位置上。當你有數個視窗要移動，使用 DeferWindowPos 函式會比使用 SetWindowPos 函式來得更有效率。

DlgSize 程式

DlgSize 程式(DlgSize.EXE)的原始碼顯示於**程式列表 3.18 ~ 3.22**，示範如何使用 Layout 技術來重新定位一個可改變大小的對話盒的子視窗。當你執行這個程式，主視窗如**圖 3.18** 所示。DlgSize 包含的控制元件有三個 button、兩個 edit、兩個 static 和一個 groupbox。這些控制元件都沒有真正的功用，它們的出現只是爲了說明方便。當 DlgSize 對話盒的大小改變，會使用 Layout 技術來重新排列其控制元件。如果改變 DlgSize 對話盒的垂直大小與水平大小，結果將如**圖 3.19** 所示。

建立此程式所需使用到的檔案，列在**表 3.7** 中。

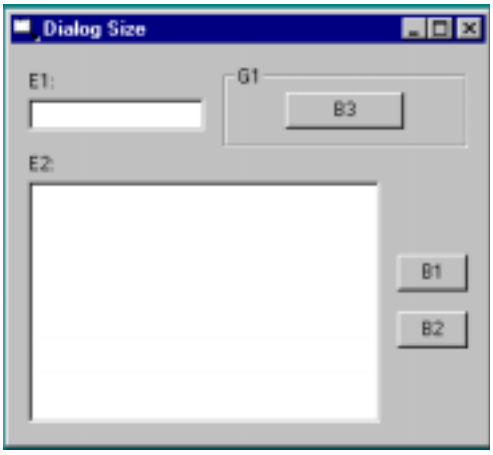


圖 3.18　DlgSize 程式的主視窗

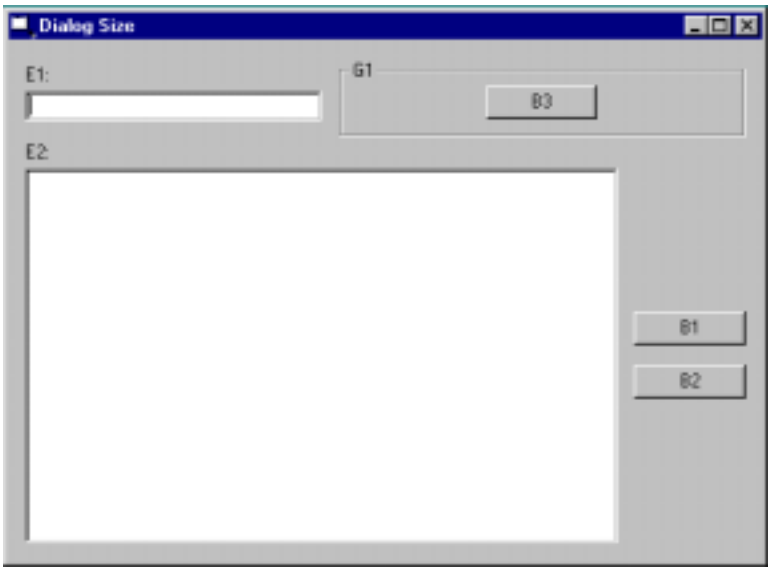


圖 3.19　調整大小後的 DlgSize 對話盒

表 3.7　建立 DlgSize 程式所需的檔案

檔案	說明
DlgSize.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Layout.C	函式庫，使你很容易動態配置對話盒的子視窗。
Layout.H	內含 Layout.C 檔中所有函式的宣告。
DlgSize.RC	內含主視窗的對話盒面板（template）及其圖示（icon）。
Resource.H	內含 DlgSize.RC 檔中所有資源的 ID。
DlgSize.ICO	主視窗圖示（icon）。
DlgSize.MAK	Visual C++ 的 MAK 檔。

DlgSize_OnSize 訊息處理常式（在 DlgSize.C 檔中）只是呼叫 Layout_ComputeLayout 來重新定位所有子控制元件。但是我們必須加上一些限制：我們不允許 DlgSize 對話盒比其預設視窗小。如果 DlgSize 對話盒太小，Layout 會 “assert”，因爲如此一來 DlgSize 對話

盒的子視窗大小將是負值[2]。爲防止這種情況發生，我們必須限制使用者對於對話盒大小的調整量。我們可以儲存對話盒預設大小至 GWL_USERDATA 中，然後利用此值做檢查。一旦檢查發現越過了其最小量，就回應一個 WM_GETMINIMAXINFO 訊息：

```
void DlgSize_OnGetMinMaxInfo (HWND hwnd, LPMINMAXINFO lpMinMaxInfo) {

   // Override the default minimum tracking size with our own size, which was
   // computed and saved in GWL_USERDATA in OnInitDialog.
   lpMinMaxInfo->ptMinTrackSize = *(POINT*) GetWindowLong(hwnd, GWL_USERDATA);
}
```

這個技巧可以防止程式使用者將對話盒調整得太小（比其預設值小）。

"Layout" 技術的其它概念

雖然我們僅探討了處理 Layout 的 OnSize 訊息處理常式，但你也可以自行處理 Layout 的其它訊息，完成一些特別需求。舉個例子，有時候對話盒會爲了某種原因，利用 CreateWindowEx 動態產生子控制元件。這很麻煩，因爲你必需自行計算控制元件的位置。然而如果你使用 Layout 技術，你甚至可以產生 0 尺寸的控制元件，然後再利用 Layout 中的規則來定位和調整其大小。

另外，使用 Layout 來重新定位子控制元件時，可能會發生彼此產生交集的情況。某些情況下，你可以指定兩個（或多個）不同的規則來設定一個對話盒外觀（例如一個控制元件的隱藏和顯示狀態，或者一個 checkbox 的 "check" 狀態）。記住，規則串列不需是靜態不動的，你可以依情況修改一筆或多筆規則，甚至可以動態建立一整個規則串列。

---

[2] 舉個例子，如果 DlgSize 對話盒太小，規則 lSTRETCH, lRIGHT (IDC_E2), lLEFT (IDC_B1), -8 會使得 IDC_E2 控制元件有一個負寬度。這是因爲 IDC_B1 的寬度是固定的，且位於距對話盒右緣 8 個對話盒單位。如果對話盒非常窄，IDC_B1 的左緣將會超出 IDC_E2 左緣，於是 IDC_E2 做了上述動作後，會使 IDC_E2 的寬度爲一個負值。事實上，要判斷是否在你的規則中產生這類情況，最簡單的方法就是不斷的嘗試。如果你想防止這種情況的發生，但又缺乏一些幾何概念，你可能就需以某種方法限制對話盒的大小。

DlgSize.ICO

程式列表 3.18　DlgSize.C

```
#0001  /****************************************************************
#0002  Module name: DlgSize.c
#0003  Written by: Jonathan W. Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Example to demonstrate use of the layout algorithm (see Layout.c)
#0006  ****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include "Resource.h"
#0014  #include "Layout.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  void DlgSize_OnSize (HWND hwnd, UINT state, int cx, int cy) {
#0021
#0022     static RULE rules[] = {
#0023
#0024        // Action    Act-on                Relative-to          Offset
#0025        // ------    ------                -----------          ------
#0026
#0027        {  lMOVE,    lRIGHT (IDC_B1),       lRIGHT  (lPARENT),     -8  },
#0028        {  lSTRETCH, lWIDTH (IDC_B1),       lWIDTHOF(lPARENT, 15), +0  },
#0029        {  lMOVE,    lTOP   (IDC_B2),       lBOTTOM (IDC_B1),      +8  },
#0030        {  lMOVE,    lLEFT  (IDC_B2),       lLEFT   (IDC_B1),      +0  },
#0031        {  lSTRETCH, lWIDTH (IDC_B2),       lWIDTH  (IDC_B1),      +0  },
#0032        {  lVCENTER, lCHILD (IDC_B3),       lCHILD  (IDC_G1),      +2  },
#0033        {  lHCENTER, lCHILD (IDC_B3),       lCHILD  (IDC_G1),      +0  },
#0034        {  lVCENTER, lGROUP (IDC_B1, IDC_B2), lCHILD  (IDC_E2),    +0  },
#0035        {  lSTRETCH, lWIDTH (IDC_E1),        lWIDTHOF(IDC_E2, 50), +0  },
#0036        {  lSTRETCH, lBOTTOM(IDC_E2),        lBOTTOM (lPARENT),     -8  },
#0037        {  lSTRETCH, lRIGHT (IDC_E2),        lLEFT   (IDC_B1),      -8  },
#0038        {  lMOVE,    lLEFT  (IDC_G1),        lRIGHT  (IDC_E1),      +8  },
#0039        {  lSTRETCH, lRIGHT (IDC_G1),        lRIGHT  (lPARENT),     -8  },
#0040        {  lEND                                                    }
#0041
#0042     };
#0043
#0044     // Place the dialog's child controls according to the rules. If there is
```

```
#0045    // a problem with the rules, Layout_ComputeLayout will assert.
#0046    Layout_ComputeLayout(hwnd, rules);
#0047 }
#0048
#0049
#0050 ////////////////////////////////////////////////////////////////
#0051
#0052
#0053 BOOL DlgSize_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0054
#0055    RECT rc;
#0056    POINT *pptMin;
#0057
#0058    adgSETDLGICONS(hwnd, IDI_LAYOUT, IDI_LAYOUT);
#0059
#0060    // Save the initial width/height of the dialog in the GWL_USERDATA extra
#0061    // bytes. We use this information later in DlgSize_OnGetMinMaxInfo to
#0062    // ensure that the dialog is not sized smaller than the initial size.
#0063    GetWindowRect(hwnd, &rc);
#0064    pptMin = malloc(sizeof (POINT));
#0065    if (pptMin) {
#0066       pptMin->x = rc.right - rc.left;
#0067       pptMin->y = rc.bottom - rc.top;
#0068    }
#0069    SetWindowLong(hwnd, GWL_USERDATA, (LONG) pptMin);
#0070    return(TRUE);               // Accept default focus window.
#0071 }
#0072
#0073
#0074 ////////////////////////////////////////////////////////////////
#0075
#0076
#0077 void DlgSize_OnDestroy(HWND hwnd) {
#0078
#0079    // Free the minimum tracking size data that is stored in GWL_USERDATA.
#0080    // If malloc fails in OnInitDialog, this value may be NULL, but this is
#0081    // okay because ANSI C states that free is defined to ignore NULL pointers.
#0082    free((void*) GetWindowLong(hwnd, GWL_USERDATA));
#0083 }
#0084
#0085
#0086 ////////////////////////////////////////////////////////////////
#0087
#0088
#0089 void DlgSize_OnGetMinMaxInfo (HWND hwnd, LPMINMAXINFO lpMinMaxInfo) {
#0090
```

```
#0091    // Override the default minimum tracking size with our own size, which was
#0092    // computed and saved in GWL_USERDATA in OnInitDialog.
#0093    lpMinMaxInfo->ptMinTrackSize = *(POINT*) GetWindowLong(hwnd, GWL_USERDATA);
#0094  }
#0095
#0096
#0097  ////////////////////////////////////////////////////////////////
#0098
#0099
#0100  void DlgSize_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0101
#0102    switch (id) {
#0103       case IDCANCEL:            // Allows dialog box to close
#0104          EndDialog(hwnd, id);
#0105          break;
#0106    }
#0107  }
#0108
#0109
#0110  ////////////////////////////////////////////////////////////////
#0111
#0112
#0113  BOOL WINAPI DlgSize_DlgProc (HWND hwnd, UINT uMsg,
#0114    WPARAM wParam, LPARAM lParam) {
#0115
#0116    switch (uMsg) {
#0117
#0118       // Standard Windows messages
#0119       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG,    DlgSize_OnInitDialog);
#0120       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,      DlgSize_OnCommand);
#0121       adgHANDLE_DLGMSG(hwnd, WM_SIZE,         DlgSize_OnSize);
#0122       adgHANDLE_DLGMSG(hwnd, WM_GETMINMAXINFO, DlgSize_OnGetMinMaxInfo);
#0123       adgHANDLE_DLGMSG(hwnd, WM_DESTROY,      DlgSize_OnDestroy);
#0124    }
#0125    return(FALSE);              // We didn't process the message.
#0126  }
#0127
#0128
#0129  ////////////////////////////////////////////////////////////////
#0130
#0131
#0132  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0133    LPSTR lpszCmdLine, int nCmdShow) {
#0134
#0135    adgWARNIFUNICODEUNDERWIN95();
#0136    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_LAYOUT),
```

```
#0137       NULL, DlgSize_DlgProc));
#0138
#0139    return(0);
#0140  }
#0141
#0142
#0143  ////////////////////////// End of File ///////////////////////////
```

程式列表 3.19　Layout.C

```
#0001  /****************************************************************
#0002  Module name: Layout.c
#0003  Written by: Jonathan W. Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Implementation of a layout algorithm which positions controls
#0006          in a window based on a set of rules.
#0007  ****************************************************************/
#0008
#0009
#0010  #include "..\Win95ADG.h"         /* See Appendix A for details */
#0011  #include <windows.h>
#0012  #include <windowsx.h>
#0013  #pragma warning(disable: 4001)    /* Single line comment */
#0014  #include "Layout.h"
#0015
#0016
#0017  ///////////////// ID for Child List Terminator ////////////////////
#0018
#0019
#0020  #define IDC_LASTCHILD 0x10001 // Special child id for last child in list.
#0021                                // Guaranteed not to be used because you
#0022                                // cannot have control ids above 0xffff.
#0023
#0024
#0025  ///////////////////////////// Limits ////////////////////////////////
#0026
#0027
#0028  #define NUMPARTS    7        // There are 7 parts: left, top, right,
#0029                               // bottom, width, height and group
#0030  #define NUMMETRICS   6        // There are 6 metrics: left, top, right,
#0031                               // bottom, width and height.
#0032  #define NUMSIDES    4        // There are 4 sides: left, top, right and
#0033                               // bottom.
#0034
#0035
```

```
#0036  ///////////////// Macros for Asserting Part Types /////////////////
#0037
#0038
#0039  #define ISPART(n)   adgINRANGE(lpLEFT, (n), lpGROUP)
#0040  #define ISMETRIC(n) adgINRANGE(lpLEFT, (n), lpHEIGHT)
#0041  #define ISSIDE(n)   adgINRANGE(lpLEFT, (n), lpBOTTOM)
#0042
#0043
#0044  /////////////////////////// Metric Flags ///////////////////////////
#0045
#0046
#0047  typedef enum {
#0048     UNKNOWN,                    // Metric is unknown
#0049     KNOWN                       // Metric is known
#0050  } MFLAG;
#0051
#0052
#0053  /////////////////////////// Rule States ///////////////////////////
#0054
#0055
#0056  enum {
#0057     UNAPPLIED,                   // Rule has not been applied
#0058     APPLIED                     // Rule has been applied
#0059  };
#0060
#0061
#0062  ///////////////// Definition of a Child Control ///////////////
#0063
#0064
#0065  typedef struct tagCHILD {
#0066     int idc;                    // Control id (lPARENT represents parent)
#0067     union {                     // Child has either a list of metrics:
#0068        int anMetric[NUMMETRICS]; //   left, top, right, bottom, width, height
#0069        struct {                 // or a RECT, a width and a height:
#0070           RECT rc;              //   RECT (left, top, right, bottom)
#0071           int nWidth;           //   width
#0072           int nHeight;          //   height
#0073        };                       // depending on which is more readable in
#0074     };                          // a given context.
#0075     BOOL fFixed;                // TRUE if child doesn't need to be placed
#0076     MFLAG afMetric[NUMMETRICS]; // Metric flags (KNOWN or UNKNOWN)
#0077  } CHILD;
#0078
#0079
#0080  ///////////////// Local Function Prototypes /////////////////
#0081
```

```
#0082
#0083  BOOL adgMapDialogRect (HWND hwndParent, PRECT prc);
#0084
#0085  int Layout_GetOppositeSide (int nSide);
#0086  int Layout_GetOtherUnknownMetric (int nUnknownMetric);
#0087  BOOL Layout_MetricIsVertical (int nMetric);
#0088  void Layout_SolveChild (CHILD* pChild);
#0089  CHILD* Layout_FindChild (CHILD* pChildList, int idcChild);
#0090  CHILD* Layout_CreateChildList (HWND hwndParent, int* pnChildren);
#0091  void Layout_ConvertDlgUnits (HWND hwndParent, RULE* pRules, CHILD* pChildList);
#0092  void Layout_MarkUnknowns (HWND hwndParent, RULE* pRules, CHILD* pChildList);
#0093  BOOL Layout_CheckChild (CHILD* pChild);
#0094  BOOL Layout_ApplyRule (HWND hwndParent, RULE* pRules,
#0095     CHILD* pChildList, RULE* pRule);
#0096  BOOL Layout_ApplyRules (HWND hwndParent, RULE* pRules, CHILD* pChildList);
#0097
#0098
#0099  ////////////////////////////////////////////////////////////
#0100
#0101
#0102  // A MapDialogRect function that works for non-dialog windows.
#0103  BOOL adgMapDialogRect (HWND hwndParent, PRECT prc) {
#0104
#0105     HDC hdc;
#0106     SIZE size;
#0107     int cxChar, cyChar;
#0108     HFONT hfont, hfontOriginal;
#0109
#0110     // This is the set of characters that Windows uses to compute the average
#0111     // character width.
#0112     static TCHAR szChars[] =
#0113        __TEXT("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");
#0114
#0115     // Check assumptions.
#0116     adgASSERT(IsWindow(hwndParent));
#0117     adgASSERT(prc);
#0118
#0119     // If the window is a dialog, just use MapDialogRect.
#0120     if (GetClassWord(hwndParent, GCW_ATOM) == 32770)
#0121        return(MapDialogRect(hwndParent, prc));
#0122
#0123     // Get a device context and select the window's font into it.
#0124     hdc = GetDC(hwndParent);
#0125     hfont = GetWindowFont(hwndParent);
#0126     if (hfont != NULL)
#0127        hfontOriginal = SelectFont(hdc, hfont);
```

```
#0128
#0129    // Unfortunately, we cannot use GetTextMetrics to get the average character
#0130    // width because the TEXTMETRIC structure's tmAveCharWidth member is
#0131    // incorrect for proportional fonts. So, instead we compute the average
#0132    // character width ourselves using the same technique employed by Windows
#0133    // itself:  We pass "a-zA-Z" to GetTextExtentPoint and average, rounding up.
#0134    // (NOTE: We do not call GetTextExtentPoint32 because this function corrects
#0135    // an error that Windows relies on)
#0136    GetTextExtentPoint(hdc, szChars, adgARRAY_SIZE(szChars), &size);
#0137    cyChar = size.cy;
#0138    cxChar = ((size.cx / (adgARRAY_SIZE(szChars) / 2)) + 1) / 2;
#0139
#0140    // Restore any original font and then release the device context.
#0141    if (hfont != NULL)
#0142       SelectFont(hdc, hfontOriginal);
#0143    ReleaseDC(hwndParent, hdc);
#0144
#0145    // Map rectangle prc based on the font dimensions (cxChar by cyChar).
#0146    SetRect(prc,
#0147       prc->left  * cxChar / 4, prc->top    * cyChar / 8,
#0148       prc->right * cxChar / 4, prc->bottom * cyChar / 8);
#0149    return(TRUE);
#0150 }
#0151
#0152
#0153 ////////////////////////////////////////////////////////////////
#0154
#0155
#0156 int Layout_GetOppositeSide (int nSide) {
#0157
#0158    int nOppositeSide;
#0159
#0160    // Check assumptions.
#0161    adgASSERT(ISSIDE(nSide));
#0162
#0163    switch (nSide) {
#0164       case lpLEFT:
#0165          nOppositeSide = lpRIGHT;
#0166          break;
#0167
#0168       case lpRIGHT:
#0169          nOppositeSide = lpLEFT;
#0170          break;
#0171
#0172       case lpTOP:
#0173          nOppositeSide = lpBOTTOM;
```

```
#0174         break;
#0175
#0176      case lpBOTTOM:
#0177         nOppositeSide = lpTOP;
#0178         break;
#0179
#0180        default:               // Invalid side
#0181         adgFAIL(__TEXT("Invalid side"));
#0182         break;
#0183    }
#0184    return(nOppositeSide);
#0185 }
#0186
#0187
#0188 ////////////////////////////////////////////////////////////////
#0189
#0190
#0191 int Layout_GetOtherUnknownMetric (int nUnknownMetric) {
#0192
#0193    int nOtherUnknownMetric = 0;
#0194
#0195    // Check assumptions.
#0196    adgASSERT(ISMETRIC(nUnknownMetric));
#0197
#0198    switch (nUnknownMetric) {
#0199       case lpLEFT:
#0200       case lpRIGHT:
#0201          nOtherUnknownMetric = lpWIDTH;
#0202          break;
#0203
#0204       case lpTOP:
#0205       case lpBOTTOM:
#0206          nOtherUnknownMetric = lpHEIGHT;
#0207          break;
#0208
#0209       case lpWIDTH:
#0210          nOtherUnknownMetric = lpRIGHT;
#0211          break;
#0212
#0213       case lpHEIGHT:
#0214          nOtherUnknownMetric = lpBOTTOM;
#0215          break;
#0216
#0217       default:
#0218          adgFAIL(__TEXT("Invalid metric"));
#0219          break;
```

```
#0220    }
#0221    return(nOtherUnknownMetric);
#0222  }
#0223
#0224
#0225  ////////////////////////////////////////////////////////////////
#0226
#0227
#0228  BOOL Layout_MetricIsVertical (int nMetric) {
#0229
#0230    BOOL fMetricIsVertical = FALSE;
#0231
#0232    // Check assumptions.
#0233    adgASSERT(ISMETRIC(nMetric));
#0234
#0235    switch (nMetric) {
#0236      case lpLEFT:
#0237      case lpRIGHT:
#0238      case lpWIDTH:
#0239         fMetricIsVertical = FALSE;
#0240         break;
#0241
#0242       case lpTOP:
#0243       case lpBOTTOM:
#0244       case lpHEIGHT:
#0245         fMetricIsVertical = TRUE;
#0246         break;
#0247
#0248       default:
#0249         adgFAIL(__TEXT("Invalid metric"));
#0250         break;
#0251    }
#0252    return(fMetricIsVertical);
#0253  }
#0254
#0255
#0256  ////////////////////////////////////////////////////////////////
#0257
#0258
#0259  void Layout_SolveChild (CHILD* pChild) {
#0260
#0261    int i;
#0262
#0263    // Check assumptions.
#0264    adgASSERT(pChild);
#0265
```

```
#0266    // Loop through all six metrics of a child, computing values for unknown
#0267    // metrics from values of known metrics (if any).
#0268    for (i = 0; i < NUMMETRICS; i++) {
#0269
#0270       // If this metric of the child is unknown, see if it can be computed in
#0271       // terms of other metrics which we do know.
#0272       if (pChild->afMetric[i] == UNKNOWN) {
#0273
#0274          // Compute left/top as right/bottom - width/height
#0275          if (i < 2) {
#0276             if ((pChild->afMetric[i + 2] == KNOWN) &&
#0277                 (pChild->afMetric[i + 4] == KNOWN)) {
#0278                pChild->anMetric[i] = pChild->anMetric[i + 2] -
#0279                   pChild->anMetric[i + 4];
#0280                pChild->afMetric[i] = KNOWN;
#0281             }
#0282          }
#0283          // Compute right/bottom as left/top + width/height
#0284          else if (i < 4) {
#0285             if ((pChild->afMetric[i - 2] == KNOWN) &&
#0286                 (pChild->afMetric[i + 2] == KNOWN)) {
#0287                pChild->anMetric[i] = pChild->anMetric[i - 2] +
#0288                   pChild->anMetric[i + 2];
#0289                pChild->afMetric[i] = KNOWN;
#0290             }
#0291          }
#0292          // Compute width/height as right/bottom - left/top
#0293          else {
#0294             if ((pChild->afMetric[i - 2] == KNOWN) &&
#0295                 (pChild->afMetric[i - 4] == KNOWN)) {
#0296                pChild->anMetric[i] = pChild->anMetric[i - 2] -
#0297                   pChild->anMetric[i - 4];
#0298                pChild->afMetric[i] = KNOWN;
#0299             }
#0300          }
#0301       }
#0302    }
#0303 }
#0304
#0305
#0306 ////////////////////////////////////////////////////////////////
#0307
#0308
#0309 CHILD* Layout_FindChild (CHILD* pChildList, int idcChild) {
#0310
#0311    CHILD *pChild;
```

```
#0312
#0313    // Check assumptions.
#0314    adgASSERT(pChildList);
#0315    adgASSERT(adgINRANGE(0, idcChild, IDC_LASTCHILD));
#0316
#0317    // Traverse the child list looking for an id
#0318    for (pChild = pChildList; pChild->idc != IDC_LASTCHILD; pChild++) {
#0319
#0320       // If we find idcChild, we solve the child for unknowns and return it
#0321       if (pChild->idc == idcChild) {
#0322          Layout_SolveChild(pChild);
#0323          return(pChild);
#0324       }
#0325    }
#0326    adgFAIL(__TEXT("Child not found in child list"));
#0327    return(NULL);
#0328 }
#0329
#0330
#0331 ////////////////////////////////////////////////////////////////
#0332
#0333
#0334 CHILD* Layout_CreateChildList (HWND hwndParent, int* pnChildren) {
#0335
#0336    int i;
#0337    HWND hwnd, hwndFirst;
#0338    CHILD* pChild, *pChildList;
#0339
#0340    // Check assumptions.
#0341    adgASSERT(IsWindow(hwndParent));
#0342    adgASSERT(pnChildren);
#0343
#0344    // Count the number of child windows in hwndParent.
#0345    hwndFirst = hwnd = GetFirstChild(hwndParent);
#0346    for (*pnChildren = 0; IsWindow(hwnd); hwnd = GetNextSibling(hwnd))
#0347       (*pnChildren)++;
#0348    if (*pnChildren == 0)
#0349       return(NULL);
#0350
#0351    // Allocate memory for the CHILD list. This list will have an entry for
#0352    // each child of the dialog, plus a CHILD structure for the parent window
#0353    // (lPARENT) and one which acts as a list terminator (IDC_LASTCHILD).
#0354    pChildList = malloc((*pnChildren + 2) * sizeof(CHILD));
#0355    adgASSERT(pChildList);
#0356    if (!pChildList)
#0357       return(NULL);
```

```
#0358    pChild = pChildList;
#0359
#0360    // Add the special-case parent 'CHILD' structure
#0361    pChild->idc = lPARENT;
#0362    GetClientRect(hwndParent, &pChild->rc);
#0363    for (i = 0; i < NUMSIDES; i++)
#0364       pChild->afMetric[i] = KNOWN;
#0365    pChild->afMetric[lpWIDTH] = pChild->afMetric[lpHEIGHT] = UNKNOWN;
#0366    Layout_SolveChild(pChild);
#0367    pChild++;
#0368
#0369    // Add all the real children of the dialog to the list
#0370    hwnd = hwndFirst;
#0371    for (; IsWindow(hwnd); hwnd = GetNextSibling(hwnd)) {
#0372
#0373       // Get child's id and bounding rectangle in client coordinates
#0374       pChild->idc = GetWindowID(hwnd);
#0375       GetWindowRect(hwnd, &pChild->rc);
#0376       MapWindowRect(HWND_DESKTOP, hwndParent, &pChild->rc);
#0377       for (i = 0; i < NUMSIDES; i++)
#0378          pChild->afMetric[i] = KNOWN;
#0379
#0380       // Solve for the width and height.
#0381       pChild->afMetric[lpWIDTH] = pChild->afMetric[lpHEIGHT] = UNKNOWN;
#0382       Layout_SolveChild(pChild);
#0383
#0384       // All children are fixed, initially.
#0385       pChild->fFixed = TRUE;
#0386       pChild++;
#0387    }
#0388
#0389    // Terminate and return the list.
#0390    pChild->idc = IDC_LASTCHILD;
#0391    return(pChildList);
#0392 }
#0393
#0394
#0395 ////////////////////////////////////////////////////////////////
#0396
#0397
#0398 void Layout_ConvertDlgUnits (HWND hwndParent, RULE* pRules, CHILD* pChildList) {
#0399
#0400    RECT rc = { 0, 0, 0, 0 };
#0401    BOOL fVertical;
#0402    RULE* pRule;
#0403
```

```
#0404    // Check assumptions.
#0405    adgASSERT(IsWindow(hwndParent));
#0406    adgASSERT(pRules);
#0407    adgASSERT(pChildList);
#0408
#0409    // Traverse the rules list
#0410    for (pRule = pRules; pRule->Action != lEND; pRule++) {
#0411
#0412       // Simultaneously map the rule's offset value, vertically and
#0413       // horizontally, from dialog units to pixels.
#0414       rc.right = rc.bottom = pRule->nOffset;
#0415       adgMapDialogRect(hwndParent, &rc);
#0416
#0417       // Determine if the current rule affects horizontal or vertical
#0418       // coordinates. We need to know this because dialog unit space is not
#0419       // isometric (horizontal and vertical dialog units are not equivalent).
#0420       switch (pRule->Action) {
#0421
#0422          case lVCENTER:
#0423             fVertical = TRUE;
#0424             break;
#0425
#0426          case lHCENTER:
#0427             fVertical = FALSE;
#0428             break;
#0429
#0430          case lMOVE:
#0431          case lSTRETCH:
#0432             fVertical = Layout_MetricIsVertical(pRule->ActOn.nMetric);
#0433             break;
#0434
#0435          default:
#0436             adgFAIL(__TEXT("Invalid action"));
#0437             break;
#0438       }
#0439
#0440       // Take the correct mapped value based on the rule being applied
#0441       pRule->nPixelOffset = fVertical ? rc.bottom : rc.right;
#0442    }
#0443 }
#0444
#0445
#0446 ////////////////////////////////////////////////////////////
#0447
#0448
#0449 void Layout_MarkUnknowns (HWND hwndParent, RULE* pRules, CHILD* pChildList) {
```

```
#0450
#0451    RULE* pRule;
#0452    CHILD* pChildActOn;
#0453    HWND hwnd;
#0454    int nOtherUnknown, nOppositeSide, idc, idcFirst, idcLast;
#0455
#0456    // Check assumptions.
#0457    adgASSERT(IsWindow(hwndParent));
#0458    adgASSERT(pRules);
#0459    adgASSERT(pChildList);
#0460
#0461    // Traverse the rule list, marking unknowns in the child list.
#0462    for (pRule = pRules; pRule->Action != lEND; pRule++) {
#0463
#0464       // Set metric flags based on the rule's proposed action
#0465       switch (pRule->Action) {
#0466
#0467          case lSTRETCH:         // Metric should be stretched
#0468
#0469             // Find the child to be acted upon
#0470             pChildActOn = Layout_FindChild(pChildList, pRule->ActOn.idc);
#0471             adgASSERT(pChildActOn);
#0472
#0473             // Since the child is going to be acted upon, it is no longer fixed.
#0474             pChildActOn->fFixed = FALSE;
#0475
#0476             // The part being acted on must be a metric.
#0477             adgASSERT(ISMETRIC(pRule->ActOn.nMetric));
#0478
#0479             // The metric being stretched must be unknown.
#0480             pChildActOn->afMetric[pRule->ActOn.nMetric] = UNKNOWN;
#0481
#0482             // If the left/top or right/bottom is unknown, so is the
#0483             // width/height. If the width/height is unknown, then the
#0484             // right/bottom is also unknown.
#0485             nOtherUnknown = Layout_GetOtherUnknownMetric(pRule->ActOn.nMetric);
#0486             pChildActOn->afMetric[nOtherUnknown] = UNKNOWN;
#0487             break;
#0488
#0489          case lMOVE:            // Control should be moved
#0490
#0491             // Find the child to be acted upon
#0492             pChildActOn = Layout_FindChild(pChildList, pRule->ActOn.idc);
#0493             adgASSERT(pChildActOn);
#0494
#0495             // Since the child is going to be acted upon, it is no longer fixed.
```

```
#0496            pChildActOn->fFixed = FALSE;
#0497
#0498            // The part being acted upon must be a side.
#0499            adgASSERT(ISSIDE(pRule->ActOn.nSide));
#0500
#0501            // The side being moved is unknown.
#0502            pChildActOn->afMetric[pRule->ActOn.nSide] = UNKNOWN;
#0503
#0504            // So is the opposite side. But, the width/height remains known.
#0505            // (Actually, this is the primary reason for having six metrics).
#0506            nOppositeSide = Layout_GetOppositeSide(pRule->ActOn.nSide);
#0507            pChildActOn->afMetric[nOppositeSide] = UNKNOWN;
#0508            break;
#0509
#0510         case lVCENTER:          // Vertically center control/group
#0511         case lHCENTER:          // Horizontally center control/group
#0512
#0513            // We must be centering a group of one or more controls.
#0514            adgASSERT(pRule->ActOn.nPart == lpGROUP);
#0515
#0516            // Go through the group of one or more controls
#0517            idcFirst = pRule->ActOn.idcFirst;
#0518            idcLast = pRule->ActOn.idcLast;
#0519            adgASSERT(idcFirst <= idcLast);
#0520            hwnd = GetFirstChild(hwndParent);
#0521            for (; IsWindow(hwnd); hwnd = GetNextSibling(hwnd)) {
#0522               idc = GetWindowID(hwnd);
#0523               if (adgINRANGE(idcFirst, idc, idcLast)) {
#0524
#0525                  // Find the child to be acted upon and set the appropriate
#0526                  // sides to unknown. Width is still known.
#0527                  pChildActOn = Layout_FindChild(pChildList, idc);
#0528                  if (pRule->Action == lHCENTER) {
#0529                     pChildActOn->afMetric[lpLEFT] = UNKNOWN;
#0530                     pChildActOn->afMetric[lpRIGHT] = UNKNOWN;
#0531                  } else {
#0532                     pChildActOn->afMetric[lpTOP] = UNKNOWN;
#0533                     pChildActOn->afMetric[lpBOTTOM] = UNKNOWN;
#0534                  }
#0535
#0536                  // Child acted upon is no longer fixed.
#0537                  pChildActOn->fFixed = FALSE;
#0538               }
#0539            }
#0540            break;
#0541
```

```
#0542         default:
#0543            adgFAIL(__TEXT("Invalid action"));
#0544            break;
#0545       }
#0546    }
#0547 }
#0548
#0549
#0550 ////////////////////////////////////////////////////////////////
#0551
#0552
#0553 BOOL Layout_CheckChild (CHILD* pChild) {
#0554
#0555    static TCHAR* pszMetric[] = {
#0556       __TEXT("left"),   __TEXT("top"),   __TEXT("right"),
#0557       __TEXT("bottom"), __TEXT("width"), __TEXT("height")
#0558    };
#0559
#0560    int i;
#0561    BOOL fOK = TRUE;
#0562    TCHAR sz[80];
#0563
#0564    // Check assumptions.
#0565    adgASSERT(pChild);
#0566
#0567    // Any unknown metric indicates a problem with the rules, so we 'assert'.
#0568    for (i = 0; i < NUMMETRICS; i++) {
#0569       if (pChild->afMetric[i] == UNKNOWN) {
#0570          wsprintf(sz, __TEXT("Layout couldn't find %s of id=%d"),
#0571             pszMetric[i], pChild->idc);
#0572          adgMB(sz);
#0573          fOK = FALSE;
#0574       }
#0575    }
#0576    return(fOK);
#0577 }
#0578
#0579
#0580 ////////////////////////////////////////////////////////////////
#0581
#0582
#0583 BOOL Layout_ApplyRule (HWND hwndParent, RULE* pRules,
#0584    CHILD* pChildList, RULE* pRule) {
#0585
#0586    CHILD* pChildRelTo, *pChildActOn, ChildRelTo;
#0587    CHILD* pChild, *pChildListNew, *pSrc, *pDest;
```

```
#0588    int nRules, nMetric, nChildren;
#0589    int idcFirst, idcLast, nOffset, nCentered;
#0590    RECT rcBounds;
#0591    HWND hwnd, hwndFirst;
#0592    RULE *pr, *prn, *prNew;
#0593
#0594    // Check assumptions.
#0595    adgASSERT(IsWindow(hwndParent));
#0596    adgASSERT(pRules);
#0597    adgASSERT(pChildList);
#0598    adgASSERT(pRule);
#0599
#0600    // Find the child and part(s) that we are going to act relative to
#0601    pChildRelTo = Layout_FindChild(pChildList, pRule->RelTo.idc);
#0602    adgASSERT(pChildRelTo);
#0603
#0604    switch (pRule->RelTo.nPart) {
#0605
#0606       case lpLEFT:
#0607       case lpTOP:
#0608       case lpRIGHT:
#0609       case lpBOTTOM:
#0610       case lpWIDTH:
#0611       case lpHEIGHT:
#0612
#0613          // We can't apply a rule relative to a metric that is unknown.
#0614          if (pChildRelTo->afMetric[pRule->RelTo.nMetric] == UNKNOWN)
#0615             return(FALSE);
#0616          break;
#0617
#0618       case lpGROUP:
#0619
#0620          // We can't apply a rule relative to a control unless we know its
#0621          // left/top and right/bottom sides (for centering).
#0622          adgASSERT(pRule->RelTo.idcFirst == pRule->RelTo.idcLast);
#0623          adgASSERT((pRule->Action == lHCENTER) || (pRule->Action == lVCENTER));
#0624          if (pRule->Action == lHCENTER) {
#0625             if ((pChildRelTo->afMetric[lpLEFT] == UNKNOWN) ||
#0626                 (pChildRelTo->afMetric[lpRIGHT] == UNKNOWN))
#0627                return(FALSE);
#0628          } else {
#0629             if ((pChildRelTo->afMetric[lpTOP] == UNKNOWN) ||
#0630                 (pChildRelTo->afMetric[lpBOTTOM] == UNKNOWN))
#0631                return(FALSE);
#0632          }
#0633          break;
```

```
#0634    }
#0635
#0636    // Make a local copy of the child we are relative to. We need to do this
#0637    // because we may need to apply a percentage to the width/height metrics
#0638    // and we don't want to modify the actual child list.
#0639    ChildRelTo = *pChildRelTo;
#0640
#0641    // Use percentage to modify the width/height of the child we are relative to
#0642    if ((pRule->RelTo.nMetric == lpWIDTH) || (pRule->RelTo.nMetric == lpHEIGHT)) {
#0643       ChildRelTo.anMetric[pRule->RelTo.nMetric] *= pRule->RelTo.nPercent;
#0644       ChildRelTo.anMetric[pRule->RelTo.nMetric] /= 100;
#0645       Layout_SolveChild(&ChildRelTo);
#0646    }
#0647
#0648    // Apply our rule based on the action field
#0649    switch (pRule->Action) {
#0650
#0651       case lSTRETCH:            // Metric should be stretched
#0652
#0653          // The part being acted on must be a metric. If it is a width/height
#0654          // metric, it must be 100% of the width/height.
#0655          adgASSERT(ISMETRIC(pRule->ActOn.nMetric));
#0656          adgASSERT(ISSIDE(pRule->ActOn.nSide) ||
#0657             (pRule->ActOn.nPercent == 100));
#0658
#0659          // The part being acted relative to must be a metric.
#0660          adgASSERT(ISMETRIC(pRule->RelTo.nMetric));
#0661
#0662          // Find the child being acted on and stretch the specified metric.
#0663          pChildActOn = Layout_FindChild(pChildList, pRule->ActOn.idc);
#0664          adgASSERT(pChildActOn);
#0665          pChildActOn->anMetric[pRule->ActOn.nMetric] =
#0666             ChildRelTo.anMetric[pRule->RelTo.nMetric] + pRule->nPixelOffset;
#0667          pChildActOn->afMetric[pRule->ActOn.nMetric] = KNOWN;
#0668          Layout_SolveChild(pChildActOn);
#0669          pRule->fState = APPLIED;
#0670          return(TRUE);
#0671
#0672       case lMOVE:               // Whole control should be moved
#0673
#0674          // The part being moved must be a side.
#0675          adgASSERT(ISSIDE(pRule->ActOn.nSide));
#0676
#0677          // The part that is being acted relative to must be a metric.
#0678          adgASSERT(ISMETRIC(pRule->RelTo.nMetric));
#0679
```

```
#0680          // Find the child being acted on and move the specified side.
#0681          pChildActOn = Layout_FindChild(pChildList, pRule->ActOn.idc);
#0682          adgASSERT(pChildActOn);
#0683          pChildActOn->anMetric[pRule->ActOn.nSide] =
#0684             ChildRelTo.anMetric[pRule->RelTo.nMetric] + pRule->nPixelOffset;
#0685          pChildActOn->afMetric[pRule->ActOn.nSide] = KNOWN;
#0686          Layout_SolveChild(pChildActOn);
#0687          pRule->fState = APPLIED;
#0688          return(TRUE);
#0689
#0690       case lVCENTER:             // Vertically center a control/group
#0691       case lHCENTER:             // Horizontally center a control/group
#0692
#0693          // We can only center a group of one or more controls relative to
#0694          // another control (a single control is a 'group' of one control).
#0695          adgASSERT(pRule->ActOn.nPart == lpGROUP);
#0696          adgASSERT(pRule->RelTo.nPart == lpGROUP);
#0697          adgASSERT(pRule->RelTo.idcFirst == pRule->RelTo.idcLast);
#0698
#0699          // First id in group must be less than or equal to the last id
#0700          idcFirst = pRule->ActOn.idcFirst;
#0701          idcLast = pRule->ActOn.idcLast;
#0702          adgASSERT(idcFirst <= idcLast);
#0703
#0704          // Ensure that the width/height is known for each control in the
#0705          // group before proceeding with any centering.
#0706          hwndFirst = GetFirstChild(hwndParent);
#0707          for (hwnd = hwndFirst; IsWindow(hwnd); hwnd = GetNextSibling(hwnd)) {
#0708             int idc = GetWindowID(hwnd);
#0709             if (adgINRANGE(idcFirst, idc, idcLast)) {
#0710                pChildActOn = Layout_FindChild(pChildList, idc);
#0711                if (pRule->Action == lHCENTER) {
#0712                   if (pChildActOn->afMetric[lpWIDTH] == UNKNOWN)
#0713                      return(FALSE);
#0714                } else {
#0715                   if (pChildActOn->afMetric[lpHEIGHT] == UNKNOWN)
#0716                      return(FALSE);
#0717                }
#0718             }
#0719          }
#0720
#0721          // Create a new list of rules which contains the subset of rules
#0722          // which act on controls in the centered group.
#0723          for (nRules = 0, pr = pRules; pr->Action != lEND; pr++)
#0724             nRules++;
#0725          nRules++;
```

```
#0726          prNew = _alloca(nRules * sizeof(RULE));
#0727          prn = prNew;
#0728          for (pr = pRules; pr->Action != lEND; pr++) {
#0729             if (adgINRANGE(idcFirst, pr->ActOn.idc, idcLast)) {
#0730                if (pRule->Action == lHCENTER) {
#0731                   if ((pr->ActOn.nPart == lpLEFT) ||
#0732                       (pr->ActOn.nPart == lpRIGHT)) {
#0733                      *prn++ = *pr;
#0734                   }
#0735                } else {
#0736                   if ((pr->ActOn.nPart == lpTOP) ||
#0737                       (pr->ActOn.nPart == lpBOTTOM)) {
#0738                      *prn++ = *pr;
#0739                   }
#0740                }
#0741             }
#0742          }
#0743          prn->Action = lEND;
#0744
#0745          // Make a local copy of the child list and set everything to KNOWN.
#0746          nChildren = 0;
#0747          for (pChild = pChildList; pChild->idc != IDC_LASTCHILD; pChild++)
#0748             nChildren++;
#0749          nChildren++;
#0750          pChildListNew = _alloca(nChildren * sizeof(CHILD));
#0751          MoveMemory(pChildListNew, pChildList, nChildren * sizeof(CHILD));
#0752          for (pChild = pChildListNew; pChild->idc != IDC_LASTCHILD; pChild++)
#0753             for (nMetric = 0; nMetric < NUMMETRICS; nMetric++)
#0754                pChild->afMetric[nMetric] = KNOWN;
#0755
#0756          // Solve for the children being centered as a sub-problem.
#0757          if (!Layout_ApplyRules (hwndParent, prNew, pChildListNew)) {
#0758             adgFAIL(__TEXT("Unable to apply rules to centered children"));
#0759             return(FALSE);
#0760          }
#0761
#0762          // Compute the bounding rectangle of the group
#0763          SetRectEmpty(&rcBounds);
#0764          hwndFirst = GetFirstChild(hwndParent);
#0765          for (hwnd = hwndFirst; IsWindow(hwnd); hwnd = GetNextSibling(hwnd)) {
#0766             int idc = GetWindowID(hwnd);
#0767             if (adgINRANGE(idcFirst, idc, idcLast)) {
#0768                pChildActOn = Layout_FindChild(pChildListNew, idc);
#0769                UnionRect(&rcBounds, &rcBounds, &pChildActOn->rc);
#0770             }
#0771          }
```

```
#0772
#0773          // Find the offset required to center the group's bounding rectangle
#0774          // against the control we are relative to.
#0775          if (pRule->Action == lHCENTER) {
#0776             nCentered = ChildRelTo.anMetric[lpLEFT] +
#0777                ((ChildRelTo.anMetric[lpWIDTH] -
#0778                (rcBounds.right - rcBounds.left)) / 2);
#0779             nOffset = nCentered - rcBounds.left;
#0780          } else {
#0781             nCentered = ChildRelTo.anMetric[lpTOP] +
#0782                ((ChildRelTo.anMetric[lpHEIGHT] -
#0783                (rcBounds.bottom - rcBounds.top)) / 2);
#0784             nOffset = nCentered - rcBounds.top;
#0785          }
#0786
#0787          // Add in any additional offset from the rule.
#0788          nOffset += pRule->nPixelOffset;
#0789
#0790          // Go through the new child list, moving each control.
#0791          adgASSERT(pRule->ActOn.idcFirst <= pRule->ActOn.idcLast);
#0792          for (hwnd = hwndFirst; IsWindow(hwnd); hwnd = GetNextSibling(hwnd)) {
#0793             int idc = GetWindowID(hwnd);
#0794             if (adgINRANGE(idcFirst, idc, idcLast)) {
#0795                pChildActOn = Layout_FindChild(pChildListNew, idc);
#0796                if (pRule->Action == lHCENTER) {
#0797                   pChildActOn->anMetric[lpLEFT]  += nOffset;
#0798                   pChildActOn->anMetric[lpRIGHT] += nOffset;
#0799                } else {
#0800                   pChildActOn->anMetric[lpTOP]   += nOffset;
#0801                   pChildActOn->anMetric[lpBOTTOM] += nOffset;
#0802                }
#0803             }
#0804          }
#0805
#0806          // Now modify the real child list based on pChildListNew.
#0807          for (pSrc = pChildListNew, pDest = pChildList;
#0808             pSrc->idc != IDC_LASTCHILD; pSrc++, pDest++) {
#0809
#0810             if (adgINRANGE(idcFirst, pSrc->idc, idcLast)) {
#0811                if (pRule->Action == lHCENTER) {
#0812                   pDest->anMetric[lpLEFT]  = pSrc->anMetric[lpLEFT];
#0813                   pDest->anMetric[lpRIGHT] = pSrc->anMetric[lpRIGHT];
#0814                   pDest->afMetric[lpLEFT]  = KNOWN;
#0815                   pDest->afMetric[lpRIGHT] = KNOWN;
#0816                } else {
#0817                   pDest->anMetric[lpTOP]   = pSrc->anMetric[lpTOP];
```

```
#0818                pDest->anMetric[lpBOTTOM] = pSrc->anMetric[lpBOTTOM];
#0819                pDest->afMetric[lpTOP]    = KNOWN;
#0820                pDest->afMetric[lpBOTTOM] = KNOWN;
#0821             }
#0822          }
#0823       }
#0824
#0825       pRule->fState = APPLIED;
#0826       return(TRUE);
#0827
#0828    default:
#0829       adgFAIL(__TEXT("Invalid action"));
#0830       return(FALSE);
#0831 }
#0832 }
#0833
#0834
#0835 ////////////////////////////////////////////////////////////
#0836
#0837
#0838 BOOL Layout_ApplyRules (HWND hwndParent, RULE* pRules, CHILD* pChildList) {
#0839
#0840   RULE* pRule;
#0841   BOOL fAppliedAtLeastOneRule, fOK = TRUE;
#0842
#0843   // Check assumptions.
#0844   adgASSERT(IsWindow(hwndParent));
#0845   adgASSERT(pRules);
#0846   adgASSERT(pChildList);
#0847
#0848   // Based on the list of rules, mark all unknown child metrics as UNKNOWN.
#0849   Layout_MarkUnknowns(hwndParent, pRules, pChildList);
#0850
#0851   // Traverse the rule list, converting offsets from dialog units to pixels.
#0852   Layout_ConvertDlgUnits(hwndParent, pRules, pChildList);
#0853
#0854   // Mark all the rules as unapplied before attempting to apply them.
#0855   for (pRule = pRules; pRule->Action != lEND; pRule++)
#0856      pRule->fState = UNAPPLIED;
#0857
#0858   // Loop through the rule list for as long as we are able to apply at least
#0859   // one rule (if we make a pass through the entire list, and we are unable
#0860   // to apply any rule, we are finished).
#0861   do {
#0862      fAppliedAtLeastOneRule = FALSE;
#0863      for (pRule = pRules; pRule->Action != lEND; pRule++) {
```

```
#0864         if (pRule->fState != APPLIED) {
#0865            if (Layout_ApplyRule(hwndParent, pRules, pChildList, pRule))
#0866               fAppliedAtLeastOneRule = TRUE;
#0867         }
#0868      }
#0869   } while (fAppliedAtLeastOneRule);
#0870
#0871   // Verify that all rules have been successfully applied.
#0872   for (pRule = pRules; pRule->Action != lEND; pRule++) {
#0873      adgASSERT(pRule->fState == APPLIED);
#0874      if (pRule->fState != APPLIED)
#0875         fOK = FALSE;
#0876   }
#0877   return(fOK);
#0878 }
#0879
#0880 ////////////////////////////////////////////////////////////////
#0881
#0882
#0883 BOOL WINAPI Layout_ComputeLayout (HWND hwndParent, RULE* pRules) {
#0884
#0885   HDWP hdwp;
#0886   BOOL fOK = TRUE;
#0887   CHILD* pChild, *pChildList;
#0888   int nChildren;
#0889
#0890   // Check assumptions.
#0891   adgASSERT(IsWindow(hwndParent));
#0892   adgASSERT(pRules);
#0893
#0894   // Don't do anything to a minimized window.
#0895   if (IsIconic(hwndParent))
#0896      return(TRUE);
#0897
#0898   // Enumerate all child windows of the dialog, allocating a CHILD structure
#0899   // for each child, with all six metric flags set to KNOWN. To simplify
#0900   // coding, we also add a special CHILD structure for the parent window with
#0901   // the id lPARENT (defined in layout.h). If there are no children, or
#0902   // memory cannot be allocated for the child list, we do nothing.
#0903   pChildList = Layout_CreateChildList(hwndParent, &nChildren);
#0904   if (pChildList == NULL)
#0905      return(FALSE);
#0906   if (nChildren == 0)
#0907      return(TRUE);
#0908
#0909   // Apply the rules from the rule list to solve for the locations of all the
```

```
#0910    // child controls.
#0911    if (!Layout_ApplyRules(hwndParent, pRules, pChildList)) {
#0912       adgFAIL(__TEXT("Unable to apply rules"));
#0913       return(FALSE);
#0914    }
#0915
#0916    // Simultaneously relocate all the children using DeferWindowPos.
#0917    hdwp = BeginDeferWindowPos(0);
#0918    adgASSERT(hdwp);
#0919
#0920    // Move each child in the CHILD list.  We enumerate the child list starting
#0921    // at pChildList + 1, because the first CHILD is lPARENT.
#0922    for (pChild = pChildList + 1; pChild->idc != IDC_LASTCHILD; pChild++) {
#0923
#0924       // Check child for any still-unsolved metrics. You may want to remove or
#0925       // #ifdef out this check once your rules are known to be working.
#0926       if (!Layout_CheckChild(pChild))
#0927          fOK = FALSE;
#0928
#0929       // Add child to DeferWindowPos list if it is not fixed.
#0930       if (!pChild->fFixed) {
#0931          HWND hwndChild = GetDlgItem(hwndParent, pChild->idc);
#0932          adgASSERT(pChild->anMetric[lpWIDTH] >= 0);
#0933          adgASSERT(pChild->anMetric[lpHEIGHT] >= 0);
#0934          hdwp = DeferWindowPos(hdwp, hwndChild, NULL,
#0935             pChild->anMetric[lpLEFT],  pChild->anMetric[lpTOP],
#0936             pChild->anMetric[lpWIDTH], pChild->anMetric[lpHEIGHT],
#0937             SWP_NOZORDER);
#0938          adgASSERT(hdwp);
#0939
#0940          // Invalidation is necessary here because some controls (edit
#0941          // controls in particular) don't repaint correctly under Windows NT
#0942          // when they are moved with DeferWindowPos.
#0943          InvalidateRect(hwndChild, NULL, TRUE);
#0944       }
#0945    }
#0946
#0947    // It is this function call which actually moves all the windows.
#0948    EndDeferWindowPos(hdwp);
#0949
#0950    // Free the allocated list of CHILD structures
#0951    free(pChildList);
#0952    return(fOK);
#0953 }
#0954
#0955 ////////////////////////// End of File //////////////////////////
```

程式列表 3.20　Layout.H

```
#0001 /************************************************************
#0002 Module name: Layout.h
#0003 Written by: Jonathan W. Locke
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Header file with external interface of layout algorithm.
#0006 ************************************************************/
#0007
#0008
#0009 ////////////////// Special Parent Identifier ////////////////////
#0010
#0011
#0012 #define lPARENT 0x10000   // Special CHILD id for parent - this id is
#0013                           // guaranteed not to be used because you
#0014                           // cannot have control ids above 0xffff.
#0015
#0016
#0017 ////////////////////// Layout Actions ///////////////////////
#0018
#0019
#0020 typedef enum {
#0021    lSTRETCH,                  // Metric should be stretched
#0022    lMOVE,                     // Control should be moved
#0023    lVCENTER,                  // Vertically center control/group
#0024    lHCENTER,                  // Horizontally center control/group
#0025    lEND,                      // Special flag for end of list
#0026 } lACTION;
#0027
#0028
#0029 ////////////////////// Layout Part Types ///////////////////////
#0030
#0031
#0032 typedef enum {
#0033    lpLEFT   = 0,              // Left side of control
#0034    lpTOP    = 1,              // Top side of control
#0035    lpRIGHT  = 2,              // Right side of control
#0036    lpBOTTOM = 3,              // Bottom side of control
#0037    lpWIDTH  = 4,              // Width of control
#0038    lpHEIGHT = 5,              // Height of control
#0039    lpGROUP  = 6,              // Group of one or more controls
#0040 } lPART;
#0041
#0042
#0043 ////////////////// Macros for Defining Rules ////////////////////
#0044
```

```
#0045
#0046  //                              Part,      Control id    Percentage
#0047  //                              metric     or first id   of control or
#0048  //     Macro                     or side    in group     last id in group
#0049  //     -----                     ----       -------      ---------
#0050  #define lLEFT(idc)                lpLEFT,    idc,        0
#0051  #define lTOP(idc)                 lpTOP,     idc,         0
#0052  #define lRIGHT(idc)               lpRIGHT,   idc,        0
#0053  #define lBOTTOM(idc)              lpBOTTOM,  idc,         0
#0054  #define lWIDTH(idc)               lpWIDTH,   idc,        100
#0055  #define lHEIGHT(idc)              lpHEIGHT,  idc,        100
#0056  #define lWIDTHOF(idc, percent)    lpWIDTH,   idc,         percent
#0057  #define lHEIGHTOF(idc, percent)   lpHEIGHT,  idc,         percent
#0058  #define lCHILD(idc)               lpGROUP,   idc,         idc
#0059  #define lGROUP(idcFirst, idcLast) lpGROUP,   idcFirst,   idcLast
#0060
#0061
#0062  ////////////////// Definition of an Action //////////////////////
#0063
#0064
#0065  typedef struct {
#0066     union {
#0067        int nPart;               // Name used only for group oriented actions
#0068        int nMetric;             // Name used only for metric oriented actions
#0069        int nSide;               // Name used only for side oriented actions
#0070     };
#0071     union {
#0072        int idc;                // Used for all actions except group actions
#0073        int idcFirst;            // Used for group actions
#0074     };
#0075     union {
#0076        int nPercent;            // Used only for width/height actions
#0077        int idcLast;             // Used for group actions
#0078     };
#0079  } lACTIONINFO;
#0080
#0081
#0082  ////////////////// Definition of a Rule //////////////////////
#0083
#0084
#0085  typedef struct {
#0086     lACTION Action;             // Layout action to take
#0087     lACTIONINFO ActOn;          // Data describing the part to act on
#0088     lACTIONINFO RelTo;          // Data for the part to position relative to
#0089     int nOffset;               // Additional offset (in dialog units)
#0090     UINT fState;               // INTERNAL USE: Rule application state flag
```

```
#0091    int nPixelOffset;            // INTERNAL USE: nOffset converted to pixels
#0092  } RULE;
#0093
#0094
#0095  ////////////////////// Prototype for Layout  ///////////////////
#0096
#0097
#0098  BOOL WINAPI Layout_ComputeLayout (HWND hwndDlg, RULE* pRules);
#0099
#0100
#0101  ////////////////////////// End of File ///////////////////////////
```

程式列表 3.21　DlgSize.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include <windows.h>
#0011
#0012  ////////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include <windows.h>\r\n"
#0030      "\0"
#0031  END
```

```
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  /////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  /////////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_LAYOUT DIALOG DISCARDABLE  -32768, 5, 198, 154
#0049  STYLE WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU |
#0050      WS_THICKFRAME
#0051  CAPTION "Dialog Size"
#0052  FONT 8, "MS Sans Serif"
#0053  BEGIN
#0054      LTEXT          "E1:",IDC_STATIC,7,10,29,8
#0055      EDITTEXT       IDC_E1,7,21,64,12,ES_AUTOHSCROLL
#0056      GROUPBOX       "G1",IDC_G1,83,8,105,32
#0057      PUSHBUTTON     "B3",IDC_B3,112,19,49,14
#0058      CONTROL        "E2:",IDC_STATIC,"Static",SS_LEFTNOWORDWRAP | WS_GROUP,7,
#0059                     42,45,8
#0060      EDITTEXT       IDC_E2,7,53,128,73,ES_AUTOHSCROLL
#0061      PUSHBUTTON     "B1",IDC_B1,32,129,49,14
#0062      PUSHBUTTON     "B2",IDC_B2,74,133,49,14
#0063  END
#0064
#0065
#0066  /////////////////////////////////////////////////////////////////
#0067  //
#0068  // Icon
#0069  //
#0070
#0071  IDI_LAYOUT            ICON    DISCARDABLE     "DlgSize.ico"
#0072
#0073  #ifndef APSTUDIO_INVOKED
#0074  /////////////////////////////////////////////////////////////////
#0075  //
#0076  // Generated from the TEXTINCLUDE 3 resource.
#0077  //
```

```
#0078
#0079
#0080  /////////////////////////////////////////////////////////////////
#0081  #endif    // not APSTUDIO_INVOKED
```

程式列表 3.22　Resource.H

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  /////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include <windows.h>
#0011
#0012  /////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  /////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include <windows.h>\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
```

```
#0039 //////////////////////////////////////////////////////////////
#0040 #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043 //////////////////////////////////////////////////////////////////
#0044 //
#0045 // Dialog
#0046 //
#0047
#0048 IDD_LAYOUT DIALOG DISCARDABLE  -32768, 5, 198, 154
#0049 STYLE WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU |
#0050     WS_THICKFRAME
#0051 CAPTION "Dialog Size"
#0052 FONT 8, "MS Sans Serif"
#0053 BEGIN
#0054     LTEXT          "E1:",IDC_STATIC,7,10,29,8
#0055     EDITTEXT       IDC_E1,7,21,64,12,ES_AUTOHSCROLL
#0056     GROUPBOX       "G1",IDC_G1,83,8,105,32
#0057     PUSHBUTTON     "B3",IDC_B3,112,19,49,14
#0058     CONTROL        "E2:",IDC_STATIC,"Static",SS_LEFTNOWORDWRAP | WS_GROUP,7,
#0059                    42,45,8
#0060     EDITTEXT       IDC_E2,7,53,128,73,ES_AUTOHSCROLL
#0061     PUSHBUTTON     "B1",IDC_B1,32,129,49,14
#0062     PUSHBUTTON     "B2",IDC_B2,74,133,49,14
#0063 END
#0064
#0065
#0066 //////////////////////////////////////////////////////////////////
#0067 //
#0068 // Icon
#0069 //
#0070
#0071 IDI_LAYOUT            ICON    DISCARDABLE    "DlgSize.ico"
#0072
#0073 #ifndef APSTUDIO_INVOKED
#0074 //////////////////////////////////////////////////////////////
#0075 //
#0076 // Generated from the TEXTINCLUDE 3 resource.
#0077 //
#0078
#0079
#0080 //////////////////////////////////////////////////////////////
#0081 #endif    // not APSTUDIO_INVOKED
```

第４章

訂製型控制元件
（Custom Controls）

內建於 Windows 中的控制元件類別，提供給我們許多標準的使用者介面元件。微軟公司賦予這些控制元件許多特性，使它們能適用於任何一個程式。Windows 95 和 Windows NT 3.51 又增加了許多新的控制元件類別。尤有甚者，你可以 subclass 或 superclass 這些控制元件類別（請看第５章），修改其預設的外觀和行爲。然而當 Windows 所提供的控制元件或甚至經過你的修改之後，都還不能符合需要，這時候我們只好自行設計一些控制元件來滿足需求。這些訂製型控制元件（custom controls）可使用在對話盒中，或做爲一個獨立視窗。

在這一章中，我將建立兩個訂製型控制元件。第一個控制元件是 legend（圖例），如圖 **4.1** 所示。很熟悉吧！其設計是用來表示在地圖、圖形或圖表中各部份圖樣的意義。這個 legend 控制元件在其文字字串的左邊會有一個可大可小的方盒，此方盒可依某支畫刷的式樣來填滿。方盒並可選擇是否要邊框 -- 框線由一個圖素（pixel）構成。

第二個控制元件是 bargraph（長條圖）控制元件，如圖 **4.2** 所示。當你希望將一些數值以圖形表示時，每個數值可以一根長桿表示。與 legend 控制元件不同的是，bargraph 控制元件可以接受滑鼠與鍵盤的輸入。當使用者以鍵盤或滑鼠選擇了一根長桿，bargraph 會發出一個通告訊息（notification message），告訴父視窗你選擇了哪一根長桿。而後，

父視窗會顯示該長桿的更多資訊，做爲回應。

圖 4.1　Legend 控制元件

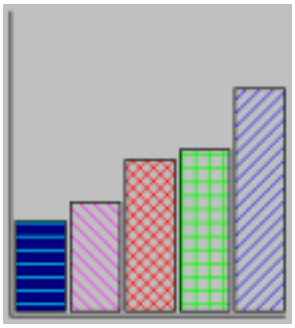


圖 4.2　Brgraph 控制元件

訂製型控制元件的設計規則

設計訂製型控制元件和設計一個視窗類別的動作相當類似。然而，前者比較稍微困難些，因爲大部份控制元件必須設計得可以同時存在於數個程式執行實體（instances）當中。

設計一個控制元件，必需有足夠的彈性以因應未來需求。當你開始設計它時，你可能會說，「只要這個訂製型控制元件能滿足我目前的需求即可」。但是你將會發現，爲你的控制元件添加新的功能，同時也會提升其它已使用此控制元件的程式的功能。但是也要注意，別過度設計，涵蓋了一些根本用不上的功能。最好是從最簡單的設計開始，根據需求加上其它功能。不要僅爲了某個程式就增加功能到控制元件身上，那似乎很不經濟，倒不如使用 subclassing 和 superclassing 的方式來完成比較好。

有一件事你必須謹記在心。設計訂製型控制元件時，不應使用全域變數或靜態變數。因爲它不像一般程式的視窗，它的產生總是數以打計。對於一般程式視窗，如果同一個程

式執行了數個實體（instances），每一實體都有它自己的位址空間，其中包含全域變數和靜態變數。然而控制元件不是這樣。一個程式可能產生同一類別的數個控制元件實體，全部都在相一個位址空間中執行，共享相同的全域變數和靜態變數。由於變數只有一份，卻被所有控制元件共享，所以改變這些變數值將會影響此類別的所有控制元件，而這通常不是我們所希望的。舉個例子，如果你產生數個 edit 控制元件，每一個都把其文字內容儲存在全域變數中，那麼當使用者在其中一個 edit 視窗中輸入文字，所有其他 edit 視窗（被同一執行緒產生者）會顯示出相同的文字。

設計訂製型控制元件必須清楚定義一個程式介面（programmatic interface），詳細描述控制元件的所有功能，以及程式設計師如何管理控制元件的運作。設計程式介面須包含一連串的 #define 敘述，放置在控制元件的表頭檔中。程式欲使用此一控制元件，只需含入這個表頭檔即可。此檔除了 #define 某些數值，也包含了函式原型的宣告、結構的定義、typedef、巨集、訊息 APIs 以及訊息剖析函式（用來輔助你使用控制元件）。一般來說，表頭檔不應包含控制元件的實作碼。這些細節可能會迷惑一些程式員。

剖析訂製型控制元件

視窗風格（Window Style）

每一個訂製型控制元件的視窗類別可以指定一些不同的視窗風格，好讓程式員可以對此控制元件的行爲和外觀作調整。視窗風格在 CreateWindowEx 函式的 dwStyle 參數中設定。

視窗風格的較高字組用來定義一般的視窗類別風格，是系統定義的旗標位元，諸如 WS_BORDER、WS_VISIBLE 和 WS_POPUP 等等。較低字組用來指定與控制元件有關的類別風格，例如 BUTTON 類別是以 BS_ 開頭，LISTBOX 類別是以 LBS_ 開頭。當你設計訂製型控制元件，你也可以自行定義一些風格位元，設定在較低字組，用以改變控制元件的外觀和行爲。

自定視窗訊息（Custom Window Messages）

每一個訂製型控制元件都可能定義一些與其視窗類別有關的專屬訊息。第一個與類別有關的專屬訊息應該被定義爲 WM_USER+0，下一個訊息應該定義爲 WM_USER+1，依此類推。就像視窗風格一樣，這些訊息應被定義在控制元件的表頭檔中。

訊息的說明一定要清楚，說明每一個訊息的作用、傳回值的意義、以及 wParam 和 lParam 的值。例如，listbox 類別中的 LB_DELETESTRING 訊息將根據 wParam 參數表示欲刪除哪一個字串，而其 lParam 參數未被使用。這個訊息的傳回值表示還剩下多少個字串，如果發生錯誤則傳回 LB_ERR。如果我們把這些資訊寫在表頭檔中，程式設計者就可以輕易地使用此控制元件。

父視窗的通告訊息（Parent Notification）

程式介面的最後一個部份，就是一旦有某個動作發生時，控制元件必須送出一個通告訊息（notification message）給其父視窗。系統定義了兩個通告訊息，最常用的是 WM_COMMAND。許多 Windows 內建控制元件皆是使用 WM_COMMAND 來通知其父視窗。舉個例子，當你按下一個按鈕，內建的 button 類別會送出 WM_COMMAND 訊息給它的父視窗，同帶著一個 BN_CLICKED 通告碼（notification code）。系統定義的另一個通告訊息是 WM_NOTIFY，它比 WM_COMMAND 更具彈性，因爲它可以送出更多資訊給其父視窗，而不單單只是通告碼。事實上，你可以送任何你想要的資訊。雖然如此，大部份程式設計者都還是喜歡使用 WM_COMMAND，它雖然簡單卻且已能夠滿足需求。

WM_COMMAND

爲了使用 WM_COMMAND，你必須在你的控制元件表頭檔中定義一些通告碼。每一個碼是 16 位元，必須是獨一無二且未被使用過的數值。當你的控制元件送出 WM_COMMAND 給其父視窗，其 wParam 的低字組爲控制元件 ID，高字組則爲控制元件的通告碼；lParam 內含控制元件的視窗代碼。

你可以這樣送出 WM_COMMAND 訊息：

```
void NotifyParent (HWND hwndControl, WORD wNotifyCode) {

   SendMessage(GetParent(hwndControl), WM_COMMAND,
      MAKEWPARAM(GetWindowID(hwndcontrol), wNotifyCode),
      (LPARAM) hwndControl);
}
```

也可以使用 WindowsX.H 表頭檔中的 FORWARD_WM_COMMAND 巨集替換之：

```
void NotifyParent (HWND hwndControl, WORD wNotifyCode) {

   FORWARD_WM_COMMAND(GetParent(hwndControl),
      GetWindowID(hwndcontrol), hwndcontrol, wNotifyCode, SendMessage);
}
```

這兩個函式都會以適當參數送出 WM_COMMAND 給控制元件的父視窗，。在這兩個例子中，NotifyParent 沒有傳回值（是個 void），因爲 WM_COMMAND 訊息的傳回值沒有意義。事實上，FORWARD_WM_COMMAND 訊息剖析函式將其結果轉型爲 void，所以不會有傳回值。如果你想要取得一些回應資訊，就應該使用 WM_NOTIFY。

WM_NOTIFY

假如你需要的僅是一個通告碼，使用 WM_COMMAND 非常恰當。但是有時你需要取得更多資訊，那就必須使用 WM_NOTIFY（注意不要和 WM_PARENT_NOTIFY 混淆），其 wParam 參數表示送出此通告訊息之控制元件的 ID，lParam 參數可以是任何複雜結構的位址，此結構的唯一要求是，第一個欄位必須是 NMHDR 結構。NMHDR 結構定義在 WinUser.H 表頭檔中，內容如下：

```
typedef struct tagNMHDR
   HWND hwndFrom;  // Window handle of the control that sent the notification
   UINT idFrom;    // Identifier of the control that sent the notification
   UINT code;      // Notification code (user defined or NM_* from CommCtrl.H
} NMHDR;
typedef NMHDR FAR * LPNMHDR;
```

其中的 hwndFrom 和 idFrom 欄位分別代表視窗代碼和視窗 ID，code 欄位用來區分使用者定義的通告碼或是定義在 CommCtrl.H 表頭檔中的一般通告碼（其數值在 WM_FIRST 和 WM_LAST 之間，如下所示）。每一個控制元件可以定義其專屬的通告碼。

```
// Generic WM_NOTIFY notification codes

#define NM_OUTOFMEMORY        (NM_FIRST-1)
#define NM_CLICK              (NM_FIRST-2)
#define NM_DBLCLK             (NM_FIRST-3)
#define NM_RETURN             (NM_FIRST-4)
#define NM_RCLICK             (NM_FIRST-5)
#define NM_RDBLCLK            (NM_FIRST-6)
#define NM_SETFOCUS           (NM_FIRST-7)
#define NM_KILLFOCUS          (NM_FIRST-8)

// WM_NOTIFY codes (NMHDR.code values) are not required
// to be in seperate ranges. but establishing these ranges makes
// validation and debugging easier.

#define NM_FIRST              (0U-  0U)      // generic to all controls
#define NM_LAST               (0U- 99U)
```

你可以自由使用這些一般性通告碼，或是自由定義屬於你自己的通告碼。自行定義的通告碼並沒有數值範圍的限制，但由於 CommCtrl.H 表頭檔爲自己使用了最高數值[1]，所以你應該將你定義的通告碼數值再往上加一（即 NM_LAST+1），這樣一來你的數值便獨一無二，也容易被除錯器辨識出來。

注意，結構中的 idField 欄位的意義和 wParam 參數的意義相同。說明文件中並沒有明白告知原因。然而，父視窗處理通告訊息時，可能在某些地方比較喜歡使用某個變數（勝於另一個變數）。如果處理常式要傳遞一個 NMHDR 結構指標給另一個函式，則在該函式中可以使用 idField，比使用 wParam 來得方便，可以避免額外的語法以及指標的取值

[1] CommCtrl.H 表頭檔利用一個無符號（unsigned）0U 值依序遞減來定義通告碼的值。例如 NM_OUTOFMEMORY 被定義爲 (NM_FIRST-1)，表示 0U-1 或 0xFFFFFFFF，NM_CLICK 被定義爲 (NM_FIRST-2)，表示 0U-2 或 0xFFFFFFFE。

（dereference）。由於你不能預測控制元件的父視窗會使用那一個值，所以你必須將這兩個變數都設爲控制元件的 ID。

注意，你應該“send” WM_NOTIFY 訊息而不是“post”它。如果你用的是 SendMessage 函式，你的通告結構（notification structure）可以宣告在堆疊之中，並將 lParam 設爲指向該結構的指標。由於 SendMessage 是一個同步函式，所以它會等到訊息處理完畢後才返回。如果你將 lParam 設爲指向堆疊中的一個變數，並呼叫 PostMessag，你就犯下了一個嚴重錯誤。由於 PostMessage 是一個非同步函式，它會爲你將訊息放到佇列中，然後立刻返回。這個被“posted”的訊息在你的函式回返之前都不可能被處理[2]，而當你的函式回返，堆疊中的結構將被摧毀，前述訊息（現在位於佇列之中）的 lParam 參數將會變成一個無效指標。

爲了回應 WM_NOTIFY，你可以使用訊息剖析函式 HANDLE_WM_NOTIFY，它被定義在 CommCtrl.H 表頭檔中。在你的 OnNotify 處理常式中，只需要簡單地檢查送出通告碼的是哪一個控制元件 ID，然後據此通告碼將 NMHDR 指標轉型爲一個適當型態的結構指標，接下來你便可以自由存取其它被送出的資訊了。和 WM_COMMAND 一樣，WM_NOTIFY 也沒有傳回值，所以我們必須宣告 OnNotify 傳回 void。然而請注意，你還是可以傳回一個數值，只要將你想傳回的值儲存在自行定義的結構中即可。

```
#include <commctrl.h>

//////////////////////////////////////////////////////////////////////

typedef struct {
   NMHDR nmh;
   int nData;
   DWORD dwReturnValue;
} WHATEVER_NOTIFY;

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

[2] 這是真的，除非你將訊息“post”到你的執行緒佇列之中，並在函式回返之前進入一個訊息迴路之中。

```
void Dlg_OnNotify (HWND hwnd, int idCtl, NMHDR* pnmh) {

   WHATEVER_NOTIFY* pwhatever;

   switch (idCtl) {
      case IDC_WHATEVER;
      if (pnmh->code == NM_CLICK) {
         pwhatever = (WHATEVER_NOTIFY*) pnmh;
         pwhatever->dwReturnValue = Dlg_DoWhatever(pwhatever->nData);
      }
      break;
   }
}

////////////////////////////////////////////////////////////////////////

BOOL WINAPI Dlg_Proc (HWND hwnd, UNIT uMsg, WPARAM wParam, LPARAM lParam)
{

   switch (uMsg) {
      .
      .
      .
      adgHANDLE_DLGMSG(hwnd, WM_NOTIFY, Dlg_OnNotify);
   }
   return;
}
```

實作 你的訂製型控制元件

一旦設計好程式介面，並且建立好表頭檔，你必須決定什麼樣的資料（如果有的話）是控制元件內部必須儲存的。舉個例子，每一個 listbox 控制元件都必須記住它有多少筆資料項、哪一筆資料項被選擇了、哪一筆資料項要顯示在最上面…等等等。

爲了儲存這類「每一個控制元件實體必須分開來維護」的資訊，你應該使用視窗的額外位元組（extra bytes）。雖然使用視窗的 properties 也可以，但是使用視窗額外位元組，不論在記憶體消耗量上面還是在執行效率上面，都遠比使用視窗 properties 來得好。在沒有特殊情況之下，當你撰寫一個控制元件類別，應該使用 GWL_USERDATA，那是個長整數值。這個值是特別保留給你的控制元件的使用者（應用程式開發人員）用的。

如果你只需要一小塊儲存空間，可以直接使用控制元件的視窗額外位元組，這個技術既方便又有效率。但是你必須小心，切勿濫用。不要過掛使用額外位元組！如果你註冊一個類別，而它擁有 40 以上的額外位元組，Windows 95 除錯版會發出警告。爲了儲存大量資訊，你應該配置一塊記憶體，並將指向此記憶體區塊的指標存放到額外位元組中。本章的 legend 控制元件因爲比較簡單，使用第一個方法就夠了。Bargraph 控制元件稍微複雜一些，故使用第二種方法。

設計 Legend 控制元件的程式介面

Legend 控制元件是一個「只能顯示」的控制元件，和 Windows 的 static 控制元件極類似。Legend 控制元件並不接受滑鼠或鍵盤的輸入，其外觀是由類別專屬的視窗風格和訊息構成。出現在 legend 方盒右側的文字可從控制元件的視窗文字中取得。

Legend 定義了三個類別專屬的視窗風格：

```
#define LS_SMALLBOX      0x00000000
#define LS_BIGBOX        0x00000001
#define LS_BOXBORDER     0x00000002
```

雖然這些風格可藉由類別專屬訊息來管理，但使用風格位元來構成控制元件的外觀毋寧是比較方便的。但或許有一天這些風格位元不足以設定你的視窗風格（因爲只有 16 種變化），所以你必須想出一個應變方法。類別專屬訊息是一個非常好用的方法，用以設定和取得每一種風格。另一種方法是自行定義一個訊息，用以取得和設定你儲存在視窗額外位元組中的視窗風格位元。四個額外位元組足以儲存 32 種型態，一般來說足夠了。

前兩個 legend 風格允許使用者設定 legend 的方盒大小。第三個風格用來設定此方盒是否有邊框。第四個風格由 LB_BIGBOX 和 LS_BORDER 組成。這四種 legend 型態如**圖 4.3** 所示。

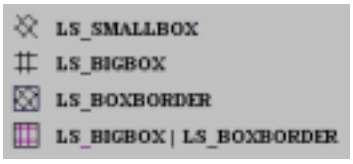


圖 4.3　Legend 視窗型態（譯註：原書 #234 頁的圖 4.3 有誤，其中的 LS_BORDER 應改爲 LS_BOXBORDER，如上）

雖然大部份控制元件的風格都是以單一位元來表示，你仍然可以組合各個位元，表示另一種風格。Windows 的 button 類別就是這麼做的：

```
#define BS_PUSHBUTTON          0x00000000L
#define BS_DEFPUSHBUTTON       0x00000001L
#define BS_CHECKBOX            0x00000002L
#define BS_AUTOCHECKBOX        0x00000003L
#define BS_RADIOBUTTON         0x00000004L
#define BS_3STATE              0x00000005L
#define BS_AUTO3STATE          0x00000006L
#define BS_GROUPBOX            0x00000007L
#define BS_USERBUTTON          0x00000008L
#define BS_AUTORADIOBUTTON     0x00000009L
#define BS_OWNERDRAW           0x0000000BL
```

Windows 可以藉由一個遮罩值 0x0000000F，與 button 的風格位元值做 AND 動作，於是取出 button 的風格。這個動作的結果將會是上述所列的 BS_* 值之一。

Legend 定義了六個類別專屬訊息。類別專屬訊息必須被定義在 WM_USER ~ 0x7FFF 範圍之內。在 WinUser.H 表頭檔中，WM_USER 被定義爲 0x0400。最前面兩個訊息是 accessor 訊息，允許程式員取得和設定一個用來填滿 legend 方盒的畫刷：

```
// Purpose: Sets the brush used to fill legend's box. The brush is used only by
//          the control. It is owned by the caller. The caller must create
//          and destroy the brush.
// wParam:  HBRUSH hbrNew - New brush
// lParam:  N/A
// Returns: void
#define LM_SETBOXBRUSH    (WM_USER + 0)
```

```
// Purpose: Returns HBRUSH being used to fill legend's box.
// wParam:  N/A
// lParam:  N/A
// Returns: HBRUSH - Current box brush
#define LM_GETBOXBRUSH   (WM_USER + 1)
```

一個應用程式必須爲每一個它所產生的 legend 控制元件做組態動作，告訴它使用哪一支畫刷填滿 legend 方盒。舉個例子，你可以使用藍色斜線（由左上到右下）的式樣來填滿 legend 方盒：

```
HBRUSH hbr = CreateHatchBrush(HS_BDIAGONAL, RGB(0, 0, 255));
SendDlgItemMessage(hdlg, IDC_LEGEND, LM_SETBOXBRUSH, (WPARAM) hbr, 0);
```

應用程式也必須知道 legend 控制元件目前所使用的畫刷式樣，這一點可藉由送出（send）一個 LM_GETBOXBRUSH 訊息來獲得：

```
HBRUSH hbr = (HBRUSH) SendDlgItemMessage(hdlg, LM_GETBOXBRUSH, 0, 0);
```

繪製控制元件時，其它四個 legend 訊息允許程式員取得和設定背景顏色和文字顏色。所有新的訂製型控制元件都應該使用這個方法，而不是試圖提供一個 WM_CTLCOLOR* 介面。WM_CTLCOLOR* 體制由 Windows 內建的控制元件使用，但新的通用控制元件（common controls）使用 accessor 訊息取得和設定顏色值。

```
// Purpose: Sets the background color of a legend control.
// wParam:  N/A
// lParam:  COLORREF crNew - New background color
// Returns: void
#define LM_SETBKCOLOR   (WM_USER + 2)

// Purpose: Gets the background color of a legend control.
// wParam:  N/A
// lParam:  N/A
// Returns: COLORREF - Current background color
#define LM_GETBKCOLOR   (WM_USER + 3)

// Purpose: Sets the text color of a legend control.
// wParam:  N/A
```

```
// lParam:  COLORREF crNew - New text color
// Returns: void
#define LM_SETTEXTCOLOR   (WM_USER + 4)

// Purpose: Gets the text color of a legend control.
// wParam:  N/A
// lParam:  N/A
// Returns: COLORREF - Current text color
#define LM_GETTEXTCOLOR   (WM_USER + 5)
```

除了定義一些訊息，Legend.H 表頭檔中也定義了一組 message API 巨集，以及一些訊息剖析器。這些巨集的定義方法與 WindowsX.H 檔中面對系統訊息的定義方法一樣。儘可能在你的程式中使用這些巨集吧，這將使你的程式碼比較具有可讀性，也比較容易維護。

這個範例到目前爲止示範的是如何將訊息送到對話盒中的 legend 控制元件。訂製型控制元件也可能被產生於對話盒之外 -- 如果使用 CreateWindowEx 函式的話；於是你可以使用 SendMessage 或 SendDlgItemMessage 來操控 legend 控制元件，只需將其第一個參數設爲由 CreateWindowEx 函式傳回的控制元件視窗代碼即可。

撰寫一些 Legend 程式碼

產生 legend 控制元件的第一個步驟是，先撰寫一個註銷和註冊 legend 視窗類別的函式。Legend_RegisterClass 函式利用 RegisterClass 函式來註冊 legend 視窗類別。任何應用程式必須先呼叫 Legend_RegisterClass，才有可能產生 legend 視窗。當程式員再也用不到 legend 視窗類別時，可以呼叫 Legend_UnregisterClass 函式註銷視窗類別，其內部乃是呼叫 UnregisterClass 函式。

```
// ANSI/Unicode declarations for legend's window class name
#define WC_LEGENDA "Legend"
#define WC_LEGENDW L"Legend"
#ifdef UNICODE
#define WC_LEGEND WC_LEGENDW
#else
#define WC_LEGEND WC_LEGENDA
#endif
```

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

// The following structure is for the window extra bytes.
// For more information, see the macros presented in Appendix A.
typedef struct tagLEGEND_WNDEXTRABYTES {
   HFONT hfont;           // Font to use for legend text
   HBRUSH hbr;            // Brush to fill legend's box with
   COLORREF crBk;         // Background color
   COLORREF crText;       // Text color
} LEGEND_WNDEXTRABYTES;

////////////////////////////////////////////////////////////////////////

ATOM WINAPI Legend_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass) {

   WNDCLASSEX wc;

   adgINITSTRUCT(wc, TRUE);
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = Legend_WndProc;
   wc.cbWndExtra    = sizeof(LEGEND_WNDEXTRABYTES);
   wc.hInstance     = hinst;
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
   wc.lpszClassName  = WC_LEGEND;

   if (fGlobalClass)
      wc.style |= CS_GLOBALCLASS;

   return(RegisterClassEx(&wc));
}

////////////////////////////////////////////////////////////////////////

BOOL WINAPI Legend_UnregisterClass (HINSTANCE hinst) {

   return(UnregisterClass(WC_LEGEND, hinst));
}
```

如果 legend 被設計爲一個 DLL，你可能會希望當此 DLL 被載入到行程位址空間時，能夠自動註冊 legend 的視窗類別。當 DllMain 函式被呼叫時其 fdwReason 參數如果是 DLL_PROCESS_ATTACH，表示這是 DLL 第一次映射到行程位址空間中，於是你可以呼叫 Legend_RegisterClass 來註冊 legend 視窗類別。如果 fdwReason 參數是 DLL_PROCESS_DETACH，表示 DLL 即將從行程位址空間中被解除映射，那麼你可以呼

叫 Legend_UnregisterClass 來註銷 legend 視窗類別：

```
BOOL WINAPI DllMain (HINSTANCE hinstDll, DWORD fdwReason, LPVOID lpvReserved)
{

   BOOL fOK = TRUE;
   switch (fdwReason) {

      case DLL_PROCESS_ATTACH:
         fOK = (Legend_RegisterClass(hinstDll, TRUE) != INVALID_ATOM);
         adgASSERT(fOK);
         break;

      case DLL_PROCESS_DETACH:
         Legend_UnregisterClass(hinstDll);
         break;
   }
   return(fOK);
}
```

我們將 Legend_RegisterClass 函式的第二個參數設爲 TRUE，用意是將 legend 的視窗類別註冊爲 CS_GLOBALCLASS 風格。當 CS_GLOBALCLASS 位元設爲 on，同一位址空間中的其他模組才可以使用你的視窗類別。如果你沒有指定 CS_GLOBALCLASS 風格，Windows 會將「做類別註冊動作的那一個 HINSTANCE」和「欲產生視窗實體的那一個 HINSTANCE」拿來比較 -- 每當要產生視窗時都會做一次檢查。如果不一樣，就不能夠產生視窗。如果你設定 CS_GLOBALCLASS 風格，就會關閉此項檢查，於是行程位址空間中的任一模組皆可使用你的視窗類別來產生視窗。

指定 CS_HREDRAW 和 CS_VREDRAW 風格會使得 Windows 在視窗大小改變時將整個視窗視爲無效區（需要重繪的區域）。如果你沒有指定這些風格，當視窗變得比較小時，不會有任何無效區被設定，而當視窗變得比較大時，只有未被其它視窗覆蓋的區域才會被設爲無效區。由於 legend 控制元件希望無論視窗大小如何改變，都必須被全部重繪，所以必須指定這兩種風格。然而請你注意，legend 控制元件就像一般的對話盒控制元件一樣，通常在產生之後就不太會改變大小了。

大部份的訂製型控制元件皆會保存一些內部資訊，所以 WNDCLASSEX 結構中的

cbWndExtra 欄位通常不爲 0。這樣可以爲每一個控制元件實體保留一個內部儲存空間，你可以使用 GetWindowWord / GetWindowLong、SetWindowWord / SetWindowLong 等函式存取該空間。Legend 需要保存一支畫刷，用以填滿 legend 控制元件的方盒、一個字型，用以輸出文字、以及兩個 COLORREF 值，用以指定控制元件的文字顏色和背景顏色。由於每個值都是 32 位元（4 個位元組），所以 legend 控制元件的視窗總共需 16 個額外位元組（extra bytes）。

通常，程式員會定義一些 #define 敘述，表示各項資訊在額外位元組（extra bytes）中的起始偏移位置，以便能夠直接存取各筆資訊：

```
#define CBWNDEXTRA  (16) // Total number of bytes required
#define GWL_FONT    (0)  // Offset of font handle (BYTE 0)
#define GWL_BOXBRUSH(4)  // Offset of brush handle (BYTE 4)
#define GWL_CRBK    (8)  // Offset of background color (BYTE 8)
#define GWL_CRTEXT  (12) //  Offset of text color (BYTE 12)
```

這個方法雖然很好，但我不建議你使用之。因爲自行計算大小和偏移量將會很容易出錯。一旦你計算錯誤，所產生的臭蟲可能很難找出。此外，每當你改變你所儲存的資料，你必須重新調整你的計算，而這是非常不方便的。附錄 A 有一些巨集（定義在 Win95ADG.H）可解除你這方面的困擾，並確保其偏移值計算的正確性。爲了使用這些巨集，我們需要定義一個結構如下（放在 Legend.C 檔中），然後設定 cbWndExtra 爲此結構的大小。當我們要存取這些位元組，可使用 adgGETWINDOWLONG 或 adgSETWINDOWLONG 巨集，自動計算出正確偏移量，並將結果傳給 GetWindowLong 或 SetWindowLong 函式[3]：

```
// The following structure is for the window extra bytes.
// For more information, see the macros presented in Appendix A.
```

[3] 在 32 位元環境中，最好使用 32 位元值來進行運算。然而爲了完整性的考量，另外也有一組巨集用來存取 16 位元視窗的額外位元組內容：adgGet / SETWINDOWWORD。如果你要使用 16 位元值以降低你的控制元件的視窗額外位元組大小，你可能需要使用編譯器的 switch 或 pragma 敘述句來確保你的 WNDEXTRABYTES 結構被包裝成 “16-bit alignment”。

```
typedef struct tagLEGEND_WNDEXTRABYTES {
   HFONT hfont;        // Font to use for legend text
   HBRUSH hbr;         // Brush to fill legend's box with
   COLORREF crBk;      // Background color
   COLORREF crText;    // Text color
} LEGEND_WNDEXTRABYTES;
.
.
.
// Set the font in the window extra bytes.
adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont, hfontNew);
.
.
.
// Get the font from the window extra bytes
HFONT hfontCur = (HFONT) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont);
```

在WNDCLASSEX結構中，最後要注意的一個欄位是 hbrBackground。當你呼叫BeginPaint函式，Windows會送出WM_ERASEBKGND 訊息給視窗函式[4]。通常，視窗函式會將此訊息直接交由 DefWindowProc 函式來處理。假如 hbrBackground 的值是 NULL，DefWindowProc函式不會繪出這視窗的背景，並傳回 0 表示視窗背景沒有被清除。由於在 legend 控制元件的視窗函式中，WM_PAINT 訊息的處理方式是繪出全部視窗，包含其背景，所以如果 DefWindowProc 不清除背景的話，視窗的重繪效率會比較高。再者，如果 DefWindowProc 清除視窗背景，在 WM_PAINT 訊息尚未導至視窗重畫之前，legend視窗會有一短暫時刻是沒有背景的，於是造成閃爍。藉著將所有繪圖動作集中在WM_PAINT 訊息處理常式中，並小心地進行重繪動作，可以避免閃爍現象發生 -- 無論重繪動作是多麼頻繁地出現。

在 legend 類別被 Legend_UnregisterClass 函式註銷之前，應用程式必須確保所有 legend視窗都已被摧毀。

訂製型控制元件的特殊訊息

[4] 除非你呼叫 InvalidateRect 函式並將其 fErase 參數設爲 FALSE。

Legend 的視窗函式 -- Legend_WndProc -- 幾乎和一般的視窗函式沒有兩樣，只除了在處理 WM_SETFONT 和 WM_GETFONT 訊息時。如果 Legend 控制元件接收鍵盤或滑鼠的輸入，它也可能會選擇 WM_GETDLGCODE、WM_SETFOCUS 或 WM_KILLFOCUS 等等訊息的其中一個或多個來處理。本章稍後介紹的 bargraph 控制元件，將示範當你的控制元件接受滑鼠或鍵盤輸入時，應該如何處理這些訊息。

WM_SETFONT 和 WM_GETFONT 允許程式在此時機指定控制元件的字型。當程式送出一個 WM_SETFONT 訊息來設定一個控制元件的字型時，其 wParam 參數代表一個字型代碼。如果程式希望控制元件能被重畫，則 lParam 爲 TRUE，否則爲 FALSE。由於 DefWindowProc 函式並未提供 WM_SETFONT 訊息的預設處理動作，所以我們必須自行設計一個 Legend_OnSetFont 函式，將字形代碼儲存到視窗的額外位元組中：

```
void WINAPI Legend_OnSetFont (HWND hwnd, HFONT hfontNew, BOOL fRedraw) {

   adgASSERT((hfontNew == NULL) || (OBJ_FONT == GetObjectType(hfontNew)));
   adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont, hfontNew);
   if (fRedraw)
      InvalidateRect(hwnd, NULL, FALSE);
}
```

一個應用程式可以藉著送出 WIM_GETFONT 訊息，詢問目前控制元件正在使用哪一種字型，並取得字型代碼。該代碼由 Legend_OnSetFont 函式儲存在視窗的額外位元組之中：

```
HFONT WINAPI Legend_OnGetFont (HWND hwnd) {

   return((HFONT) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont));
}
```

許多控制元件會去處裡另一個特殊訊息：WM_ENABLE。此訊息使一個視窗呈現致能（enable）狀態 -- 不管視窗目前是處於致能（enable）或除能（disable）狀態。當一個視窗處於除能狀態，Windows 不會再送出滑鼠或鍵盤訊息給該視窗。通常，當一個視窗被除能，它會以「灰色狀態（grayed state）」重畫它自己，給使用者一個視覺上的指示，意思是此控制元件目前不可用。

爲了維護控制元件外觀和狀態的一致性，控制元件應該在收到 WM_ENABLE 訊息時呼叫 InvalidateRect 函式來強制重繪自己。之後，當控制元件再收到 WM_PAINT 訊息，會呼叫 IsWindowEnable 函式，看看自己是否爲致能狀態，以決定是否重繪自己。你可以試著修改下一節的 bargraph 控制元件，使它能夠回應 WM_ENABLE 訊息。

畫出一個 Legend

Legend 控制元件的大部份工作都是在處理 WM_PAINT 訊息。繪圖動作以 BeginPaint 函式做爲開始：

```
void Legend_OnPaint (HWND hwnd) {

   PAINTSTRUCT ps;
   .
   .
   .
   // Start painting.
   BeginPaint(hwnd, &ps);
```

然後，我們依據目前的視窗額外位元組（extra bytes）的值設定控制元件的背景顏色和文字顏色：

```
void Legend_OnPaint (HWND hwnd) {
   .
   .
   .
   // Set background and text colors.
   SetBkColor(ps.hdc, Legend_GetBkColor(hwnd));
   SetTextColor(ps.hdc, Legend_GetTextColor(hwnd));
```

這些訊息處理常式會利用附錄 A 所提供的視窗額外位元組處理巨集，取得並設定 legend 控制元件的文字顏色和背景顏色：

```
void WINAPI Legend_OnSetBkColor (HWND hwnd, COLORREF crBkNew) {

   adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crBk, crBkNew);
```

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

////////////////////////////////////////////////////////////////////////

COLORREF WINAPI Legend_OnGetBkColor (HWND hwnd) {

   return((COLORREF) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crBk));
}

////////////////////////////////////////////////////////////////////////

void WINAPI Legend_OnSetTextColor (HWND hwnd, COLORREF crTextNew) {

   adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crText, crTextNew);
   InvalidateRect(hwnd, NULL, TRUE);
}

////////////////////////////////////////////////////////////////////////

COLORREF WINAPI Legend_OnGetTextColor (HWND hwnd) {

   return((COLORREF) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crText));
}
```

現在我們已經取得了文字顏色和背景顏色，接下來必須取得視窗工作區（client area）的大小，然後再利用這些值來計算 legend 控制元件中的方盒大小。首先我們計算此方盒尺寸之最大值，令其高度與視窗工作區同高，而其寬度同其高度。然後根據 LS_BIGBOX 風格位元是否設立來決定如何縮小方盒。如果 LS_BIGBOX 有設立，則將其方盒長寬各縮小 1/6。如果 LS_BIGBOX 沒有設立，則將其方盒長寬各縮小 1/4。接下來我們呼叫 InflateRect 函式將方盒縮小，其第二個和第三個參數傳以負值，表示要做縮小的動作：

```
void Legend_OnPaint (HWND hwnd) {

   RECT rcClient, rcBox;
   int  nBoxSide, nShrinkBox;
   .
   .
   .
   // Calculate the size of our legend box based on the client rect and the
   // window style. If the LS_BIGBOX window style is set, then we shrink the
```

```
   // box area by 1/6th of its size; otherwise, we shrink it by 1/4th. These
   // values were arbitrarily determined to look good through experimentation.
   GetClientRect(hwnd, &rcClient);
   nBoxSide = rcClient.bottom;
   SetRect(&rcBox, 0, 0, nBoxSide, nBoxSide);
   nShrinkBox = -(nBoxSide / (GetWindowStyle(hwnd) & LS_BIGBOX ? 6 : 4));
   InflateRect(&rcBox, nShrinkBox, nShrinkBox);
```

如果 legend 控制元件有 LS_BOXBORDER 風格，我們會畫上方盒邊框，然後將此方盒縮小 1 個圖素（pixel），這麼一來我們填入方盒中的花式才不會覆蓋到邊框：

```
void Legend_OnPaint (HWND hwnd) {

   PAINTSTRUCT ps;
   RECT rcBox;
   .
   .
   .
   // If we have a border around the box, draw it and shrink rcBox by one.
   if (GetWindowStyle(hwnd) & LS_BOXBORDER) {
      HBRUSH hbrFrame = CreateSolidBrush(GetTextColor(ps.hdc));
      FrameRect(ps.hdc, &rcBox, hbrFrame);
      DeleteBrush(hbrFrame);
      InflateRect(&rcBox, -1, -1);
   }
```

現在我們可以準備填充方盒了。首先利用 message API 巨集：Legend_GetBoxBrush，取得畫刷代碼。注意，我們可以直接使用 adgGETWINDOWLONG 巨集取得畫刷代碼，但這不是一個好習慣，因爲這會使我們的碼依賴某些實作上的細節，而這些細節可能是會改變的。最後，我們以 SetBrushOrgEx 函式重設畫刷原點爲（0,0），並使用 PatBlt 函式將方盒填滿。

如果我們沒有設定畫刷原點，可能會導至控制元件「欲重繪的部份」和「不需重繪的部份」的背景畫刷式樣排列混亂。如果你把 Legend_OnPaint 中的 SetBrushOrgEx 那一行標示爲程式註解，再重新編譯並執行 CustCntl.EXE，就可以看到這樣的混亂結果。將主對話盒往下或右移動一或兩個圖素，並拖拉另一個視窗覆蓋到 legend 控制元件的部份區域，使該區域成爲無效區；當控制元件重繪時，只重繪無效區，你會看到 Legend 控制元

件的方盒內，畫刷式樣的排列很混亂。如果你將先前標記起來的 SetBrushOrg 那一行還原並重新編譯，問題就不存在了。

```
HBRUSH WINAPI Legend_OnGetBoxBrush (HWND hwnd) {

   // Return the current brush handle stored in the window extra bytes.
   return((HBRUSH) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hbr));
}

////////////////////////////////////////////////////////////////////////

void Legend_OnPaint (HWND hwnd) {

   PAINTSTRUCT ps;
   RECT rcBox;
   HBRUSH hbrOriginal;
   .
   .
   .
   // Fill the box with the appropriate brush.
   SetBrushOrgEx(ps.hdc, 0, 0, NULL);
   hbrOriginal = SelectBrush(ps.hdc, Legend_GetBoxBrush(hwnd));
   PatBlt(ps.hdc, rcBox.left, rcBox.top, rcBox.right - rcBox.left,
      rcBox.bottom - rcBox.top, PATCOPY);
   SelectBrush(ps.hdc, hbrOriginal);
```

現在方盒已繪製完成，我們可以在 Device Context（DC）的截割區域（clipping region）中排除此一區域。運用這個技巧，我們就可以接下來利用 ExtTextOut 函式繪出 legend 控制元件中的文字，並同時填滿控制元件的背景 -- 這使我們的控制元件不致於閃爍：

```
void Legend_OnPaint (HWND hwnd) {

   RECT rcBox;
   PAINTSTRUCT ps;
   .
   .
   .
   // If we drew a border, restore rcBox to it's original size.
   if (GetWindowStyle(hwnd) & LS_BOXBORDER)
      InflateRect(&rcBox, 1, 1);
```

```
   // Exclude rcBox (everything we have drawn so far) from the clipping region
   // so that we can use ExtTextOut to fill in the control's background and draw
   // the text at the same time without blinking.
   ExcludeClipRect(ps.hdc, rcBox.left, rcBox.top, rcBox.right, rcBox.bottom);
```

現在我們已經準備好輸出控制元件方盒右側的文字了。由於我們不知道文字有多長，所以我們必須動態配置一個緩衝區，用來存放文字。緩衝區大小可以利用GetWindowTextLength函式取得，再加上1，爲的是字串結束位元，然後乘以 sizeof(TCHAR)將字元轉換爲位元組。_alloca函式從堆疊中配置文字緩衝區。任何使用_alloca 函式配置而得的記憶體，當函式回返時會自動被釋放掉。

```
void Legend_OnPaint (HWND hwnd) {

   int nTextLength;
   LPTSTR psz;
   .
   .
   .
   // Allocate a buffer of sufficient size to hold the window text and then
   // call GetWindowText to retrieve it. The _alloca function allocates
   // storage from the stack. This memory is automatically freed when our
   // function returns.
   nTextLength = GetWindowTextLength(hwnd);
   psz = _alloca((nTextLength + 1) * sizeof(TCHAR));
   GetWindowText(hwnd, psz, nTextLength + 1);
```

接下來，我們利用 GetWindowFont 取得視窗字型。事實上那是一個巨集，定義在WindowsX 中：

```
#define GetWindowFont(hwnd) FORWARD_WM_GETFONT((hwnd), SendMessage)
```

這個巨集最終會呼叫 Legend_OnGetFont 函式，取得視窗額外位元組中的字型代碼HFONT。我們以此代碼爲 Device Context 選了一個字型，然後呼叫 GetTextExtentPoint32函式，決定視窗文字的大小（以圖素爲單位）。有了這些資訊便可使用 ExtTextOut 函式將我們的字串輸出至垂直中心點。

SDK 技術文件告訴我們 ExtTextOut 函式的原型如下：

```
BOOL ExtTextOut(hdc, X, Y, fuOptions, lprc, lpszString, cbCount, lpDx)
HDC hdc;            // handle of device context
int X;              // x-coordinate of reference point
int Y;              // y-coordinate of reference point
UINT  fuOptions;    // text-output options
CONST RECT* lprc;   // optional clipping and/or opaquing rectangle
LPCTSTR  lpString;  // Address of string
UINT  cbCount;      // string length
CONST INT* lpDx;    // Address of array of intercharacter spacing values
```

ExtTextOut 函式的第一個參數是 DC 代碼（HDC）。第二和第三個參數分別表示垂直中心點的 x, y 座標，亦即放置字串的地方。第四個參數表示背景顏色，用來填滿“clipping/opaquing”矩形（ETO_OPAQUE），並且不會有文字被輸出在矩形之外（ETO_CLIPPED）。第五個參數表示“clipping/opaquing”矩形位址（本例爲視窗工作區的位址）。第六和第七個參數分別表示欲輸出字串的位址和字串長度。第八個參數是 NULL，表示我們想使用 Windows 預設的字元間距。

```
void Legend_OnPaint (HWND hwnd) {

  PAINTSTRUCT ps;
  RECT rcClient;
  int nBoxSiden, TextLength;
  HFONT hfont, hfontOriginal = NULL;
  SIZE size;
  TCHAR* psz;
  .
  .
  .
  // Draw the window text, vertically centered, to the right of the box.
  // ETO_OPAQUE causes the background to be filled with the current
  // background color at the same time. The arbitrary offset of (nBoxSide / 3)
  // is used to add some padding space between the box and the text string.
  hfont = GetWindowFont(hwnd);
  if (hfont != NULL)
     hfontOriginal = SelectFont(ps.hdc, hfont);
     GetTextExtentPoint32(ps.hdc, psz, nTextLength, &size);
     ExtTextOut(ps.hdc, nBoxSide + (nBoxSide / 3),
        (rcClient.bottom - size.cy) / 2, ETO_OPAQUE | ETO_CLIPPED,
```

```
      &rcClient, psz, nTextLength, NULL);
   if (hfont != NULL)
      SelectFont(ps.hdc, hfontOriginal);
```

在 Legend_OnPaint 函式回返之前，我們必須呼叫 EndPaint，釋放 DC（Device Context）。

設計 Bargraph 控制元件

雖然許多程式都需要顯示長條圖（bargraph），但是 Windows 作業系統並沒有提供長條圖控制元件。

Bargraph 控制元件會在視窗的左側和底邊顯示兩個座標軸，然後顯示一連串的長桿。每一長桿使用一支畫刷，每一長桿表示一個特定的值。除了顯示數值之外，bargraph 控制元件還可以接受滑鼠和鍵盤的輸入，供使用者選擇圖中的任何一根長桿。如果選擇了一根長桿，會有一個通告訊息（notification）產生出來並傳送給其父控制元件，允許父控制元件執行一個動作：顯示此一長桿的相關資訊。

程式員必須告訴 bargraph 控制元件，圖中每一長桿需使用那一種畫刷，以及它所表示的數值。舉個例，應用程式可能選擇一支實心畫刷來填滿第一長桿，並設定此長桿高度爲 100 單位。確定圖中所有長桿的尺度後，程式員必須將整個圖形的高度告訴 bargraph 控制元件。如果第一長桿的高度是 100 單位，其它長桿的高度都是 900 單位，那麼我們可能以 1000 單位來表示整個圖形的高度比較理想。

設計 Bargraph 的程式介面

不同於 legend 控制元件的是，bargraph 控制元件沒有定義任何類別專屬的視窗型態。所以我們設計 bargraph 控制元件的第一步是先定義一些類別專屬的視窗訊息。

```
// Purpose: Sets the array of BAR structures for the control. The brush in each
//          BAR structure is only used by the control. It is owned by the
//          caller. Therefore, the caller must create and destroy the brush
//          handle.
// wParam:  UINT cBars - Number of BAR structures in array
```

```
// lParam:  BAR* pbar - Pointer to first BAR in array
// Returns: void
#define BGM_SETBARS    (WM_USER + 0)

// Purpose: Gets pointer to BAR array for control.
// wParam:  N/A
// lParam:  N/A
// Returns: BAR* - Pointer to first BAR in array
#define BGM_GETBARS    (WM_USER + 1)

// Purpose: Gets number of BAR structures in array.
// wParam:  N/A
// lParam:  N/A
// Returns: UINT - Number of BARs
#define BGM_GETCOUNT    (WM_USER + 2)

// Purpose: Sets the current graph height (in user-defined units).
// wParam:  UINT nHeight - New graph height
// lParam:  N/A
// Returns: void
#define BGM_SETHEIGHT   (WM_USER + 3)

// Purpose: Gets the current graph height (in user-defined units).
// wParam:  N/A
// lParam:  N/A
// Returns: UINT - Current graph height
#define BGM_GETHEIGHT   (WM_USER + 4)

// Purpose: Sets the background color of a bargraph control.
// wParam:  N/A
// lParam:  COLORREF crNew - New background color
// Returns: void
#define BGM_SETBKCOLOR   (WM_USER + 5)

// Purpose: Gets the background color of a bargraph control.
// wParam:  N/A
// lParam:  N/A
// Returns: COLORREF - Current background color
#define BGM_GETBKCOLOR   (WM_USER + 6)

// Purpose: Sets the text color of a bargraph control.
// wParam:  N/A
// lParam:  COLORREF crNew - New text color
// Returns: void
#define BGM_SETTEXTCOLOR  (WM_USER + 7)
```

```
// Purpose: Gets the text color of a bargraph control.
// wParam:  N/A
// lParam:  N/A
// Returns: COLORREF - Current text color
#define BGM_GETTEXTCOLOR   (WM_USER + 8)
```

BGM_SETHEIGHT 和 BGM_GETHEIGHT 允許程式員指定和取得 bargraph 控制元件的高度。預設情況下 bargraph 控制元件會將所有長桿繪製在高度爲 0 到 100 單位內。因此，如果 bargraph 控制元件內的一根長桿高度爲 50 單位，它將佔據整個控制元件高度的 50%，如圖 **4.4**。

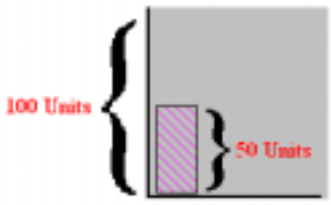


圖 4.4　一個高 50 單位的長條圖在高度爲 100 單位的 bargraph 控制元件中

如果我們想要將 bargraph 控制元件的高度設爲 50 單位，我們可送出如下訊息：

```
SendDlgItemMessage(hDlg, IDC_BARGRAPH, BGM_SETHEIGHT, 50, 0);
```

或者我們可以使用定義在 Bargraph.H 表頭檔中的 message API，設定 bargraph 控制元件的高度：

```
BarGraph_SetHeight(GetDlgItem(hdlg, IDC_BARGRAPH), 50);
```

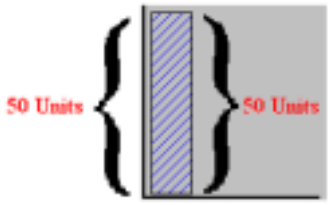


圖 4.5　一個高 50 單位的長條圖在高度爲 50 單位的 bargraph 控制元件中

收到 BGM_SETHEIGHT 訊息後，控制元件會自動重繪。這一次我們的高度爲 50 單位的長桿會抵達控制元件的頂端（如圖 **4.5**）。

BGM_SETBARS 和 BGM_GETBARS 訊息允許程式員取得和設定一個指標，指向 BAR 結構陣列。控制元件繪製長條圖時，會根據 BAR 結構來繪圖。BAR 結構定義在 Bargraph.H 表頭檔中：

```
// BAR structure defines each bar in the bar graph.
typedef struct tagBAR {
   UINT nHeight;       // Height of bar in user-defined units
   HBRUSH hbr;         // Brush to fill bar with
} BAR;
```

BAR 結構中的 nHeight 欄位表示長桿高度（其尺寸從 0 到 bargraph 控制元件的高度）。hbr 爲畫刷代碼，表示欲填入長桿內的式樣。BMG_SETBARS 訊息的 wParam 表示陣列中有多少個 BAR 結構，其 lParam 爲一個指標，指向陣列中的第一個 BAR 結構。

我們的設計是希望由呼叫者管理 BAR 結構陣列的記憶體，以及每一長桿的畫刷代碼。雖然這個方法會帶給呼叫者不少負擔，但確實有不少好處。最大的好處是，bargraph 控制元件不需涉及 BAR 陣列結構及其 HBRUSH 欄位的建立及管理工作。這使得 bargraph 控制元件更具彈性，因爲應用程式設計者能夠自行決定 BAR 結構陣列的建構，以及畫刷的出處。如果程式員要以 cache 方法儲存所有的 HBRUSHs，或不論任何一根長桿都使用同一支畫刷，都沒有問題。此外，你也可以利用 CreateSoildBrush、CreateBrushIndirect、CreatePatternBrush、CreateDIBPatternBrush、CreateDIBPatternBrushPt 和 CreateHatchBrush 函式（更不必說 GetStockObject 函式了）自行設計各式各樣的畫刷。這其中寓含的意義是，如果你對你的介面的「如何使用」做了太多假設，它將會失去彈性。舉個例子，如果 bargraph 控制元件總是使用實心的 RGB 顏色來填滿你的長桿，程式介面當然是單純得多，但是 bargraph 也因此沒有辦法以各種花樣填入長條圖 -- 生活不是太單調了嗎？

只要細心思索，你就會注意到，送給控制元件的通告訊息（notification）幾乎可以毫無限制。舉個例子，每當 bargraph 控制元件要繪製長桿，就送出 BGN_BARPAINT 訊息給其

父視窗，告訴它要畫哪一根長桿，及其矩形範圍；如果父視窗同意，便回應 BGN_BARPAINT，並接管繪圖工作。你可以想像，父視窗能以很多種的方法來回應 BGN_BARPAINT 訊息，使長條圖的顯示別具特色。

雖然我們不實作出 BGN_BARPAINT 訊息，我們卻需要實作另一個通告訊息。當長條圖收到了滑鼠或鍵盤輸入，我們必須使用 WM_NOTIFY 訊息（定義在 WinUser.H 表頭檔中）通知父視窗。爲了完成我們的表頭檔和我們的 bargraph 設計，我們只需定義一個結構來儲存「通告資料」，且它能藉著 WM_NOTIFY 訊息的 lParam 參數傳遞。撰寫 BarGraph_NotifyParent 函式時，我們必須對 WM_NOTIFY 訊息有更多的瞭解，下一節我再詳細介紹之。

```
// The following is the BARGRAPH_NOTIFY structure sent to the parent in the
// lParam of a WM_NOTIFY message. The NMHDR structure can be found in WinUser.H.
typedef struct tagBARGRAPH_NOTIFY {
   NMHDR nmh;         // Notify message header
   UINT uBar;         // Index of bar in graph
} BARGRAPH_NOTIFY;
```

實作出 Bargraph 程式碼

BarGraph_RegisterClass 和 BarGraph_UnregisterClass 函式本質上與 legend 函式所做的事是一樣的。然而，由於我們的 bargraph 控制元件比 legend 控制元件複雜許多，所以我將以不同的方法來儲存各個視窗實體的資料。我並不是在視窗的額外位元組（extra bytes）中直接存放資料，而是存放一個指標，指向一個已配置於記憶體中的結構。這個結構包含控制元件實體（instance）的資料。我定義了一個 THISDATA 結構，並定義了兩個巨集，用來存取視窗額外位元組中的 THISDATA 指標。結構與巨集的定義如下：

```
typedef struct {
   UINT nHeight;              // Bar graph's height (in user-defined units)
   UINT cBars;                // Number of bars in the graph
   BAR* pbar;                 // Array of bars to draw
   UINT nSelectedBar;          // Bar to highlight when focused
   UINT nMouseDownBar;         // Bar where mouse button went down
   UINT nCancelBar;           // Bar to reselect if choice is canceled
```

```
   COLORREF crBk;             // Background color
   COLORREF crText;           // Text color
} THISDATA;

////////////////////////////////////////////////////////////////////////

// The following structure is for the window extra bytes.
// For more information, see the macros presented in appendix A.
typedef struct {
   THISDATA* pthis;
} BARGRAPH_WNDEXTRABYTES;

////////////////////////////////////////////////////////////////////////

// Macros to get and set the THISDATA pointer stored in the window extra bytes
#define GETTHISDATA(hwnd) \
   ((THISDATA*) adgGETWINDOWLONG(hwnd, BARGRAPH_WNDEXTRABYTES, pthis))
#define SETTHISDATA(hwnd, pthisNew) \
   adgSETWINDOWLONG(hwnd, BARGRAPH_WNDEXTRABYTES, pthis, pthisNew);
```

在 BarGraph_OnCreate 函式中，我配置一塊空間用以存放 THISDATA 結構，並使用 SETTHISDATA 巨集將一個指向此結構的指標存放到視窗額外位元組中。如果記憶體配置失敗，BarGraph_OnCreate 函式會傳回 FALSE，而我們的視窗會被摧毀。此後，如果想取得 THISDATA 指標，可以使用 GETTHISDATA 巨集來取得。一旦視窗將被摧毀，Bargraph_OnDestroy 函式會釋放 THISDATA 結構的記憶體。注意，這段程式碼並沒有檢查指標是否爲 NULL[5]，這沒有什麼關係，因爲 ANSI C 說釋放一個 NULL 指標是被允許的（NULL 指標將簡單地被忽略掉）。

```
BOOL BarGraph_OnCreate (HWND hwnd, CREATESTRUCT* pcs) {

   COLORREF crBk;
   HWND hwndParent = GetParent(hwnd);

   // Allocate memory for a THISDATA structure and store a pointer to it in
   // the window extra bytes.
   THISDATA* pthis = malloc(sizeof(THISDATA));
   adgASSERT(pthis != NULL);
```

[5] 在 BarGraph_OnCreate 函式中，如果記憶體配置失敗，THISDATA 指標將是 NULL。

```
   SETTHISDATA(hwnd, pthis);

   // If our control's parent window is a dialog, get our default background
   // color with WM_CTLCOLORDLG, otherwise assume GetSysColor(COLOR_WINDOW).
   if (GetClassWord(hwndParent, GCW_ATOM) == 32770) {

      LOGBRUSH lb;
      HDC hdc = GetDC(hwndParent);
      HBRUSH hbr = (HBRUSH)SendMessage(hwndParent, WM_CTLCOLORDLG,
         (WPARAM)hdc, (LPARAM)hwndParent);
      ReleaseDC(hwndParent, hdc);
      GetObject(hbr, sizeof lb, (LPVOID)&lb);
      crBk = lb.lbColor;
   } else  {
      crBk = GetSysColor(COLOR_WINDOW);
   }

   // If our allocation succeeds, we set some default values
   if (pthis != NULL) {
      BarGraph_SetHeight(hwnd, 100);
      BarGraph_SetBars(hwnd, 0, NULL);
      BarGraph_SetBkColor(hwnd, crBk);
      BarGraph_SetTextColor(hwnd, GetSysColor(COLOR_WINDOWTEXT));
      pthis->nSelectedBar = 0;
   }

   // If our allocation fails, we return FALSE, and the window is destroyed.
   return(pthis != NULL);
}

////////////////////////////////////////////////////////////////////////

void BarGraph_OnDestroy (HWND hwnd) {

   // NOTE: ANSI C states that it is OK to pass NULL to free.
   // This can happen if malloc fails in BarGraph_OnCreate.
   THISDATA* pthis = GETTHISDATA(hwnd);
   free(pthis);
}
```

現在，爲了實作出 BarGraph_OnGetHeight 和 BarGraph_OnSetHeight 兩個存取函式，我們簡單地利用 GETTHISDATA 巨集來取得 THISDATA 指標（位於視窗額外位元組中）：

```
UINT WINAPI BarGraph_OnGetHeight (HWND hwnd) {
```

```
   return(GETTHISDATA(hwnd)->nHeight);
}

////////////////////////////////////////////////////////////////////

void WINAPI BarGraph_OnSetHeight (HWND hwnd, UINT nHeightNew) {

   GETTHISDATA(hwnd)->nHeight = nHeightNew;

   // Our data changed, so we need to force a repaint.
   InvalidateRect(hwnd, NULL, FALSE);
}
```

繪出 Bargraph

繪出 bargraph 控制元件和繪出 legend 控制元件的方法非常類似。主要的不同是在於 legend 的 OnPaint 函式僅需繪出一個方盒，而 bargraph 的 OnPaint 函式需繪出數個矩形。爲了儲存每一根長桿的高度和畫刷，我寫了 BarGraph_OnSetBars、BarGraph_OnGetBars 和 BarGraph_OnGetCount 三個函式。每次我都是使用 GETTHISDATA 巨集來取得控制元件的 THISDATA 結構指標。

```
BAR* WINAPI BarGraph_OnGetBars (HWND hwnd) {

   return(GETTHISDATA(hwnd)->pbar);
}

////////////////////////////////////////////////////////////////////

UINT WINAPI BarGraph_OnGetCount (HWND hwnd) {

   return(GETTHISDATA(hwnd)->cBars);
}

////////////////////////////////////////////////////////////////////

void WINAPI BarGraph_OnSetBars (HWND hwnd, UINT cBars, BAR* pbar) {

   THISDATA* pthis;
   pthis = GETTHISDATA(hwnd);
   pthis->pbar = pbar;
```

```
   pthis->cBars = cBars;
   pthis->nSelectedBar = 0;

   // Our data changed, so we need to force a repaint.
   InvalidateRect(hwnd, NULL, FALSE);
}
```

在 BrGraph_OnPaint 函式中，我利用一個迴圈來繪出每一個 BAR 結構長條圖。爲了繪出一根長桿，我們必須使用 BarGraph_GetBarRect 函式計算其矩形範圍。你同時也可以利用此函式填滿和繪出矩形外框。

```
UINT bar;
RECT rcBar;
for (bar = 0; bar < pthis->cBars; bar++) {
   BarGraph_GetBarRect(hwnd, bar, &rcBar);
   // Draw the bar designated by the rectangle rcbar.
}
```

當然，bargraph 的 OnPaint 函式要比這來得複雜許多，但是這裡卻沒有什麼新東西是你需要學習的，因爲它和 legend 的 OnPaint 函式類似。唯一困難的事情是關於如何撰寫 BarGraph_OnPaint 函式中的 BarGraph_GetBarRect 函式：

```
RECT* BarGraph_GetBarRect (HWND hwnd, UINT n, RECT* prc) {

   RECT rcClient;
   THISDATA* pthis = GETTHISDATA(hwnd);
   int cxClient, cyClient;

   // Check window handle, RECT pointer, and the value of n
   adgASSERT(IsWindow(hwnd));
   adgASSERT(prc != NULL);
   adgASSERT(n < pthis->cBars);

   // Get the size of the client area but don't include the axes lines on the
   // left and bottom.
   GetClientRect(hwnd, &rcClient);
   rcClient.left++;
   rcClient.bottom--;
   cxClient = rcClient.right - rcClient.left;
   cyClient = rcClient.bottom - rcClient.top;
```

```
   // This arbitrary value defines the spacing in pixels between bars
   #define BARSPACING 2

   // Find the left and right sides of bar n.
   #define BARTOCLIENT(uBar) (rcClient.left + \
      (int) (((__int64) cxClient * (uBar)) / pthis->cBars))
   prc->left = BARTOCLIENT(n) + BARSPACING;
   prc->right = BARTOCLIENT(n + 1);
   #undef BARTOCLIENT

   // Set the top based on the height of the bar and the height of the bar
   // graph (in user-defined units). Set the bottom to be the bottom of the
   // client area (minus BARSPACING to leave space between the bottom axis
   // line and the bottom of each bar).
   prc->top = rcClient.bottom -
      (int) (((__int64) cyClient * pthis->pbar[n].nHeight) / pthis->nHeight);
   prc->bottom = rcClient.bottom - BARSPACING;
   #undef BARSPACING

   return(prc);
}
```

這個函式並不是真的很複雜，但的確需要一點點巧思。在開始解決這個問題之前，最好是先一步步分析。首先，我們必須想想如何計算第 n 根長桿矩形的四個點座標。

在做任何計算之前，我們必須取得視窗工作區（client area）的大小，其寬度是 cxClient 個圖素，其高度是 cyClient 個圖素，左上角座標爲 (rcClentt.left, rcClient.top)。這個區域必須稍微縮小一點，我們的長條圖才不會與座標軸重疊。座標軸是先前我們以 BarGraph_OnPaint 函式繪出的左邊線和下邊線。

現在，如果我們的圖中有 pthis->cBars 根長桿，而且要放在寬度爲 cxClient 個圖素的區域內，我們可以使用一個簡單的比例計算，取得第 ubar 根長桿的左座標值（prc->left）：

```
uBar / pthis->cBars = prc->left / cxClient
```

於是 prc->left 爲：

```
prc->left = (uBar / pthis->cBars) * cxClient
```

只要再加上視窗工作區的左值，我們便可以正確得到此一長桿在圖中的左值：

```
prc->left = rcClient.left + ((uBar / pthis->cBars) * cxClient);
```

但是這個作法存在兩個嚴重的瑕疵。第一，我們使用整數運算，將 uBar（其值介於 0 與 (pthis->cBar-1) 之間）除以 pthis->cBar，會得到一個介於 0 與 1 之間的小數值。由於我們使用的是整數運算，所以這個小數值會被四捨五入爲 0！爲避免這個問題發生，我們必需重寫式子，讓乘法先運算：

```
prc->left = rcClient.left + ((cxClient * uBars) / pthis->cBars);
```

這樣的話，當我們做除法運算時，精確度將會提升，因爲除法最後才做。

另一個問題比較複雜。如果 (cxClient*uBar) 太大以至無法以 32 位元整數來表示，會發生什麼事呢？這個溢位（overflow）情況在計算 prc->left 時幾乎不可能發生，但是如果圖形中 y 尺寸是由程式員任何指定呢？

```
prc->top = rcClient.bottom -
   ((cyClient * pthis->pbar[n].nHeight) / pthis->nHeight);
```

如果我們將 cyClient 設爲 2048 個圖素（相當大，但有可能），則 pthis->pbar[n].nHeight 的值會被限制在 (2^32)/2048，也就是 2,097,152 個單位內。對於任何一個不知道 BarGraph_OnSetHeight 函式內的 nHeughtNew 參數被限制在 2,097,152 以內的人而言，這將成爲一個潛在的臭蟲。任何比這個值還大的數值將會導致溢位（overflow），造成 bargraph 的不正確繪圖。

我們可以忽略這個問題，並在文件中詳細說明這個限制。但是這個問題其實很容易就可以解決，只要使用 64 位元整數就可以防止乘法溢位。

```
prc->left = (__int64) cxClient * uBar / pthis->cBars);
prc->top = (__int64) cyClient * pthis->pbar[n].nHeight / pthis->nUnits;
```

BarGraph_GetBarRect 函式的其餘部份沒有什麼特殊的。我將上述這些式子寫成一個巨

集，稱爲 BARTOCLIENT，使用 64 位元整數來求出一根長桿的左值（以圖素爲單位）。然後我利用這個巨集兩次，求出長桿的左值和右值；我還定義了一個 BARSPACING 常數，表示兩根長桿之間的間隙。

```
// This arbitrary value defines the spacing in pixels between bars

#define BARSPACING 2

// Find the left and right sides of bar n.
#define BARTOCLIENT(uBar) (rcClient.left + \
   (int) (((__int64) cxClient * (uBar)) / pthis->cBars))
prc->left = BARTOCLIENT(n) + BARSPACING;
prc->right = BARTOCLIENT(n + 1);
#undef BARTOCLIENT
```

最後，我再使用一個 64 位元值計算出此長桿矩形的頂值和底值（亦即矩形左上角的 y 值和右下角的 y 值）：

```
// Set the top based on the height of the bar and the height of the bar
// graph (in user-defined units). Set the bottom to be the bottom of the
// client area (minus BARSPACING to leave space between the bottom axis
// line and the bottom of each bar).
prc->top = rcClient.bottom -
   (int) (((__int64) cyClient * pthis->pbar[n].nHeight) / pthis->nHeight);
prc->bottom = rcClient.bottom - BARSPACING;
#undef BARSPACING
```

選擇－根式桿

Bargraph 控制元件允許使用者以滑鼠或鍵盤來選擇一根長桿。鍵盤介面要求使用者先以 Tab 鍵將鍵盤焦點轉移到 bargraph 控制元件上，然後再使用左右方向鍵來走訪各個長桿。按下空白鍵或【Return】鍵即可選取目前高亮度的長桿。

滑鼠介面則要求捕捉 WM_LBUTTONDOWN 和 WM_LBUTTONUP 訊息。如果在 bargraph 控制元件中按下滑鼠左鍵，程式會收到 WM_LBUTTONDOWN 訊息。這時候我們首先得判斷滑鼠落於那一根長桿中，並將此值（長桿編號）儲存在此一執行實體

（instance）的 THISDATA 結構中的 nMouseDownBar 和 nSelectedBar 變數（但是得先將原先的 nSelectedBar 內容儲存到「instance 變數」nCancelBar 之中。這些變數隨後會在 BarGraph_OnLButtonUp 和 BarGraph_OnMouseMove 函式中用來處理「使用者取消選取某一根長桿」的情況。爲了讓滑鼠在長桿上選按時有視覺效果，我們將控制元件設爲無效區並取得鍵盤焦點，如此一來 WM_PAINT 發生時我們就可以將它高亮度顯示出來。最後，我們取得滑鼠的捕捉權，於是使用者不論在什麼地方放開滑鼠按鍵，程式都會收到 WM_LBUTTONUP 訊息。

```
void BarGraph_OnLButtonDown (HWND hwnd, BOOL fDoubleClick, int x, int y,
   UINT keyFlags) {

   UINT bar = BarGraph_BarFromPoint(hwnd, x, y);
   if (bar != INVALID_BAR) {

      // If the user clicked on a valid bar, set nMouseDownBar and
      // nSelectedBar to the bar that was clicked on, saving the original
      // nSelectedBar in nCancelBar so that we can restore the old selection
      // if the user cancels. Then we force a repaint and set focus and mouse
      // capture.
      GETTHISDATA(hwnd)->nMouseDownBar = bar;
      GETTHISDATA(hwnd)->nCancelBar = GETTHISDATA(hwnd)->nSelectedBar;
      GETTHISDATA(hwnd)->nSelectedBar = bar;
      InvalidateRect(hwnd, NULL, FALSE);
      SetFocus(hwnd);
      SetCapture(hwnd);
   }
}
```

爲了判斷游標位於哪一根長桿中，我們建立一個 BarGraph_BarFromPoint 函式，其內呼叫 PtInRect 函式，並將圖中每一根長桿的矩形範圍傳入。如果 PtInRect 傳回 TRUE，表示游標在此矩形範圍內，於是我們傳回此長桿的編號（索引）。如果迴圈終止時還沒有傳回矩形編號（索引），我們就傳回 INVALID_BAR，表示失敗。BarGraph_GetBarRect 函式可用來決定外圍矩形，這個函式被 BarGraph_OnPaint 函式在填滿每一根長桿時用到。

```
#define INVALID_BAR ((UINT)-1)

////////////////////////////////////////////////////////////////////////

UINT BarGraph_BarFromPoint (HWND hwnd, int x, int y) {

   UINT bar;
   RECT rcBar;
   THISDATA* pthis = GETTHISDATA(hwnd);
   POINT pt;
   pt.x = x;
   pt.y = y;

   // If point (x, y) is inside a bar, return the index of that bar.
   for (bar = 0; bar < pthis->cBars; bar++)
     if (PtInRect(BarGraph_GetBarRect(hwnd, bar, &rcBar), pt))
        return(bar);

   // Failed.  Point (x, y) is not inside any bar in the graph.
   return(INVALID_BAR);
}
```

因爲我們在 BarGraph_OnLButtonDown 函式中設定了滑鼠的捕捉權，所以不管滑鼠位於哪裡，我們都可以在 BarGraph_OnMouseMove 函式中取得所有的 WM_MOUSEMOVE 訊息。當使用者按下滑鼠左鍵（不要放開），移動滑鼠，我們不是將 nMouseDownBar（目前所選擇的長桿）設爲高亮度，就是將 nCancelBar（前次所選擇的長桿）設爲高亮度 -- 視滑鼠是否仍在原 nMouseDownBar 所指定的長桿中而定。當滑鼠左鍵放開，我們會取消滑鼠捕捉權並檢查滑鼠是否仍在 nMouseDownBar 長桿位置上。假如是，我們就改變「被選擇的長桿」並通知其父視窗說現在已選擇了某一根長桿。

```
void BarGraph_OnLButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {

   // We will only respond to a mouse button up if we set capture previously
   // on a mouse button down.
   if (GetCapture() == hwnd) {

      // Get the bar we were over when the button was released.
      UINT bar = BarGraph_BarFromPoint(hwnd, x, y);

      // Releasing capture generates a WM_CAPTURECHANGED message, which sets
      // nSelectedBar to nCancelBar (canceling the selection by default).
```

```
      ReleaseCapture();

      // Check if the user let up the mouse on the same bar it went down on.
      if (bar == GETTHISDATA(hwnd)->nMouseDownBar) {

         // Change the currently selected bar.
         GETTHISDATA(hwnd)->nSelectedBar = bar;
         InvalidateRect(hwnd, NULL, FALSE);

         // Tell the parent which bar was clicked on.
         BarGraph_NotifyParent(hwnd, NM_CLICK, bar);
      }
   }
}
```

關於滑鼠捕捉權，這裡有一個很小但很重要的觀念：鍵盤焦點和滑鼠捕捉權是兩件完全不相干的事，並且各自獨立。你不需要先獲得一個才能得到另一個。但任何時候如果你利用 GetCapture 得知是你這個視窗（控制元件）擁有滑鼠捕捉權，你應該準備在 OnKillFocus 函式中取消滑鼠捕捉權，因爲如果另一個控制元件取得了鍵盤焦點，而你這個控制元件卻仍然擁有滑鼠捕捉權，會造成使用者的困惑，而且可能滋生臭蟲。如何能在獲得滑鼠捕捉權時放棄鍵盤焦點呢？答案是使用者按下 Tab 鍵。

除了要隨時注意到焦點的失去之外，你也應該注意隨時可能失去滑鼠捕捉權。這裡有一些事件會導致你的執行緒失去滑鼠捕捉權：

- 如果一個訊息窗或對話盒可以冒出來（popup）。
- 如果使用者可以按下 Alt+Tab 鍵切換到另一個視窗。
- 如果使用者可以按下 Ctrl+Esc 鍵帶出【開始】選單（在 Windows NT 之下則是工作列表，task list）。
- 如果你的執行緒所產生的另一個視窗可以呼叫 SetCapture 函式。

雖然這些場景似乎很模糊，但它們的確可能發生。對於一個初學者來說僅能想像其所導致的問題的嚴重性。所幸，取消滑鼠的捕捉權之後，Windows 會送給我們一個

WM_CAPTURECHANGED 訊息[6]。

藉著撰寫以下的 BarGraph_OnKillFocus 函式和 BarGraph_OnCaptureChanged 函式，我們可以防止上述所有問題：

```
void BarGraph_OnKillFocus (HWND hwnd, HWND hwndNewFocus) {

   // If we have the capture, release it.
   if (GetCapture() == hwnd)
      ReleaseCapture();

   // Remove the focus indicator by causing a repaint.
   InvalidateRect(hwnd, NULL, FALSE);
}

////////////////////////////////////////////////////////////////////////

void BarGraph_OnCaptureChanged(HWND hwnd, HWND hwndNewCapture) {

   // If we are losing the capture, cancel the current selection.
   GETTHISDATA(hwnd)->nSelectedBar = GETTHISDATA(hwnd)->nCancelBar;
   InvalidateRect(hwnd, NULL, FALSE);
}
```

除了處理 WM_KILLFOCUS 訊息，我們還需要處理 WM_SETFOCUS 訊息。在 BarGraph_OnSetFocus 函式中唯一要做的事就是將我們的視窗設爲無效區。之後，當呼叫 BarGraph_OnPaint 函式，請注意，GetFocus 函式會傳回控制元件的 hwnd，表示控制元件擁有鍵盤輸入焦點，並且會適當地重繪。

爲了支援滑鼠而撰寫的最後一個函式是 BarGraph_NotifyParent，它僅僅是將訊息包裝到 BARGRAPH_NOTIFY 結構中，並以 WM_NOTIFY 訊息的 lParam 參數形式，傳給其父視窗。BARGRAPH_NOTIFY 結構中的欄位 NMHDR（也是個結構）包裝了控制元件的視窗代碼、控制元件 ID、以及傳遞給 BarGraph_NotifyParent 函式的通告碼（notification code）。我們使用定義在 CommCtrl.H 中的 NM_CLICK 做爲通告碼，表示使用者在控制

[6] Windows NT 3.51 並不支援這個訊息，但 Windows NT 的未來版本將會支援。

元件內按下了滑鼠右鍵。其它唯一需要設定的欄位是 uBar（這就是我們爲什麼不能使用 WM_CMMAND 訊息的原因），它將被設定等於傳進來的參數值。

```
void BarGraph_NotifyParent (HWND hwndCtl, UINT code, UINT uBar) {

   BARGRAPH_NOTIFY bgn;
   adgASSERT(IsWindow(hwndCtl));
   bgn.nmh.hwndFrom = hwndCtl;
   bgn.nmh.idFrom = GetWindowID(hwndCtl);
   bgn.nmh.code = code;
   bgn.uBar = uBar;
   FORWARD_WM_NOTIFY(GetParent(hwndCtl), bgn.nmh.idFrom, &bgn, SendMessage);
}
```

WM_GETDLGCODE

Windows 會定期將 WM_GETDLGCODE 訊息送給對話盒中的視窗。這些訊息的送出是爲了兩個完全不同的原因：

- 查詢控制元件的型態
- 確定控制元件有興趣處理的按鍵

不論哪一種情況，控制元件都必須回應一個或數個 DLGC_* 識別碼（定義在 WinUser.H 中）。如果回應多個識別碼，就以 OR 串接起來。如果 lParam 參數是 0，表示 WM_GETDLGCODE 訊息的送出是爲了查詢控制元件的型態。如果 lParam 不爲 0，WM_GETDLGCODE 訊息就是被 IsDialogMessage 函式送出，用以回應鍵盤訊息；此時的 lParam 參數內含一個指標，指向鍵盤訊息的 MSG 結構。

第一類 DLGC_* 碼（列於**表 4.1**）用來識別控制元件的型態。這些 DLGC_* 位元決定了 Windows 應如何對待這個控制元件（從特殊對話盒行爲的觀點來看）。舉個例，default push button 就有特殊的對話盒行爲，因爲它在回應 WM_GETDLGCODE 訊息時設立了 DLGC_DEFPUSHBUTTON 位元。

表 4.1　DLGC_* 值，用來決定控制元件的型態

識別碼	數值	意義
DLGC_BUTTON	0x2000	此控制元件被對待如同 button。所有 BS_* 型態的 Windows button 類別皆會傳回這個位元，但 BS_GROUPBOX 例外，它傳回的是 DLGC_STATIC，因爲它不是 “checkable”。
DLGC_DEFPUSHBUTTON	0x0010	此控制元件被對待如同 default push button。當 buttons 使用了這一種風格，總是會傳回一個代碼，其中設立了此一位元。
DLGC_UNDEFPUSHBUTTON	0x0020	此控制元件被對待如同一般按鈕。如果你的控制元件沒有被設定爲 default push button，但你想要有 push button 的行爲，你必須設立這個位元。所有使用 BS_PUSHBUTTON 風格的按鈕皆會傳回此一位元。
DLGC_RADIOBUTTON	0x0040	此控制元件被對待如同 radio button。Windows 使用這個位元來判斷群組（group）中的哪些控制元件是 radio button。BS_AUTORADIOBUTTON 和方向鍵的動作皆倚賴這個位元的設立。
DLGC_STATIC	0x0100	控制元件不處理任何鍵盤訊息。當處理助憶碼（mnemonics，有底線的字元，用以在對話盒中從某一控制元件直接跳到另一控制元件上）時，會傳回 DLGC_STATIC。控制元件如果只是簡單地傳回 0，或是它不處理 WM_GETDLGCODE 訊息，就表示它不會去搜尋助憶碼。
DLGC_HASSETSEL	0x0008	控制元件處理 EM_SETSEL 訊息。設立此位元之控制元件收到鍵盤焦點時，會獲得一個 EM_SETSEL 訊息。

第二類 DLGC_* 碼（列於表 **4.2**）用來告訴 Windows 你的控制元件想要處理什麼樣的按鍵。當對話盒成爲作用中的（active）視窗時，如果使用者按下一個按鍵，IsDialogMessage 函式會處理這個按鍵訊息並送出 WM_GETDLGCODE 給目前擁有鍵盤焦點的視窗。這給了控制元件一個機會，讓它告訴 Windows 說不要執行預設處理動作，而直接將此鍵盤訊息送往控制元件。如果 lParam 參數不爲 0，它將指向目前的鍵盤訊息的 MSG 結構。你

可以根據這個 MSG 結構決定要傳回哪一個 DLGC_* 碼（視哪一個按鍵被按下而定）。你也可以無條件的選擇一個或多個 DLGC_* 碼傳回。

舉個例子，如果 edit 控制元件的視窗函式這樣子回應 WM_GETDLGCODE 訊息：

```
UNIT uNormalEditDlgCodes = DLGC_HASSETSEL | DLGC_WANTCHARS |
DLGC_WANTARRORS;
return (uNormalEditDlgCodes | DLGC_WANTTAB);
```

edit 控制元件會繼續如往常般地運作，但是 Tab 鍵訊息會被送往控制元件，而且 Windows 不會將輸入焦點轉換到下一個控制元件手上。這正是我們所期望的。

Bargraph 的鍵盤介面要求我們處理方向鍵、空白鍵和【Return】鍵。假如我們得到一個 WM_KEYDOWN 訊息並判斷那是個空白鍵或【Return】鍵，我們就傳回 DLGC_WANTMESSAGE。此外，我們亦總是向 Windows 要求處理方向鍵。

由於 bargraph 負責處理【Return】鍵，所以當 bargraph 控制元件取得了輸入焦點，Windows 不會對 default push button 做任何處理。你可以在 CustCntl 程式中加入一個 default push button 以驗證這件事情。

```
UINT BarGraph_OnGetDlgCode (HWND hwnd, LPMSG lpmsg) {

   // We want to process the arrow keys unconditionally.
   UINT uRet = DLGC_WANTARROWS;

   // If lpmsg is not NULL, it points to the keyboard message that generated
   // the current WM_GETDLGCODE message.
   if (lpmsg != NULL) {

      // We want to override the default processing if the keyboard message is
      // a WM_KEYDOWN produced by the VK_RETURN or VK_SPACE key.
      if (lpmsg->message == WM_KEYDOWN)
         if ((lpmsg->wParam == VK_RETURN) || (lpmsg->wParam == VK_SPACE))
            uRet |= DLGC_WANTMESSAGE;
   }
   return(uRet);
}
```

表 4.2　DLGC_* 值，用來判斷控制元件想要處理的是哪一種按鍵

識別碼	值	意義
DLGC_WANTARROWS	0x0001	表示控制元件想要處理方向鍵。控制元件會針對 VK_LEFT、VK_RIGHT、VK_UP 和 VK_DOWN 按鍵收到 WM_KEYUP 和 WM_KEYDOWN 訊息。當控制元件取得輸入焦點，Windows 對於方向鍵的預設處理動作會暫時關閉（disabled）。
DLGC_WANTTAB	0x0002	表示控制元件想要處理 Tab 鍵。控制元件針對 VK_TAB 按鍵收到 WM_KEYUP、WM_KEYDOWN 和 WM_CHAR 訊息。當控制元件取得輸入焦點，Windows 對於 Tab 鍵的預設處理動作會暫時關閉（disabled）。這表示你不能以 Tab 鍵離開你的控制元件！
DLGC_WANTALLKEYS	0x0004	表示控制元件想要處理所有按鍵。控制元件會針對所有按鍵收到 WM_KEYUP、WM_KEYDOWN 和 WM_CHAR 訊息。助憶鍵、方向鍵、Tab 鍵、Return 鍵和 Esc 鍵的預設處理動作都被關閉（disabled）。當控制元件取得輸入焦點，所有對話盒行爲將會停止 -- 只有一個例外，那就是 Alt 鍵 + 助憶鍵。
DLGC_WANTMESSAGE	0x0004	表示控制元件想要處理特定訊息。這個識別碼只不過是 DLGC_WANTALLKEYS 的另一個名稱而已。然而，不同名稱是理所當然的，因爲 DLGC_WANTALLKEYS 是在當你想無條件處理所有鍵盤訊息時使用。而 DLGC_WANTMESSAGE 是在當你想有條件處理特定鍵盤訊息時使用。這個「引發 WM_GETDLGCODE 訊息被送出」的鍵盤訊息可以從 OnGetDlgCode 訊息處理常式的 lpmsg 參數獲得。如果這是你想要處理的訊息之一，你可以傳回 DLGC_WANTMESSAGE，於是它就會被送給你。如果你回應一個鍵盤訊息時不曾送出 WM_GETDLGCODE 訊息，則 lpmsg 參數應該設爲 NULL。
DLGC_WANTCHARS	0x0080	表示控制元件想要處理 WM_CHAR 訊息。一個傳回此位元之控制元件會收到 WM_CHAR 訊息，但並不包括任何特殊鍵，像是方向鍵、Return 鍵、Escape 鍵或 Tab 鍵等等。由於這個控制元件會負責處理字元訊息，所以對話盒的助憶碼不起作用。你必須使用 Alt 鍵來處理助憶碼。

現在，我們可以在 BarGraph_OnKey 函式中收到方向鍵了。一旦獲得 VK_LEFT 或 VK_RIGHT 虛擬鍵碼(virtual-key code)，我們就把目前被選取之長桿的編號減 1 或加 1。如果目前選取之長桿位於最邊緣（左或右），我們就讓它有「環繞」效果。如果獲得的是 VK_SPACE 或 VK_RETURN 虛擬鍵碼，我們就將使用者的選擇通知父視窗，使用 BarGraph_NotifyParent 函式（與在 BarGraph_OnLButtonUp 中的處理方式相同）。

```
void BarGraph_OnKey (HWND hwnd, UINT vk, BOOL fDown, int cRepeat, UINT flags)
{

   if (fDown) {

      THISDATA* pthis = GETTHISDATA(hwnd);

      switch (vk) {

         case VK_LEFT:
         case VK_RIGHT:

            // Select the next bar to the left or right and wrap if need be
            pthis->nSelectedBar += ((vk == VK_LEFT) ? (pthis->cBars - 1) : 1);
            pthis->nSelectedBar %= pthis->cBars;
            InvalidateRect(hwnd, NULL, FALSE);
            break;

         case VK_RETURN:
         case VK_SPACE:
            BarGraph_NotifyParent(hwnd, NM_CLICK, pthis->nSelectedBar);
            break;
      }
   }
}
```

訂製型控制元件的優點

CustCntl 程式（CustCntl.EXE）的原始碼顯示於**程式列表 4.1 ~ 4.9**，示範如何使用訂製型控制元件 legend 和 bargraph。此程式的執行畫面如**圖 4.6**。

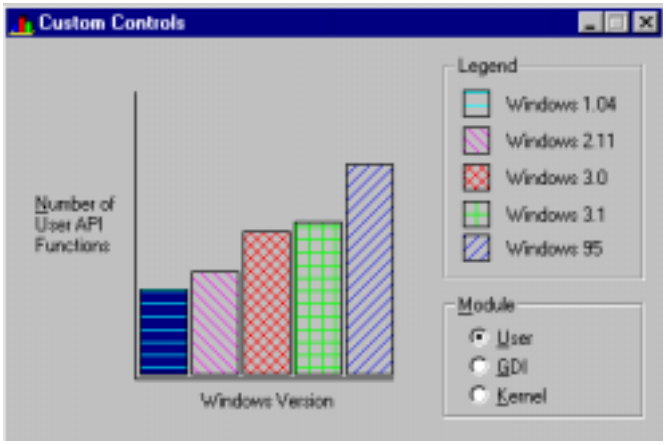


圖 4.6　CustCntl.EXE：訂製型控制元件之應用

如你所見，CustCntl 顯示了 Windows 各版本的 User、Kernel 和 GDI 三大模組所開放的 API 函式總數量。此圖顯示每一個模組提供的函式數量，而不是像 Windows SDK 所說的 API 函式總量。從此圖中你也可以看到 Windows 作業系統的演化過程。

此程式所需檔案，列於表 **4.3**。

表 4.3 建立 CustCntl 程式所需的檔案

檔案	說明
CustCntl.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Legend.C	內含 legend 控制元件的設計。你可以做適當修改以應用於自己的程式中。
Legend.MSG	MsgCrack 的輸入檔，用來產生 Legend.H（附錄 B 有詳細說明）。
Legend.H	內含 legend 控制元件的視窗風格、訊息、訊息 APIs、訊息剖析器及函式原型。任何程式如果用到 legend 控制元件，都必須含入此檔。
BarGraph.C	內含 Bargraph 控制元件的設計。你可以做適當修改以應用於自己的程式中。

BarGraph.MSG	MsgCrack 的輸入檔，用來產生 BarGraph.H（附錄 B 有詳細說明）。
BarGraph.H	內含 bargraph 控制元件的視窗風格、訊息、訊息 APIs、訊息剖析器及函式原型。任何程式如果用到 bargraph 控制元件，都必須含入此檔。
CustCntl.RC	內含主視窗對話盒面板（template）及其圖示（icon）。
Resource.H	內含 CustCntl.RC 檔中所有資源的 ID。
CustCntl.ICO	主視窗圖示（icon）。
CustCntl.MAK	Visual C++ 的 MAK 檔。

CustCntl.C 檔中都是一些例行公事，所以在此我們不詳細討論。首先我們看看 WinMain 函式，它註冊 legend 與 bargraph 兩個控制元件類別，使主對話盒中能夠使用這兩個控制元件：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   // Register the legend and bar graph control classes.
   ATOM atomLegend = Legend_RegisterClass(hinstExe, FALSE);
   ATOM atomBarGraph = BarGraph_RegisterClass(hinstExe, FALSE);

   adgWARNIFUNICODEUNDERWIN95();

   // Check whether class registration succeeded.
   if ((atomLegend != INVALID_ATOM) && (atomBarGraph != INVALID_ATOM)) {

      // Create the application window.
      adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_CUSTCNTL),
         NULL, CustCntl_DlgProc));
   }

   // Unregister window classes.
   if (atomLegend != INVALID_ATOM)
      Legend_UnregisterClass(hinstExe);
   if (atomBarGraph != INVALID_ATOM)
      BarGraph_UnregisterClass(hinstExe);

   return(0);
}
```

程式的其餘部份主要是撰寫一些用於對話盒與 bargraph 和 legend 控制元件之間的溝通函式。在 CustCntl_OnInitDialog 函式中，我們爲 bargraph 控制元件配置一個 BAR 結構陣列，並爲陣列中的每一個 BAR 結構指定畫刷和高度。這些畫刷會被指定給對應的 legend 控制元件。而在 CustCntl_OnDestroy 函式中，我們刪除畫刷並釋放 BAR 陣列。

爲了處理使用者的輸入通告訊息，我爲 bargraph 控制元件所送來的 WM_NOTIFY 訊息寫了一個處理常式。這個函式只是簡單地產生一個訊息盒（message box），顯示使用者所選中之長桿的相關資訊。我也寫了一個 CustCntl_OnCommand 訊息處理常式，當使用者按下對話盒中的 radio button，程式得以改變長條圖的狀態，以反應使用者的選擇（某一個 Windows 模組）。

CustCntl.ICO

程式列表 4.1　CustCntl.C

```
#0001  /****************************************************************
#0002  Module name: CustCntl.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Application to demonstrate use of bar graphs and legends.
#0006  ****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"       /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)  /* Single line comment */
#0013  #include <commctrl.h>
#0014  #include <stdio.h>
#0015  #include <tchar.h>
#0016  #include "Resource.h"
#0017  #include "Legend.h"
#0018  #include "BarGraph.h"
#0019
#0020
#0021  ////////////////////////////////////////////////////////////////
#0022
#0023
#0024  // IDs of first and last legend controls and total number of legends
#0025  #define FIRST_LEGEND  IDC_LEGEND0
#0026  #define LAST_LEGEND   IDC_LEGEND4
```

```
#0027  #define NUM_LEGENDS   (LAST_LEGEND - FIRST_LEGEND + 1)
#0028
#0029
#0030  ////////////////////////////////////////////////////////////////
#0031
#0032
#0033  // The height of our API graph (in units of 1 API function)
#0034  #define APIGRAPH_HEIGHT 750
#0035
#0036
#0037  ////////////////////////////////////////////////////////////////
#0038
#0039
#0040  void CustCntl_SetBars (HWND hwnd, int nIDC) {
#0041
#0042    int i;
#0043    HWND hwndBarGraph = GetDlgItem(hwnd, IDC_BARGRAPH);
#0044    int cBars = BarGraph_GetCount(hwndBarGraph);
#0045    BAR* pbar = BarGraph_GetBars(hwndBarGraph);
#0046
#0047    //   Windows Version     1.04 2.11 3.0  3.1  '95
#0048    //                     ---- ---- ---- ---- ----
#0049    int anAPICallsUser[] =   { 236, 283, 387, 413, 563 };
#0050    int anAPICallsKernel[] = { 103, 141, 192, 233, 707 };
#0051    int anAPICallsGDI[] =    { 183, 213, 235, 283, 332 };
#0052
#0053    if (pbar) {
#0054
#0055       int *pnAPICalls = NULL;
#0056
#0057       switch (nIDC) {
#0058
#0059          case IDC_GDI:
#0060             pnAPICalls = anAPICallsGDI;
#0061             SetDlgItemText(hwnd, IDC_APILABEL,
#0062                __TEXT("&Number of GDI API Functions"));
#0063             break;
#0064
#0065          case IDC_USER:
#0066             pnAPICalls = anAPICallsUser;
#0067             SetDlgItemText(hwnd, IDC_APILABEL,
#0068                __TEXT("&Number of User API Functions"));
#0069             break;
#0070
#0071          case IDC_KERNEL:
#0072             pnAPICalls = anAPICallsKernel;
```

```
#0073            SetDlgItemText(hwnd, IDC_APILABEL,
#0074               __TEXT("&Number of Kernel API Functions"));
#0075            break;
#0076       }
#0077
#0078       // Change the heights of each bar.
#0079       for (i = 0; i < cBars; i++)
#0080          pbar[i].nHeight = pnAPICalls[i];
#0081
#0082       // Set the bar graph's bars to our new bars.
#0083       BarGraph_SetBars(hwndBarGraph, cBars, pbar);
#0084    }
#0085 }
#0086
#0087
#0088 ////////////////////////////////////////////////////////////////
#0089
#0090
#0091 BOOL CustCntl_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0092
#0093    int i;
#0094    HWND hwndBarGraph;
#0095    int cBars;
#0096    BAR* pbar;
#0097
#0098    LOGBRUSH alb[] = {
#0099       { BS_HATCHED, RGB(0, 255, 255), HS_HORIZONTAL },
#0100       { BS_HATCHED, RGB(255, 0, 255), HS_FDIAGONAL },
#0101       { BS_HATCHED, RGB(255, 0, 0),  HS_DIAGCROSS },
#0102       { BS_HATCHED, RGB(0, 255, 0),  HS_CROSS   },
#0103       { BS_HATCHED, RGB(0, 0, 255),  HS_BDIAGONAL }
#0104    };
#0105
#0106    cBars = NUM_LEGENDS;
#0107    pbar = malloc(sizeof(BAR) * cBars);
#0108    adgASSERT(pbar);
#0109    hwndBarGraph = GetDlgItem(hwnd, IDC_BARGRAPH);
#0110    if (pbar != NULL) {
#0111
#0112       // Set brushes of legend controls and bar graph bars. Set heights to
#0113       // zero for now.
#0114       for (i = 0; i < cBars; i++) {
#0115          pbar[i].hbr = CreateBrushIndirect(&alb[i]);
#0116          pbar[i].nHeight = 0;
#0117          if (pbar[i].hbr != NULL) {
#0118             Legend_SetBoxBrush(GetDlgItem(hwnd, FIRST_LEGEND + i),
```

```
#0119              pbar[i].hbr);
#0120         }
#0121      }
#0122
#0123      // Set the array of bars for our API bar graph and specify the height
#0124      // (in units of one API) of the graph.
#0125      BarGraph_SetBars(hwndBarGraph, cBars, pbar);
#0126      BarGraph_SetHeight(hwndBarGraph, APIGRAPH_HEIGHT);
#0127
#0128      // Initially, select the IDC_USER radio button and set the bar graph's
#0129      // bars based on this selection.
#0130      CheckRadioButton(hwnd, IDC_USER, IDC_KERNEL, IDC_USER);
#0131      CustCntl_SetBars(hwnd, IDC_USER);
#0132   } else {
#0133
#0134      // If memory allocation failed, then end the dialog box before it has a
#0135      // chance to show up on the screen.
#0136      BarGraph_SetBars(hwndBarGraph, 0, NULL);
#0137      EndDialog(hwnd, FALSE);
#0138   }
#0139   adgSETDLGICONS(hwnd, IDI_CUSTCNTL, IDI_CUSTCNTL);
#0140   return(TRUE);        // Accept default focus window.
#0141 }
#0142
#0143
#0144 ////////////////////////////////////////////////////////////////
#0145
#0146
#0147 void CustCntl_OnDestroy (HWND hwnd) {
#0148
#0149   int i;
#0150
#0151   // Destroy the brushes assigned to each control.
#0152   for (i = 0; i < NUM_LEGENDS; i++) {
#0153
#0154      HBRUSH hbr = Legend_GetBoxBrush(GetDlgItem(hwnd, FIRST_LEGEND + i));
#0155      if (hbr != NULL)
#0156         DeleteObject(hbr);
#0157   }
#0158
#0159   // Destroy the array of BAR structures allocated in CustCntl_OnInitDialog.
#0160   // It is valid to pass NULL to free, which can occur if the malloc in
#0161   // OnInitDialog fails.
#0162   free(BarGraph_GetBars(GetDlgItem(hwnd, IDC_BARGRAPH)));
#0163 }
#0164
```

```
#0165
#0166 ////////////////////////////////////////////////////////////////
#0167
#0168
#0169 // Simple routine to get rid of accelerator characters in psrc by removing
#0170 // ALL ampersands (including '&&' escapes).
#0171 void RemoveAccelerators (PTSTR psrc) {
#0172
#0173    PTSTR pdest = psrc;
#0174    for (; *psrc; psrc++) {
#0175       if (*psrc != __TEXT('&'))
#0176          *pdest++ = *psrc;
#0177    }
#0178    *pdest = 0;
#0179 }
#0180
#0181
#0182 ////////////////////////////////////////////////////////////////
#0183
#0184
#0185 // For a set of radio buttons with ids between idcFirst and idcLast,
#0186 // inclusive, return the first radio button that is checked. This handy
#0187 // function could be used in any application.
#0188 int GetRadioCheck (HWND hwnd, int idcFirst, int idcLast) {
#0189
#0190    int idc;
#0191    for (idc = idcFirst; idc <= idcLast; idc++) {
#0192       if (IsDlgButtonChecked(hwnd, idc))
#0193          return(idc);
#0194    }
#0195    return(-1);
#0196 }
#0197
#0198
#0199 ////////////////////////////////////////////////////////////////
#0200
#0201
#0202 LRESULT CustCntl_OnNotify (HWND hwnd, int idCtl, NMHDR* pnmh) {
#0203
#0204    BARGRAPH_NOTIFY* pbgn;
#0205
#0206    switch (idCtl) {
#0207
#0208       case IDC_BARGRAPH:
#0209          pbgn = (BARGRAPH_NOTIFY*) pnmh;
#0210          if (pbgn->nmh.code == NM_CLICK) {
```

```
#0211
#0212            TCHAR sz[256];
#0213            TCHAR szWindowsVersion[64];
#0214            TCHAR szModule[64];
#0215            int idcModule;
#0216
#0217            // Get the name of the Windows version that was chosen.
#0218            GetDlgItemText(hwnd, IDC_LEGEND0 + pbgn->uBar, szWindowsVersion,
#0219               adgARRAY_SIZE(szWindowsVersion));
#0220            RemoveAccelerators(szWindowsVersion);
#0221
#0222            // Get the name of the currently selected module.
#0223            szModule[0] = 0;
#0224            idcModule = GetRadioCheck(hwnd, IDC_USER, IDC_KERNEL);
#0225            adgASSERT(idcModule != -1);
#0226            if (idcModule != -1) {
#0227               GetDlgItemText(hwnd, idcModule, szModule,
#0228                  adgARRAY_SIZE(szModule));
#0229               RemoveAccelerators(szModule);
#0230            }
#0231
#0232            // Format and display our message to the user.
#0233            wsprintf(sz,
#0234               __TEXT("You picked bar #%d.\r\n")
#0235               __TEXT("%s %s - %d API calls."),
#0236               pbgn->uBar + 1,
#0237               szWindowsVersion,
#0238               szModule,
#0239               BarGraph_GetBars(GetDlgItem(hwnd, idCtl))[pbgn->uBar]);
#0240            MessageBox(hwnd, sz, __TEXT("Notification"), MB_OK);
#0241         }
#0242         break;
#0243    }
#0244    return(0);
#0245 }
#0246
#0247
#0248 ////////////////////////////////////////////////////////////////
#0249
#0250
#0251 void CustCntl_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0252
#0253    switch (id) {
#0254
#0255       case IDC_KERNEL:
#0256       case IDC_USER:
```

```
#0257       case IDC_GDI:
#0258          CustCntl_SetBars(hwnd, id);
#0259          break;
#0260
#0261       case IDCANCEL:      // Allows dialog box to close
#0262          EndDialog(hwnd, id);
#0263          break;
#0264    }
#0265 }
#0266
#0267
#0268 ////////////////////////////////////////////////////////////////
#0269
#0270
#0271 // This function processes messages sent to CustCntl's dialog box.
#0272 BOOL WINAPI CustCntl_DlgProc (HWND hwnd, UINT uMsg,
#0273    WPARAM wParam, LPARAM lParam) {
#0274
#0275    switch (uMsg) {
#0276
#0277       // Standard Window's messages
#0278       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, CustCntl_OnInitDialog);
#0279       adgHANDLE_DLGMSG(hwnd, WM_DESTROY,    CustCntl_OnDestroy);
#0280       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    CustCntl_OnCommand);
#0281       adgHANDLE_DLGMSG(hwnd, WM_NOTIFY,     CustCntl_OnNotify);
#0282    }
#0283    return(FALSE);        // We didn't process the message.
#0284 }
#0285
#0286
#0287 ////////////////////////////////////////////////////////////////
#0288
#0289
#0290 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0291    LPSTR lpszCmdLine, int nCmdShow) {
#0292
#0293    // Register the legend and bar graph control classes.
#0294    ATOM atomLegend = Legend_RegisterClass(hinstExe, FALSE);
#0295    ATOM atomBarGraph = BarGraph_RegisterClass(hinstExe, FALSE);
#0296
#0297    adgWARNIFUNICODEUNDERWIN95();
#0298
#0299    // Check whether class registration succeeded.
#0300    if ((atomLegend != INVALID_ATOM) && (atomBarGraph != INVALID_ATOM)) {
#0301
#0302       // Create the application window.
```

```
#0303       adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_CUSTCNTL),
#0304          NULL, CustCntl_DlgProc));
#0305    }
#0306
#0307    // Unregister window classes.
#0308    if (atomLegend != INVALID_ATOM)
#0309       Legend_UnregisterClass(hinstExe);
#0310    if (atomBarGraph != INVALID_ATOM)
#0311       BarGraph_UnregisterClass(hinstExe);
#0312
#0313    return(0);
#0314 }
#0315
#0316
#0317 ////////////////////////// End of File ///////////////////////////////
```

程式列表 4.2　Legend.C

```
#0001 /******************************************************************
#0002 Module name: Legend.c
#0003 Written by: Jonathan Locke
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Legend custom control implementation file.
#0006 ******************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"       /* See Appendix A for details */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)  /* Single line comment */
#0013 #include <stdio.h>
#0014 #include <tchar.h>
#0015 #include "Legend.h"
#0016
#0017
#0018 ////////////////////////////////////////////////////////////////////
#0019
#0020
#0021 // The following structure is for the window extra bytes.
#0022 // For more information, see the macros presented in Appendix A.
#0023 typedef struct {
#0024    HFONT hfont;          // Font to use for legend text
#0025    HBRUSH hbr;           // Brush to fill legend's box with
#0026    COLORREF crBk;        // Background color
#0027    COLORREF crText;      // Text color
```

```
#0028 } LEGEND_WNDEXTRABYTES;
#0029
#0030
#0031 ////////////////////////////////////////////////////////////
#0032
#0033
#0034 BOOL Legend_OnCreate (HWND hwnd, CREATESTRUCT* pCreateStruct) {
#0035
#0036    COLORREF crBk;
#0037    HWND hwndParent = GetParent(hwnd);
#0038
#0039    // Initialize default attributes
#0040    SetWindowFont(hwnd, NULL, FALSE);
#0041
#0042    // If our control's parent window is a dialog, get our default background
#0043    // color with WM_CTLCOLORDLG, otherwise assume GetSysColor(COLOR_WINDOW).
#0044    if (GetClassWord(hwndParent, GCW_ATOM) == 32770) {
#0045
#0046       LOGBRUSH lb;
#0047       HDC hdc = GetDC(hwndParent);
#0048       HBRUSH hbr = (HBRUSH)SendMessage(hwndParent, WM_CTLCOLORDLG,
#0049          (WPARAM)hdc, (LPARAM)hwndParent);
#0050       ReleaseDC(hwndParent, hdc);
#0051       GetObject(hbr, sizeof lb, (LPVOID)&lb);
#0052       crBk = lb.lbColor;
#0053    } else  {
#0054       crBk = GetSysColor(COLOR_WINDOW);
#0055    }
#0056
#0057    Legend_SetBkColor(hwnd, crBk);
#0058    Legend_SetBoxBrush(hwnd, GetStockObject(DKGRAY_BRUSH));
#0059    Legend_SetTextColor(hwnd, GetSysColor(COLOR_WINDOWTEXT));
#0060    return(TRUE);
#0061 }
#0062
#0063
#0064 ////////////////////////////////////////////////////////////
#0065
#0066
#0067 void WINAPI Legend_OnSetFont (HWND hwnd, HFONT hfontNew, BOOL fRedraw) {
#0068
#0069    adgASSERT((hfontNew == NULL) || (OBJ_FONT == GetObjectType(hfontNew)));
#0070    adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont, hfontNew);
#0071    if (fRedraw)
#0072       InvalidateRect(hwnd, NULL, FALSE);
#0073 }
```

```
#0074
#0075
#0076 ////////////////////////////////////////////////////////////////
#0077
#0078
#0079 HFONT WINAPI Legend_OnGetFont (HWND hwnd) {
#0080
#0081    return((HFONT) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hfont));
#0082 }
#0083
#0084
#0085 ////////////////////////////////////////////////////////////////
#0086
#0087
#0088 void Legend_OnPaint (HWND hwnd) {
#0089
#0090    RECT rcClient, rcBox;
#0091    PAINTSTRUCT ps;
#0092    HFONT hfont, hfontOriginal = NULL;
#0093    HBRUSH hbrOriginal;
#0094    SIZE size;
#0095    LPTSTR psz;
#0096    int nTextLength, nBoxSide, nShrinkBox;
#0097
#0098    // Start painting.
#0099    BeginPaint(hwnd, &ps);
#0100
#0101    // Set background and text colors.
#0102    SetBkColor(ps.hdc, Legend_GetBkColor(hwnd));
#0103    SetTextColor(ps.hdc, Legend_GetTextColor(hwnd));
#0104
#0105    // Calculate the size of our legend box based on the client rect and the
#0106    // window style. If the LS_BIGBOX window style is set, then we shrink the
#0107    // box area by 1/6th of its size; otherwise, we shrink it by 1/4th. These
#0108    // values were arbitrarily determined to look good through experimentation.
#0109    GetClientRect(hwnd, &rcClient);
#0110    nBoxSide = rcClient.bottom;
#0111    SetRect(&rcBox, 0, 0, nBoxSide, nBoxSide);
#0112    nShrinkBox = -(nBoxSide / (GetWindowStyle(hwnd) & LS_BIGBOX ? 6 : 4));
#0113    InflateRect(&rcBox, nShrinkBox, nShrinkBox);
#0114
#0115    // If we have a border around the box, draw it and shrink rcBox by one.
#0116    if (GetWindowStyle(hwnd) & LS_BOXBORDER) {
#0117       HBRUSH hbrFrame = CreateSolidBrush(GetTextColor(ps.hdc));
#0118       FrameRect(ps.hdc, &rcBox, hbrFrame);
#0119       DeleteBrush(hbrFrame);
```

```
#0120       InflateRect(&rcBox, -1, -1);
#0121    }
#0122
#0123    // Fill the box with the appropriate brush.
#0124    SetBrushOrgEx(ps.hdc, 0, 0, NULL);
#0125    hbrOriginal = SelectBrush(ps.hdc, Legend_GetBoxBrush(hwnd));
#0126    PatBlt(ps.hdc, rcBox.left, rcBox.top, rcBox.right - rcBox.left,
#0127       rcBox.bottom - rcBox.top, PATCOPY);
#0128    SelectBrush(ps.hdc, hbrOriginal);
#0129
#0130    // If we drew a border, restore rcBox to it's original size.
#0131    if (GetWindowStyle(hwnd) & LS_BOXBORDER)
#0132       InflateRect(&rcBox, 1, 1);
#0133
#0134    // Exclude rcBox (everything we have drawn so far) from the clipping region
#0135    // so that we can use ExtTextOut to fill in the control's background and draw
#0136    // the text at the same time without blinking.
#0137    ExcludeClipRect(ps.hdc, rcBox.left, rcBox.top, rcBox.right, rcBox.bottom);
#0138
#0139    // Allocate a buffer of sufficient size to hold the window text and then
#0140    // call GetWindowText to retrieve it. The _alloca function allocates
#0141    // storage from the stack. This memory is automatically freed when our
#0142    // function returns.
#0143    nTextLength = GetWindowTextLength(hwnd);
#0144    psz = _alloca((nTextLength + 1) * sizeof(TCHAR));
#0145    GetWindowText(hwnd, psz, nTextLength + 1);
#0146
#0147    // Draw the window text, vertically centered, to the right of the box.
#0148    // ETO_OPAQUE causes the background to be filled with the current
#0149    // background color at the same time. The arbitrary offset of (nBoxSide / 3)
#0150    // is used to add some padding space between the box and the text string.
#0151    hfont = GetWindowFont(hwnd);
#0152    if (hfont != NULL)
#0153       hfontOriginal = SelectFont(ps.hdc, hfont);
#0154       GetTextExtentPoint32(ps.hdc, psz, nTextLength, &size);
#0155       ExtTextOut(ps.hdc, nBoxSide + (nBoxSide / 3),
#0156          (rcClient.bottom - size.cy) / 2, ETO_OPAQUE | ETO_CLIPPED,
#0157          &rcClient, psz, nTextLength, NULL);
#0158       if (hfont != NULL)
#0159       SelectFont(ps.hdc, hfontOriginal);
#0160
#0161    // Done painting.
#0162    EndPaint(hwnd, &ps);
#0163 }
#0164
#0165
```

```
#0166 ////////////////////////////////////////////////////////////////
#0167
#0168
#0169 void Legend_OnSetText (HWND hwnd, LPCTSTR lpszText) {
#0170
#0171    // We must call DefWindowProc here instead of SetWindowText because
#0172    // SetWindowText sends a WM_SETTEXT message, which results in infinite
#0173    // recursion. DefWindowProc actually sets the text.
#0174    FORWARD_WM_SETTEXT(hwnd, lpszText, DefWindowProc);
#0175
#0176    // Force control to redraw with new text.
#0177    InvalidateRect(hwnd, NULL, FALSE);
#0178 }
#0179
#0180
#0181 ////////////////////////////////////////////////////////////////
#0182
#0183
#0184 void WINAPI Legend_OnSetBoxBrush (HWND hwnd, HBRUSH hbrNew) {
#0185
#0186    // Check window and brush handles for validity.
#0187    adgASSERT(OBJ_BRUSH == GetObjectType(hbrNew));
#0188
#0189    // Set the new brush and trigger a repaint.
#0190    adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hbr, hbrNew);
#0191    InvalidateRect(hwnd, NULL, FALSE);
#0192 }
#0193
#0194
#0195 ////////////////////////////////////////////////////////////////
#0196
#0197
#0198 HBRUSH WINAPI Legend_OnGetBoxBrush (HWND hwnd) {
#0199
#0200    // Return the current brush handle stored in the window extra bytes.
#0201    return((HBRUSH) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, hbr));
#0202 }
#0203
#0204
#0205 ////////////////////////////////////////////////////////////////
#0206
#0207
#0208 void WINAPI Legend_OnSetBkColor (HWND hwnd, COLORREF crBkNew) {
#0209
#0210    adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crBk, crBkNew);
#0211    InvalidateRect(hwnd, NULL, TRUE);
```

```
#0212 }
#0213
#0214
#0215 ////////////////////////////////////////////////////////////////
#0216
#0217
#0218 COLORREF WINAPI Legend_OnGetBkColor (HWND hwnd) {
#0219
#0220    return((COLORREF) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crBk));
#0221 }
#0222
#0223
#0224 ////////////////////////////////////////////////////////////////
#0225
#0226
#0227 void WINAPI Legend_OnSetTextColor (HWND hwnd, COLORREF crTextNew) {
#0228
#0229    adgSETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crText, crTextNew);
#0230    InvalidateRect(hwnd, NULL, TRUE);
#0231 }
#0232
#0233
#0234 ////////////////////////////////////////////////////////////////
#0235
#0236
#0237 COLORREF WINAPI Legend_OnGetTextColor (HWND hwnd) {
#0238
#0239    return((COLORREF) adgGETWINDOWLONG(hwnd, LEGEND_WNDEXTRABYTES, crText));
#0240 }
#0241
#0242
#0243 ////////////////////////////////////////////////////////////////
#0244
#0245
#0246 LRESULT WINAPI Legend_WndProc (HWND hwnd, UINT uMsg,
#0247    WPARAM wParam, LPARAM lParam) {
#0248
#0249    switch (uMsg) {
#0250
#0251       // Standard Windows messages
#0252       HANDLE_MSG(hwnd, WM_CREATE,      Legend_OnCreate);
#0253       HANDLE_MSG(hwnd, WM_PAINT,       Legend_OnPaint);
#0254       HANDLE_MSG(hwnd, WM_SETTEXT,     Legend_OnSetText);
#0255       HANDLE_MSG(hwnd, WM_SETFONT,     Legend_OnSetFont);
#0256       HANDLE_MSG(hwnd, WM_GETFONT,     Legend_OnGetFont);
#0257
```

```
#0258       // Legend control class-specific messages
#0259       HANDLE_MSG(hwnd, LM_SETBOXBRUSH,  Legend_OnSetBoxBrush);
#0260       HANDLE_MSG(hwnd, LM_GETBOXBRUSH,  Legend_OnGetBoxBrush);
#0261       HANDLE_MSG(hwnd, LM_SETBKCOLOR,   Legend_OnSetBkColor);
#0262       HANDLE_MSG(hwnd, LM_GETBKCOLOR,   Legend_OnGetBkColor);
#0263       HANDLE_MSG(hwnd, LM_SETTEXTCOLOR, Legend_OnSetTextColor);
#0264       HANDLE_MSG(hwnd, LM_GETTEXTCOLOR, Legend_OnGetTextColor);
#0265    }
#0266    return(DefWindowProc(hwnd, uMsg, wParam, lParam));
#0267 }
#0268
#0269
#0270 ////////////////////////////////////////////////////////////////
#0271
#0272
#0273 ATOM WINAPI Legend_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass) {
#0274
#0275    WNDCLASSEX wc;
#0276
#0277    adgINITSTRUCT(wc, TRUE);
#0278    wc.style         = CS_HREDRAW | CS_VREDRAW;
#0279    wc.lpfnWndProc    = Legend_WndProc;
#0280    wc.cbWndExtra     = sizeof(LEGEND_WNDEXTRABYTES);
#0281    wc.hInstance      = hinst;
#0282    wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
#0283    wc.lpszClassName  = WC_LEGEND;
#0284
#0285    if (fGlobalClass)
#0286       wc.style |= CS_GLOBALCLASS;
#0287
#0288    return(RegisterClassEx(&wc));
#0289 }
#0290
#0291
#0292 ////////////////////////////////////////////////////////////////
#0293
#0294
#0295 BOOL WINAPI Legend_UnregisterClass (HINSTANCE hinst) {
#0296
#0297    return(UnregisterClass(WC_LEGEND, hinst));
#0298 }
#0299
#0300
#0301 /////////////////////////// End of File ///////////////////////////
```

程式列表 4.3　Legend.MSG

```
#0001  // Module name: Legend.msg
#0002  // Written by: Jonathan Locke
#0003  // Notices: Copyright (c) 1995 Jeffrey Richter
#0004  // Purpose: 'MsgCrack' input file for legend control.
#0005
#0006  MessageBase WM_USER
#0007  MessageClass Legend
#0008
#0009  Message LM_SETBOXBRUSH SetBoxBrush \
#0010   - Sets the brush used to fill legend's box. The brush is only used by the\
#0011   control. It is owned by the caller. The caller must create and destroy\
#0012   the brush.
#0013  wParam HBRUSH hbrNew - New brush
#0014  .
#0015
#0016  Message LM_GETBOXBRUSH GetBoxBrush \
#0017   - Returns HBRUSH being used to fill legend's box.
#0018  Returns HBRUSH - Current box brush
#0019  .
#0020
#0021  Message LM_SETBKCOLOR SetBkColor \
#0022   - Sets the background color of a legend control.
#0023  lParam COLORREF crNew - New background color
#0024  .
#0025
#0026  Message LM_GETBKCOLOR GetBkColor \
#0027   - Gets the background color of a legend control.
#0028  Returns COLORREF - Current background color
#0029  .
#0030
#0031  Message LM_SETTEXTCOLOR SetTextColor \
#0032   - Sets the text color of a legend control.
#0033  lParam COLORREF crNew - New text color
#0034  .
#0035
#0036  Message LM_GETTEXTCOLOR GetTextColor \
#0037   - Gets the text color of a legend control.
#0038  Returns COLORREF - Current text color
#0039  .
```

程式列表 4.4　Legend.H

```
#0001  /************************************************************
#0002  Module name: Legend.h
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Legend custom control header file.
#0006  ************************************************************/
#0007
#0008
#0009  //////////////////////////// Class Name ////////////////////////////
#0010
#0011
#0012  // ANSI/Unicode declarations for legend's window class name
#0013  #define WC_LEGENDA "Legend"
#0014  #define WC_LEGENDW L"Legend"
#0015  #ifdef UNICODE
#0016  #define WC_LEGEND WC_LEGENDW
#0017  #else
#0018  #define WC_LEGEND WC_LEGENDA
#0019  #endif
#0020
#0021
#0022  ////////////////// Class-Specific Window Styles //////////////////
#0023
#0024
#0025  #define LS_SMALLBOX      0x00000000
#0026  #define LS_BIGBOX        0x00000001
#0027  #define LS_BOXBORDER     0x00000002
#0028
#0029
#0030  ////////////////////// Class Registration //////////////////////
#0031
#0032
#0033  // Register and unregister the legend window class.
#0034  ATOM WINAPI Legend_RegisterClass(HINSTANCE hinst, BOOL fGlobalClass);
#0035  BOOL WINAPI Legend_UnregisterClass(HINSTANCE hinst);
#0036
#0037
#0038  ////////////////////// Parent Notification //////////////////////
#0039
#0040
#0041  // Legend controls have no class-specific notification codes at this time.
#0042
#0043
#0044  /////////////// Class-Specific Window Messages ///////////////
```

```
#0045
#0046
#0047  //{{adgMSGCRACK_MESSAGES
#0048
#0049  // Purpose: Sets the brush used to fill legend's box. The brush is used only by
#0050  //         the control. It is owned by the caller. The caller must create
#0051  //         and destroy the brush.
#0052  // wParam:  HBRUSH hbrNew - New brush
#0053  // lParam:  N/A
#0054  // Returns: void
#0055  #define LM_SETBOXBRUSH   (WM_USER + 0)
#0056
#0057  // Purpose: Returns HBRUSH being used to fill legend's box.
#0058  // wParam:  N/A
#0059  // lParam:  N/A
#0060  // Returns: HBRUSH - Current box brush
#0061  #define LM_GETBOXBRUSH   (WM_USER + 1)
#0062
#0063  // Purpose: Sets the background color of a legend control.
#0064  // wParam:  N/A
#0065  // lParam:  COLORREF crNew - New background color
#0066  // Returns: void
#0067  #define LM_SETBKCOLOR   (WM_USER + 2)
#0068
#0069  // Purpose: Gets the background color of a legend control.
#0070  // wParam:  N/A
#0071  // lParam:  N/A
#0072  // Returns: COLORREF - Current background color
#0073  #define LM_GETBKCOLOR   (WM_USER + 3)
#0074
#0075  // Purpose: Sets the text color of a legend control.
#0076  // wParam:  N/A
#0077  // lParam:  COLORREF crNew - New text color
#0078  // Returns: void
#0079  #define LM_SETTEXTCOLOR   (WM_USER + 4)
#0080
#0081  // Purpose: Gets the text color of a legend control.
#0082  // wParam:  N/A
#0083  // lParam:  N/A
#0084  // Returns: COLORREF - Current text color
#0085  #define LM_GETTEXTCOLOR   (WM_USER + 5)
#0086
#0087  //}}adgMSGCRACK_MESSAGES
#0088
#0089
#0090  ////////////////// Legend Control Message APIs ////////////////////
```

```
#0091
#0092
#0093  //{{adgMSGCRACK_APIS
#0094
#0095  #define Legend_SetBoxBrush(hwnd, hbrNew) \
#0096     ((void)SendMessage((hwnd), LM_SETBOXBRUSH, (WPARAM)(DWORD)(hbrNew), 0))
#0097
#0098  #define Legend_GetBoxBrush(hwnd) \
#0099     ((HBRUSH )SendMessage((hwnd), LM_GETBOXBRUSH, 0, 0))
#0100
#0101  #define Legend_SetBkColor(hwnd, crNew) \
#0102     ((void)SendMessage((hwnd), LM_SETBKCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0103
#0104  #define Legend_GetBkColor(hwnd) \
#0105     ((COLORREF )SendMessage((hwnd), LM_GETBKCOLOR, 0, 0))
#0106
#0107  #define Legend_SetTextColor(hwnd, crNew) \
#0108     ((void)SendMessage((hwnd), LM_SETTEXTCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0109
#0110  #define Legend_GetTextColor(hwnd) \
#0111     ((COLORREF )SendMessage((hwnd), LM_GETTEXTCOLOR, 0, 0))
#0112
#0113  //}}adgMSGCRACK_APIS
#0114
#0115
#0116  ////////////////////// Message Crackers //////////////////////
#0117
#0118
#0119  //{{adgMSGCRACK_CRACKERS
#0120
#0121  // void Cls_OnSetBoxBrush(HWND hwnd, HBRUSH hbrNew)
#0122  #define HANDLE_LM_SETBOXBRUSH(hwnd, wParam, lParam, fn) \
#0123     ((fn)((hwnd), (HBRUSH)(wParam)), 0)
#0124  #define FORWARD_LM_SETBOXBRUSH(hwnd, hbrNew, fn) \
#0125     (void)((fn)((hwnd), LM_SETBOXBRUSH, (WPARAM)(DWORD)(hbrNew), 0))
#0126
#0127  // HBRUSH Cls_OnGetBoxBrush(HWND hwnd)
#0128  #define HANDLE_LM_GETBOXBRUSH(hwnd, wParam, lParam, fn) \
#0129     (LRESULT)(fn)((hwnd))
#0130  #define FORWARD_LM_GETBOXBRUSH(hwnd, fn) \
#0131     (HBRUSH )((fn)((hwnd), LM_GETBOXBRUSH, 0, 0))
#0132
#0133  // void Cls_OnSetBkColor(HWND hwnd, COLORREF crNew)
#0134  #define HANDLE_LM_SETBKCOLOR(hwnd, wParam, lParam, fn) \
#0135     ((fn)((hwnd), (COLORREF)(lParam)), 0)
#0136  #define FORWARD_LM_SETBKCOLOR(hwnd, crNew, fn) \
```

```
#0137    (void)((fn)((hwnd), LM_SETBKCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0138
#0139  // COLORREF Cls_OnGetBkColor(HWND hwnd)
#0140  #define HANDLE_LM_GETBKCOLOR(hwnd, wParam, lParam, fn) \
#0141    (LRESULT)(fn)((hwnd))
#0142  #define FORWARD_LM_GETBKCOLOR(hwnd, fn) \
#0143    (COLORREF )((fn)((hwnd), LM_GETBKCOLOR, 0, 0))
#0144
#0145  // void Cls_OnSetTextColor(HWND hwnd, COLORREF crNew)
#0146  #define HANDLE_LM_SETTEXTCOLOR(hwnd, wParam, lParam, fn) \
#0147    ((fn)((hwnd), (COLORREF)(lParam)), 0)
#0148  #define FORWARD_LM_SETTEXTCOLOR(hwnd, crNew, fn) \
#0149    (void)((fn)((hwnd), LM_SETTEXTCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0150
#0151  // COLORREF Cls_OnGetTextColor(HWND hwnd)
#0152  #define HANDLE_LM_GETTEXTCOLOR(hwnd, wParam, lParam, fn) \
#0153    (LRESULT)(fn)((hwnd))
#0154  #define FORWARD_LM_GETTEXTCOLOR(hwnd, fn) \
#0155    (COLORREF )((fn)((hwnd), LM_GETTEXTCOLOR, 0, 0))
#0156
#0157  //}}adgMSGCRACK_CRACKERS
#0158
#0159
#0160  ////////////////////////// End of File //////////////////////////
```

程式列表 4.5　BarGraph.C

```
#0001  /*************************************************************
#0002  Module name: BarGraph.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: BarGraph custom control implementation file.
#0006  *************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include <stdio.h>
#0014  #include <tchar.h>
#0015  #include <commctrl.h>
#0016  #include "BarGraph.h"
#0017
#0018
```

```
#0019 ////////////////////////////////////////////////////////////////
#0020
#0021
#0022 typedef struct {
#0023    UINT nHeight;              // Bar graph's height (in user-defined units)
#0024    UINT cBars;                // Number of bars in the graph
#0025    BAR* pbar;                 // Array of bars to draw
#0026    UINT nSelectedBar;          // Bar to highlight when focused
#0027    UINT nMouseDownBar;         // Bar where mouse button went down
#0028    UINT nCancelBar;           // Bar to reselect if choice is canceled
#0029    COLORREF crBk;             // Background color
#0030    COLORREF crText;           // Text color
#0031 } THISDATA;
#0032
#0033
#0034 ////////////////////////////////////////////////////////////////
#0035
#0036
#0037 // The following structure is for the window extra bytes.
#0038 // For more information, see the macros presented in appendix A.
#0039 typedef struct {
#0040    THISDATA* pthis;
#0041 } BARGRAPH_WNDEXTRABYTES;
#0042
#0043
#0044 ////////////////////////////////////////////////////////////////
#0045
#0046
#0047 // Macros to get and set the THISDATA pointer stored in the window extra bytes
#0048 #define GETTHISDATA(hwnd) \
#0049    ((THISDATA*) adgGETWINDOWLONG(hwnd, BARGRAPH_WNDEXTRABYTES, pthis))
#0050 #define SETTHISDATA(hwnd, pthisNew) \
#0051    adgSETWINDOWLONG(hwnd, BARGRAPH_WNDEXTRABYTES, pthis, pthisNew);
#0052
#0053
#0054 ////////////////////////////////////////////////////////////////
#0055
#0056
#0057 #define INVALID_BAR ((UINT)-1)
#0058
#0059
#0060 ////////////////////////////////////////////////////////////////
#0061
#0062
#0063 BOOL BarGraph_OnCreate (HWND hwnd, CREATESTRUCT* pcs) {
#0064
```

```
#0065    COLORREF crBk;
#0066    HWND hwndParent = GetParent(hwnd);
#0067
#0068    // Allocate memory for a THISDATA structure and store a pointer to it in
#0069    // the window extra bytes.
#0070    THISDATA* pthis = malloc(sizeof(THISDATA));
#0071    adgASSERT(pthis != NULL);
#0072    SETTHISDATA(hwnd, pthis);
#0073
#0074    // If our control's parent window is a dialog, get our default background
#0075    // color with WM_CTLCOLORDLG, otherwise assume GetSysColor(COLOR_WINDOW).
#0076    if (GetClassWord(hwndParent, GCW_ATOM) == 32770) {
#0077
#0078       LOGBRUSH lb;
#0079       HDC hdc = GetDC(hwndParent);
#0080       HBRUSH hbr = (HBRUSH)SendMessage(hwndParent, WM_CTLCOLORDLG,
#0081          (WPARAM)hdc, (LPARAM)hwndParent);
#0082       ReleaseDC(hwndParent, hdc);
#0083       GetObject(hbr, sizeof lb, (LPVOID)&lb);
#0084       crBk = lb.lbColor;
#0085    } else  {
#0086       crBk = GetSysColor(COLOR_WINDOW);
#0087    }
#0088
#0089    // If our allocation succeeds, we set some default values
#0090    if (pthis != NULL) {
#0091       BarGraph_SetHeight(hwnd, 100);
#0092       BarGraph_SetBars(hwnd, 0, NULL);
#0093       BarGraph_SetBkColor(hwnd, crBk);
#0094       BarGraph_SetTextColor(hwnd, GetSysColor(COLOR_WINDOWTEXT));
#0095       pthis->nSelectedBar = 0;
#0096    }
#0097
#0098    // If our allocation fails, we return FALSE, and the window is destroyed.
#0099    return(pthis != NULL);
#0100 }
#0101
#0102
#0103 ////////////////////////////////////////////////////////////////
#0104
#0105
#0106 void BarGraph_OnDestroy (HWND hwnd) {
#0107
#0108    // NOTE: ANSI C states that it is OK to pass NULL to free.
#0109    // This can happen if malloc fails in BarGraph_OnCreate.
#0110    THISDATA* pthis = GETTHISDATA(hwnd);
```

```
#0111    free(pthis);
#0112 }
#0113
#0114
#0115 ////////////////////////////////////////////////////////////////
#0116
#0117
#0118 UINT BarGraph_OnGetDlgCode (HWND hwnd, LPMSG lpmsg) {
#0119
#0120    // We want to process the arrow keys unconditionally.
#0121    UINT uRet = DLGC_WANTARROWS;
#0122
#0123    // If lpmsg is not NULL, it points to the keyboard message that generated
#0124    // the current WM_GETDLGCODE message.
#0125    if (lpmsg != NULL) {
#0126
#0127       // We want to override the default processing if the keyboard message is
#0128       // a WM_KEYDOWN produced by the VK_RETURN or VK_SPACE key.
#0129       if (lpmsg->message == WM_KEYDOWN)
#0130          if ((lpmsg->wParam == VK_RETURN) || (lpmsg->wParam == VK_SPACE))
#0131             uRet |= DLGC_WANTMESSAGE;
#0132    }
#0133    return(uRet);
#0134 }
#0135
#0136
#0137 ////////////////////////////////////////////////////////////////
#0138
#0139
#0140 RECT* BarGraph_GetBarRect (HWND hwnd, UINT n, RECT* prc) {
#0141
#0142    RECT rcClient;
#0143    THISDATA* pthis = GETTHISDATA(hwnd);
#0144    int cxClient, cyClient;
#0145
#0146    // Check window handle, RECT pointer, and the value of n
#0147    adgASSERT(IsWindow(hwnd));
#0148    adgASSERT(prc != NULL);
#0149    adgASSERT(n < pthis->cBars);
#0150
#0151    // Get the size of the client area but don't include the axes lines on the
#0152    // left and bottom.
#0153    GetClientRect(hwnd, &rcClient);
#0154    rcClient.left++;
#0155    rcClient.bottom--;
#0156    cxClient = rcClient.right - rcClient.left;
```

```
#0157    cyClient = rcClient.bottom - rcClient.top;
#0158
#0159    // This arbitrary value defines the spacing in pixels between bars
#0160    #define BARSPACING 2
#0161
#0162    // Find the left and right sides of bar n.
#0163    #define BARTOCLIENT(uBar) (rcClient.left + \
#0164       (int) (((__int64) cxClient * (uBar)) / pthis->cBars))
#0165    prc->left = BARTOCLIENT(n) + BARSPACING;
#0166    prc->right = BARTOCLIENT(n + 1);
#0167    #undef BARTOCLIENT
#0168
#0169    // Set the top based on the height of the bar and the height of the bar
#0170    // graph (in user-defined units). Set the bottom to be the bottom of the
#0171    // client area (minus BARSPACING to leave space between the bottom axis
#0172    // line and the bottom of each bar).
#0173    prc->top = rcClient.bottom -
#0174       (int) (((__int64) cyClient * pthis->pbar[n].nHeight) / pthis->nHeight);
#0175    prc->bottom = rcClient.bottom - BARSPACING;
#0176    #undef BARSPACING
#0177
#0178    return(prc);
#0179 }
#0180
#0181
#0182 ////////////////////////////////////////////////////////////////
#0183
#0184
#0185 UINT BarGraph_BarFromPoint (HWND hwnd, int x, int y) {
#0186
#0187    UINT bar;
#0188    RECT rcBar;
#0189    THISDATA* pthis = GETTHISDATA(hwnd);
#0190    POINT pt;
#0191    pt.x = x;
#0192    pt.y = y;
#0193
#0194    // If point (x, y) is inside a bar, return the index of that bar.
#0195    for (bar = 0; bar < pthis->cBars; bar++)
#0196      if (PtInRect(BarGraph_GetBarRect(hwnd, bar, &rcBar), pt))
#0197         return(bar);
#0198
#0199    // Failed.  Point (x, y) is not inside any bar in the graph.
#0200    return(INVALID_BAR);
#0201 }
#0202
```

```
#0203
#0204 ////////////////////////////////////////////////////////////////
#0205
#0206
#0207 void BarGraph_NotifyParent (HWND hwndCtl, UINT code, UINT uBar) {
#0208
#0209    BARGRAPH_NOTIFY bgn;
#0210    adgASSERT(IsWindow(hwndCtl));
#0211    bgn.nmh.hwndFrom = hwndCtl;
#0212    bgn.nmh.idFrom = GetWindowID(hwndCtl);
#0213    bgn.nmh.code = code;
#0214    bgn.uBar = uBar;
#0215    FORWARD_WM_NOTIFY(GetParent(hwndCtl), bgn.nmh.idFrom, &bgn, SendMessage);
#0216 }
#0217
#0218
#0219 ////////////////////////////////////////////////////////////////
#0220
#0221
#0222 void BarGraph_OnLButtonDown (HWND hwnd, BOOL fDoubleClick, int x, int y,
#0223    UINT keyFlags) {
#0224
#0225    UINT bar = BarGraph_BarFromPoint(hwnd, x, y);
#0226    if (bar != INVALID_BAR) {
#0227
#0228       // If the user clicked on a valid bar, set nMouseDownBar and
#0229       // nSelectedBar to the bar that was clicked on, saving the original
#0230       // nSelectedBar in nCancelBar so that we can restore the old selection
#0231       // if the user cancels. Then we force a repaint and set focus and mouse
#0232       // capture.
#0233       GETTHISDATA(hwnd)->nMouseDownBar = bar;
#0234       GETTHISDATA(hwnd)->nCancelBar = GETTHISDATA(hwnd)->nSelectedBar;
#0235       GETTHISDATA(hwnd)->nSelectedBar = bar;
#0236       InvalidateRect(hwnd, NULL, FALSE);
#0237       SetFocus(hwnd);
#0238       SetCapture(hwnd);
#0239    }
#0240 }
#0241
#0242
#0243 ////////////////////////////////////////////////////////////////
#0244
#0245
#0246 void BarGraph_OnLButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {
#0247
#0248    // We will only respond to a mouse button up if we set capture previously
```

```
#0249    // on a mouse button down.
#0250    if (GetCapture() == hwnd) {
#0251
#0252       // Get the bar we were over when the button was released.
#0253       UINT bar = BarGraph_BarFromPoint(hwnd, x, y);
#0254
#0255       // Releasing capture generates a WM_CAPTURECHANGED message, which sets
#0256       // nSelectedBar to nCancelBar (canceling the selection by default).
#0257       ReleaseCapture();
#0258
#0259       // Check if the user let up the mouse on the same bar it went down on.
#0260       if (bar == GETTHISDATA(hwnd)->nMouseDownBar) {
#0261
#0262          // Change the currently selected bar.
#0263          GETTHISDATA(hwnd)->nSelectedBar = bar;
#0264          InvalidateRect(hwnd, NULL, FALSE);
#0265
#0266          // Tell the parent which bar was clicked on.
#0267          BarGraph_NotifyParent(hwnd, NM_CLICK, bar);
#0268       }
#0269    }
#0270 }
#0271
#0272
#0273 ////////////////////////////////////////////////////////////////
#0274
#0275
#0276 void BarGraph_OnMouseMove (HWND hwnd, int x, int y, UINT keyFlags) {
#0277
#0278    // If the mouse moves while we have capture
#0279    if (GetCapture() == hwnd) {
#0280
#0281       // Get the bar we are currently over.
#0282       UINT bar = BarGraph_BarFromPoint(hwnd, x, y);
#0283       UINT nSelectedBar = GETTHISDATA(hwnd)->nSelectedBar;
#0284
#0285       // Check if the mouse is over the same bar the button went down on.
#0286       if (bar == GETTHISDATA(hwnd)->nMouseDownBar) {
#0287
#0288          // Highlight the bar we are over, if need be, to indicate that
#0289          // releasing the mouse button will choose the current bar.
#0290          if (nSelectedBar != bar) {
#0291             GETTHISDATA(hwnd)->nSelectedBar = bar;
#0292             InvalidateRect(hwnd, NULL, FALSE);
#0293          }
#0294       }
```

```
#0295       else
#0296       {
#0297          // Since the user isn't over the bar where the mouse went down,
#0298          // releasing the mouse button would be a cancel operation. So, we
#0299          // highlight the cancel bar to indicate this, if need be.
#0300          if (nSelectedBar != GETTHISDATA(hwnd)->nCancelBar) {
#0301             GETTHISDATA(hwnd)->nSelectedBar = GETTHISDATA(hwnd)->nCancelBar;
#0302             InvalidateRect(hwnd, NULL, FALSE);
#0303          }
#0304       }
#0305    }
#0306 }
#0307
#0308
#0309 ////////////////////////////////////////////////////////////////
#0310
#0311
#0312 void BarGraph_OnSetFocus (HWND hwnd, HWND hwndOldFocus) {
#0313
#0314    // Cause a repaint so the focus indicator will be drawn.
#0315    InvalidateRect(hwnd, NULL, FALSE);
#0316 }
#0317
#0318
#0319 ////////////////////////////////////////////////////////////////
#0320
#0321
#0322 void BarGraph_OnKillFocus (HWND hwnd, HWND hwndNewFocus) {
#0323
#0324    // If we have the capture, release it.
#0325    if (GetCapture() == hwnd)
#0326       ReleaseCapture();
#0327
#0328    // Remove the focus indicator by causing a repaint.
#0329    InvalidateRect(hwnd, NULL, FALSE);
#0330 }
#0331
#0332
#0333 ////////////////////////////////////////////////////////////////
#0334
#0335
#0336 void BarGraph_OnCaptureChanged(HWND hwnd, HWND hwndNewCapture) {
#0337
#0338    // If we are losing the capture, cancel the current selection.
#0339    GETTHISDATA(hwnd)->nSelectedBar = GETTHISDATA(hwnd)->nCancelBar;
#0340    InvalidateRect(hwnd, NULL, FALSE);
```

```
#0341 }
#0342
#0343
#0344 ////////////////////////////////////////////////////////////////
#0345
#0346
#0347 void BarGraph_OnKey (HWND hwnd, UINT vk, BOOL fDown, int cRepeat, UINT flags) {
#0348
#0349    if (fDown) {
#0350
#0351       THISDATA* pthis = GETTHISDATA(hwnd);
#0352
#0353       switch (vk) {
#0354
#0355          case VK_LEFT:
#0356          case VK_RIGHT:
#0357
#0358             // Select the next bar to the left or right and wrap if need be
#0359             pthis->nSelectedBar += ((vk == VK_LEFT) ? (pthis->cBars - 1) : 1);
#0360             pthis->nSelectedBar %= pthis->cBars;
#0361             InvalidateRect(hwnd, NULL, FALSE);
#0362             break;
#0363
#0364          case VK_RETURN:
#0365          case VK_SPACE:
#0366             BarGraph_NotifyParent(hwnd, NM_CLICK, pthis->nSelectedBar);
#0367             break;
#0368       }
#0369    }
#0370 }
#0371
#0372
#0373 ////////////////////////////////////////////////////////////////
#0374
#0375
#0376 void BarGraph_OnPaint (HWND hwnd) {
#0377
#0378    PAINTSTRUCT ps;
#0379    UINT bar;
#0380    RECT rcClient, rcBar;
#0381    HBRUSH hbrFrame, hbrOriginal, hbrBk;
#0382    THISDATA* pthis = GETTHISDATA(hwnd);
#0383    BOOL fHighlight;
#0384    HPEN hpen, hpenOriginal;
#0385
#0386    // Start painting.
```

```
#0387    BeginPaint(hwnd, &ps);
#0388
#0389    // Draw left and bottom axes of bar graph.
#0390    GetClientRect(hwnd, &rcClient);
#0391    hpen = CreatePen(PS_SOLID, 1, BarGraph_GetTextColor(hwnd));
#0392    hpenOriginal = SelectObject(ps.hdc, hpen);
#0393    MoveToEx(ps.hdc, rcClient.left, rcClient.top, NULL);
#0394    LineTo(ps.hdc, rcClient.left, rcClient.bottom - 1);
#0395    LineTo(ps.hdc, rcClient.right, rcClient.bottom - 1);
#0396    SelectObject(ps.hdc, hpenOriginal);
#0397    DeleteObject(hpen);
#0398    rcClient.left++;
#0399    rcClient.bottom--;
#0400
#0401    // Draw each bar in the graph.
#0402    for (bar = 0; bar < pthis->cBars; bar++) {
#0403
#0404       // Get the bar's rectangle.
#0405       BarGraph_GetBarRect(hwnd, bar, &rcBar);
#0406
#0407       // Frame the rectangle.
#0408       hbrFrame = CreateSolidBrush (BarGraph_GetTextColor(hwnd));
#0409       FrameRect(ps.hdc, &rcBar, hbrFrame);
#0410       DeleteBrush(hbrFrame);
#0411       InflateRect(&rcBar, -1, -1);
#0412
#0413       // Fill the bar with the appropriate brush. If the bar we are drawing is
#0414       // the bar to highlight when we are focused (nSelectedBar), and we have
#0415       // the focus, we use COLOR_HIGHLIGHT as the background color. Otherwise,
#0416       // we use BarGraph_GetBkColor(hwnd).
#0417       fHighlight = ((bar == pthis->nSelectedBar) && (GetFocus() == hwnd));
#0418       SetBkColor(ps.hdc, fHighlight ? GetSysColor(COLOR_HIGHLIGHT) :
#0419          BarGraph_GetBkColor(hwnd));
#0420       SetBrushOrgEx(ps.hdc, 0, 0, NULL);
#0421       hbrOriginal = SelectBrush(ps.hdc, pthis->pbar[bar].hbr);
#0422       PatBlt(ps.hdc, rcBar.left, rcBar.top, rcBar.right - rcBar.left,
#0423          rcBar.bottom - rcBar.top, PATCOPY);
#0424       SelectBrush(ps.hdc, hbrOriginal);
#0425
#0426       // Inflate rect back.
#0427       InflateRect(&rcBar, 1, 1);
#0428
#0429       // Exclude the bar from the clipping region.
#0430       ExcludeClipRect(ps.hdc, rcBar.left, rcBar.top, rcBar.right, rcBar.bottom);
#0431    }
#0432
```

```
#0433    // Fill the client area with the background color.
#0434    hbrBk = CreateSolidBrush(BarGraph_GetBkColor(hwnd));
#0435    SetBrushOrgEx(ps.hdc, 0, 0, NULL);
#0436    hbrOriginal = SelectBrush(ps.hdc, hbrBk);
#0437    PatBlt(ps.hdc, rcClient.left, rcClient.top, rcClient.right - rcClient.left,
#0438       rcClient.bottom - rcClient.top, PATCOPY);
#0439    SelectBrush(ps.hdc, hbrOriginal);
#0440    DeleteObject(hbrBk);
#0441
#0442    // Done painting.
#0443    EndPaint(hwnd, &ps);
#0444 }
#0445
#0446
#0447 ////////////////////////////////////////////////////////////////////
#0448
#0449
#0450 BAR* WINAPI BarGraph_OnGetBars (HWND hwnd) {
#0451
#0452    return(GETTHISDATA(hwnd)->pbar);
#0453 }
#0454
#0455
#0456 ////////////////////////////////////////////////////////////////////
#0457
#0458
#0459 UINT WINAPI BarGraph_OnGetCount (HWND hwnd) {
#0460
#0461    return(GETTHISDATA(hwnd)->cBars);
#0462 }
#0463
#0464
#0465 ////////////////////////////////////////////////////////////////////
#0466
#0467
#0468 void WINAPI BarGraph_OnSetBars (HWND hwnd, UINT cBars, BAR* pbar) {
#0469
#0470    THISDATA* pthis;
#0471    pthis = GETTHISDATA(hwnd);
#0472    pthis->pbar = pbar;
#0473    pthis->cBars = cBars;
#0474    pthis->nSelectedBar = 0;
#0475
#0476    // Our data changed, so we need to force a repaint.
#0477    InvalidateRect(hwnd, NULL, FALSE);
#0478 }
```

```
#0479
#0480
#0481  ////////////////////////////////////////////////////////////////
#0482
#0483
#0484  UINT WINAPI BarGraph_OnGetHeight (HWND hwnd) {
#0485
#0486     return(GETTHISDATA(hwnd)->nHeight);
#0487  }
#0488
#0489
#0490  ////////////////////////////////////////////////////////////////
#0491
#0492
#0493  void WINAPI BarGraph_OnSetHeight (HWND hwnd, UINT nHeightNew) {
#0494
#0495     GETTHISDATA(hwnd)->nHeight = nHeightNew;
#0496
#0497     // Our data changed, so we need to force a repaint.
#0498     InvalidateRect(hwnd, NULL, FALSE);
#0499  }
#0500
#0501
#0502  ////////////////////////////////////////////////////////////////
#0503
#0504
#0505  void WINAPI BarGraph_OnSetBkColor (HWND hwnd, COLORREF cr) {
#0506
#0507     GETTHISDATA(hwnd)->crBk = cr;
#0508     InvalidateRect(hwnd, NULL, FALSE);
#0509  }
#0510
#0511
#0512  ////////////////////////////////////////////////////////////////
#0513
#0514
#0515  COLORREF WINAPI BarGraph_OnGetBkColor (HWND hwnd) {
#0516
#0517     return(GETTHISDATA(hwnd)->crBk);
#0518  }
#0519
#0520
#0521  ////////////////////////////////////////////////////////////////
#0522
#0523
#0524  void WINAPI BarGraph_OnSetTextColor (HWND hwnd, COLORREF cr) {
```

```
#0525
#0526    GETTHISDATA(hwnd)->crText = cr;
#0527    InvalidateRect(hwnd, NULL, FALSE);
#0528  }
#0529
#0530
#0531  ////////////////////////////////////////////////////////////////
#0532
#0533
#0534  COLORREF WINAPI BarGraph_OnGetTextColor (HWND hwnd) {
#0535
#0536    return(GETTHISDATA(hwnd)->crText);
#0537  }
#0538
#0539
#0540  ////////////////////////////////////////////////////////////////
#0541
#0542
#0543  LRESULT WINAPI BarGraph_WndProc (HWND hwnd, UINT uMsg,
#0544    WPARAM wParam, LPARAM lParam) {
#0545
#0546    switch (uMsg) {
#0547
#0548       // Standard window messages
#0549       HANDLE_MSG(hwnd, WM_CREATE,        BarGraph_OnCreate);
#0550       HANDLE_MSG(hwnd, WM_DESTROY,       BarGraph_OnDestroy);
#0551       HANDLE_MSG(hwnd, WM_GETDLGCODE,    BarGraph_OnGetDlgCode);
#0552       HANDLE_MSG(hwnd, WM_PAINT,         BarGraph_OnPaint);
#0553       HANDLE_MSG(hwnd, WM_LBUTTONDOWN,   BarGraph_OnLButtonDown);
#0554       HANDLE_MSG(hwnd, WM_LBUTTONUP,     BarGraph_OnLButtonUp);
#0555       HANDLE_MSG(hwnd, WM_MOUSEMOVE,     BarGraph_OnMouseMove);
#0556       HANDLE_MSG(hwnd, WM_SETFOCUS,      BarGraph_OnSetFocus);
#0557       HANDLE_MSG(hwnd, WM_KILLFOCUS,     BarGraph_OnKillFocus);
#0558       HANDLE_MSG(hwnd, WM_CAPTURECHANGED, BarGraph_OnCaptureChanged);
#0559       HANDLE_MSG(hwnd, WM_KEYDOWN,       BarGraph_OnKey);
#0560
#0561       // Control-specific messages
#0562       HANDLE_MSG(hwnd, BGM_SETBARS,      BarGraph_OnSetBars);
#0563       HANDLE_MSG(hwnd, BGM_GETBARS,      BarGraph_OnGetBars);
#0564       HANDLE_MSG(hwnd, BGM_GETCOUNT,     BarGraph_OnGetCount);
#0565       HANDLE_MSG(hwnd, BGM_SETHEIGHT,    BarGraph_OnSetHeight);
#0566       HANDLE_MSG(hwnd, BGM_GETHEIGHT,    BarGraph_OnGetHeight);
#0567       HANDLE_MSG(hwnd, BGM_SETBKCOLOR,   BarGraph_OnSetBkColor);
#0568       HANDLE_MSG(hwnd, BGM_GETBKCOLOR,   BarGraph_OnGetBkColor);
#0569       HANDLE_MSG(hwnd, BGM_SETTEXTCOLOR, BarGraph_OnSetTextColor);
#0570       HANDLE_MSG(hwnd, BGM_GETTEXTCOLOR, BarGraph_OnGetTextColor);
```

```
#0571
#0572    }
#0573    return(DefWindowProc(hwnd, uMsg, wParam, lParam));
#0574  }
#0575
#0576
#0577  ////////////////////////////////////////////////////////////////
#0578
#0579
#0580  ATOM WINAPI BarGraph_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass) {
#0581
#0582    WNDCLASSEX wc;
#0583
#0584    adgINITSTRUCT(wc, TRUE);
#0585    wc.style        = CS_HREDRAW | CS_VREDRAW;
#0586    wc.lpfnWndProc   = BarGraph_WndProc;
#0587    wc.cbWndExtra    = sizeof(BARGRAPH_WNDEXTRABYTES);
#0588    wc.hInstance     = hinst;
#0589    wc.hCursor      = LoadCursor(NULL, IDC_ARROW);
#0590    wc.lpszClassName = WC_BARGRAPH;
#0591
#0592    if (fGlobalClass)
#0593       wc.style |= CS_GLOBALCLASS;
#0594
#0595    return(RegisterClassEx(&wc));
#0596  }
#0597
#0598
#0599  ////////////////////////////////////////////////////////////////
#0600
#0601
#0602  BOOL WINAPI BarGraph_UnregisterClass (HINSTANCE hinst) {
#0603
#0604    return(UnregisterClass(WC_BARGRAPH, hinst));
#0605  }
#0606
#0607
#0608  ////////////////////////// End of File ///////////////////////////
```

程式列表 4.6　BarGraph.MSG

```
#0001  // Module name: BarGraph.msg
#0002  // Written by: Jonathan Locke
#0003  // Notices: Copyright (c) 1995 Jeffrey Richter
#0004  // Purpose: 'MsgCrack' input file for BarGraph control.
```

```
#0005
#0006  MessageBase WM_USER
#0007  MessageClass BarGraph
#0008
#0009  Message BGM_SETBARS SetBars \
#0010   - Sets the array of BAR structures for the control. The brush in each BAR\
#0011   structure is only USED by the control. It is OWNED by the caller. Therefore,\
#0012   the caller must create and destroy the brush handle.
#0013  wParam UINT cBars - Number of BAR structures in array
#0014  lParam BAR* pbar - Pointer to first BAR in array
#0015  .
#0016
#0017  Message BGM_GETBARS GetBars \
#0018   - Gets pointer to BAR array for control.
#0019  Returns BAR* - Pointer to first BAR in array
#0020  .
#0021
#0022  Message BGM_GETCOUNT GetCount \
#0023   - Gets number of BAR structures in array
#0024  Returns UINT - Number of BARs
#0025  .
#0026
#0027  Message BGM_SETHEIGHT SetHeight \
#0028   - Sets the current graph height (in user-defined units)
#0029  wParam UINT nHeight - New graph height
#0030  .
#0031
#0032  Message BGM_GETHEIGHT GetHeight \
#0033   - Gets the current graph height (in user-defined units)
#0034  Returns UINT - Current graph height
#0035  .
#0036
#0037  Message BGM_SETBKCOLOR SetBkColor \
#0038   - Sets the background color of a bargraph control.
#0039  lParam COLORREF crNew - New background color
#0040  .
#0041
#0042  Message BGM_GETBKCOLOR GetBkColor \
#0043   - Gets the background color of a bargraph control.
#0044  Returns COLORREF - Current background color
#0045  .
#0046
#0047  Message BGM_SETTEXTCOLOR SetTextColor \
#0048   - Sets the text color of a bargraph control.
#0049  lParam COLORREF crNew - New text color
#0050  .
```

```
#0051
#0052  Message BGM_GETTEXTCOLOR GetTextColor \
#0053   - Gets the text color of a bargraph control.
#0054  Returns COLORREF - Current text color
#0055  .
```

程式列表 4.7　BarGraph.H

```
#0001  /************************************************************
#0002  Module name: BarGraph.h
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Bargraph custom control header file.
#0006  ************************************************************/
#0007
#0008
#0009  //////////////////////////// Class Name ////////////////////////////
#0010
#0011
#0012  // ANSI/Unicode declarations for bargraph's window class name
#0013  #define WC_BARGRAPHA "BarGraph"
#0014  #define WC_BARGRAPHW L"BarGraph"
#0015
#0016  #ifdef UNICODE
#0017  #define WC_BARGRAPH WC_BARGRAPHW
#0018  #else
#0019  #define WC_BARGRAPH WC_BARGRAPHA
#0020  #endif
#0021
#0022
#0023  ////////////////// Class-Specific Window Styles ////////////////////
#0024
#0025
#0026  // Bargraph controls don't have any class-specific window styles at this time.
#0027
#0028
#0029  ////////////////////// Class Registration ///////////////////////
#0030
#0031
#0032  // Register and unregister the bargraph window class.
#0033  ATOM WINAPI BarGraph_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass);
#0034  BOOL WINAPI BarGraph_UnregisterClass (HINSTANCE hinst);
#0035
#0036
#0037  ////////////////////// Parent Notification ///////////////////////
```

```
#0038
#0039
#0040  // The following is the BARGRAPH_NOTIFY structure sent to the parent in the
#0041  // lParam of a WM_NOTIFY message. The NMHDR structure can be found in
#0042  // WinUser.H.
#0043  typedef struct tagBARGRAPH_NOTIFY {
#0044    NMHDR nmh;         // Notify message header
#0045    UINT uBar;         // Index of bar in graph
#0046  } BARGRAPH_NOTIFY;
#0047
#0048
#0049  ////////////////////// Definition of a Bar ////////////////////////
#0050
#0051
#0052  // BAR structure defines each bar in the bar graph.
#0053  typedef struct tagBAR {
#0054    UINT nHeight;       // Height of bar in user-defined units
#0055    HBRUSH hbr;         // Brush to fill bar with
#0056  } BAR;
#0057
#0058
#0059  ///////////////// Class-Specific Window Messages /////////////////
#0060
#0061
#0062  //{{adgMSGCRACK_MESSAGES
#0063
#0064  // Purpose: Sets the array of BAR structures for the control. The brush in each
#0065  //          BAR structure is only used by the control. It is owned by the
#0066  //          caller. Therefore, the caller must create and destroy the brush
#0067  //          handle.
#0068  // wParam:  UINT cBars - Number of BAR structures in array
#0069  // lParam:  BAR* pbar - Pointer to first BAR in array
#0070  // Returns: void
#0071  #define BGM_SETBARS    (WM_USER + 0)
#0072
#0073  // Purpose: Gets pointer to BAR array for control.
#0074  // wParam:  N/A
#0075  // lParam:  N/A
#0076  // Returns: BAR* - Pointer to first BAR in array
#0077  #define BGM_GETBARS    (WM_USER + 1)
#0078
#0079  // Purpose: Gets number of BAR structures in array.
#0080  // wParam:  N/A
#0081  // lParam:  N/A
#0082  // Returns: UINT - Number of BARs
#0083  #define BGM_GETCOUNT    (WM_USER + 2)
```

```
#0084
#0085  // Purpose: Sets the current graph height (in user-defined units).
#0086  // wParam:  UINT nHeight - New graph height
#0087  // lParam:  N/A
#0088  // Returns: void
#0089  #define BGM_SETHEIGHT    (WM_USER + 3)
#0090
#0091  // Purpose: Gets the current graph height (in user-defined units).
#0092  // wParam:  N/A
#0093  // lParam:  N/A
#0094  // Returns: UINT - Current graph height
#0095  #define BGM_GETHEIGHT    (WM_USER + 4)
#0096
#0097  // Purpose: Sets the background color of a bargraph control.
#0098  // wParam:  N/A
#0099  // lParam:  COLORREF crNew - New background color
#0100  // Returns: void
#0101  #define BGM_SETBKCOLOR    (WM_USER + 5)
#0102
#0103  // Purpose: Gets the background color of a bargraph control.
#0104  // wParam:  N/A
#0105  // lParam:  N/A
#0106  // Returns: COLORREF - Current background color
#0107  #define BGM_GETBKCOLOR    (WM_USER + 6)
#0108
#0109  // Purpose: Sets the text color of a bargraph control.
#0110  // wParam:  N/A
#0111  // lParam:  COLORREF crNew - New text color
#0112  // Returns: void
#0113  #define BGM_SETTEXTCOLOR   (WM_USER + 7)
#0114
#0115  // Purpose: Gets the text color of a bargraph control.
#0116  // wParam:  N/A
#0117  // lParam:  N/A
#0118  // Returns: COLORREF - Current text color
#0119  #define BGM_GETTEXTCOLOR   (WM_USER + 8)
#0120
#0121  //}}adgMSGCRACK_MESSAGES
#0122
#0123
#0124  ////////////////// Bargraph Control Message APIs ////////////////////
#0125
#0126
#0127  //{{adgMSGCRACK_APIS
#0128
#0129  #define BarGraph_SetBars(hwnd, cBars, pbar) \
```

```
#0130    ((void)SendMessage((hwnd), BGM_SETBARS, (WPARAM)(DWORD)(cBars),
(LPARAM)(DWORD)(pbar)))
#0131
#0132  #define BarGraph_GetBars(hwnd) \
#0133    ((BAR* )SendMessage((hwnd), BGM_GETBARS, 0, 0))
#0134
#0135  #define BarGraph_GetCount(hwnd) \
#0136    ((UINT )SendMessage((hwnd), BGM_GETCOUNT, 0, 0))
#0137
#0138  #define BarGraph_SetHeight(hwnd, nHeight) \
#0139    ((void)SendMessage((hwnd), BGM_SETHEIGHT, (WPARAM)(DWORD)(nHeight), 0))
#0140
#0141  #define BarGraph_GetHeight(hwnd) \
#0142    ((UINT )SendMessage((hwnd), BGM_GETHEIGHT, 0, 0))
#0143
#0144  #define BarGraph_SetBkColor(hwnd, crNew) \
#0145    ((void)SendMessage((hwnd), BGM_SETBKCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0146
#0147  #define BarGraph_GetBkColor(hwnd) \
#0148    ((COLORREF )SendMessage((hwnd), BGM_GETBKCOLOR, 0, 0))
#0149
#0150  #define BarGraph_SetTextColor(hwnd, crNew) \
#0151    ((void)SendMessage((hwnd), BGM_SETTEXTCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0152
#0153  #define BarGraph_GetTextColor(hwnd) \
#0154    ((COLORREF )SendMessage((hwnd), BGM_GETTEXTCOLOR, 0, 0))
#0155
#0156  //}}adgMSGCRACK_APIS
#0157
#0158
#0159  ////////////////////// Message Crackers //////////////////////
#0160
#0161
#0162  //{{adgMSGCRACK_CRACKERS
#0163
#0164  // void Cls_OnSetBars (HWND hwnd, UINT cBars, BAR* pbar)
#0165  #define HANDLE_BGM_SETBARS(hwnd, wParam, lParam, fn) \
#0166    ((fn)((hwnd), (UINT)(wParam), (BAR*)(lParam)), 0)
#0167  #define FORWARD_BGM_SETBARS(hwnd, cBars, pbar, fn) \
#0168    (void)((fn)((hwnd), BGM_SETBARS, (WPARAM)(DWORD)(cBars),
(LPARAM)(DWORD)(pbar)))
#0169
#0170  // BAR* Cls_OnGetBars(HWND hwnd)
#0171  #define HANDLE_BGM_GETBARS(hwnd, wParam, lParam, fn) \
#0172    (LRESULT)(fn)((hwnd))
#0173  #define FORWARD_BGM_GETBARS(hwnd, fn) \
```

```
#0174    (BAR* )((fn)((hwnd), BGM_GETBARS, 0, 0))
#0175
#0176  // UINT Cls_OnGetCount(HWND hwnd)
#0177  #define HANDLE_BGM_GETCOUNT(hwnd, wParam, lParam, fn) \
#0178    (LRESULT)(fn)((hwnd))
#0179  #define FORWARD_BGM_GETCOUNT(hwnd, fn) \
#0180    (UINT)((fn)((hwnd), BGM_GETCOUNT, 0, 0))
#0181
#0182  // void Cls_OnSetHeight (HWND hwnd, UINT nHeight)
#0183  #define HANDLE_BGM_SETHEIGHT(hwnd, wParam, lParam, fn) \
#0184    ((fn)((hwnd), (UINT)(wParam)), 0)
#0185  #define FORWARD_BGM_SETHEIGHT(hwnd, nHeight, fn) \
#0186    (void)((fn)((hwnd), BGM_SETHEIGHT, (WPARAM)(DWORD)(nHeight), 0))
#0187
#0188  // UINT Cls_OnGetHeight (HWND hwnd)
#0189  #define HANDLE_BGM_GETHEIGHT(hwnd, wParam, lParam, fn) \
#0190    (LRESULT)(fn)((hwnd))
#0191  #define FORWARD_BGM_GETHEIGHT(hwnd, fn) \
#0192    (UINT )((fn)((hwnd), BGM_GETHEIGHT, 0, 0))
#0193
#0194  // void Cls_OnSetBkColor (HWND hwnd, COLORREF crNew)
#0195  #define HANDLE_BGM_SETBKCOLOR(hwnd, wParam, lParam, fn) \
#0196    ((fn)((hwnd), (COLORREF)(lParam)), 0)
#0197  #define FORWARD_BGM_SETBKCOLOR(hwnd, crNew, fn) \
#0198    (void)((fn)((hwnd), BGM_SETBKCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0199
#0200  // COLORREF Cls_OnGetBkColor (HWND hwnd)
#0201  #define HANDLE_BGM_GETBKCOLOR(hwnd, wParam, lParam, fn) \
#0202    (LRESULT)(fn)((hwnd))
#0203  #define FORWARD_BGM_GETBKCOLOR(hwnd, fn) \
#0204    (COLORREF )((fn)((hwnd), BGM_GETBKCOLOR, 0, 0))
#0205
#0206  // void Cls_OnSetTextColor (HWND hwnd, COLORREF crNew)
#0207  #define HANDLE_BGM_SETTEXTCOLOR(hwnd, wParam, lParam, fn) \
#0208    ((fn)((hwnd), (COLORREF)(lParam)), 0)
#0209  #define FORWARD_BGM_SETTEXTCOLOR(hwnd, crNew, fn) \
#0210    (void)((fn)((hwnd), BGM_SETTEXTCOLOR, 0, (LPARAM)(DWORD)(crNew)))
#0211
#0212  // COLORREF Cls_OnGetTextColor(HWND hwnd)
#0213  #define HANDLE_BGM_GETTEXTCOLOR(hwnd, wParam, lParam, fn) \
#0214    (LRESULT)(fn)((hwnd))
#0215  #define FORWARD_BGM_GETTEXTCOLOR(hwnd, fn) \
#0216    (COLORREF )((fn)((hwnd), BGM_GETTEXTCOLOR, 0, 0))
#0217
#0218  //}}adgMSGCRACK_CRACKERS
#0219
```

```
#0220
#0221  //////////////////////////// End of File /////////////////////////////
```

程式列表 4.8　CustCntl.RC

```
#0001  //Microsoft Visual C++-generated resource script
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012
#0013  ////////////////////////////////////////////////////////////////////
#0014  #undef APSTUDIO_READONLY_SYMBOLS
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////////
#0018  //
#0019  // Icon
#0020  //
#0021
#0022  IDI_CUSTCNTL           ICON    DISCARDABLE     "CUSTCNTL.ICO"
#0023
#0024  ////////////////////////////////////////////////////////////////////
#0025  //
#0026  // Dialog box
#0027  //
#0028
#0029  IDD_CUSTCNTL DIALOG DISCARDABLE  0x8000, 5, 272, 154
#0030  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0031  CAPTION "Custom Controls"
#0032  FONT 8, "MS Sans Serif"
#0033  BEGIN
#0034      LTEXT         "&Number of User API Functions",IDC_APILABEL,12,59,37,40,
#0035                    NOT WS_GROUP
#0036      CONTROL        "",IDC_BARGRAPH,"BarGraph",WS_TABSTOP,53,20,107,111
#0037      CTEXT         "Windows Version",IDC_STATIC,53,135,109,9,NOT WS_GROUP
#0038      CONTROL        "Windows 1.04",IDC_LEGEND0,"Legend",0x3,187,18,71,14
#0039      CONTROL        "Windows 2.11",IDC_LEGEND1,"Legend",0x3,187,32,71,14
#0040      CONTROL        "Windows 3.0",IDC_LEGEND2,"Legend",0x3,187,46,71,14
```

```
#0041     CONTROL        "Windows 3.1",IDC_LEGEND3,"Legend",0x3,187,60,71,14
#0042     CONTROL        "Windows 95",IDC_LEGEND4,"Legend",0x3,187,73,71,14
#0043     GROUPBOX       "Legend",IDC_STATIC,181,6,83,87
#0044     GROUPBOX       "&Module",IDC_STATIC,181,98,83,48
#0045     CONTROL        "&User",IDC_USER,"Button",BS_AUTORADIOBUTTON | WS_GROUP,
#0046                    191,110,32,10
#0047     CONTROL        "&GDI",IDC_GDI,"Button",BS_AUTORADIOBUTTON,191,121,31,10
#0048     CONTROL        "&Kernel",IDC_KERNEL,"Button",BS_AUTORADIOBUTTON,191,132,
#0049                    35,10
#0050  END
#0051
#0052
#0053  #ifdef APSTUDIO_INVOKED
#0054  ////////////////////////////////////////////////////////////////////
#0055  //
#0056  // TEXTINCLUDE
#0057  //
#0058
#0059  1 TEXTINCLUDE DISCARDABLE
#0060  BEGIN
#0061     "resource.h\0"
#0062  END
#0063
#0064  2 TEXTINCLUDE DISCARDABLE
#0065  BEGIN
#0066     "#include ""windows.h""\r\n"
#0067     "\r\n"
#0068     "\0"
#0069  END
#0070
#0071  3 TEXTINCLUDE DISCARDABLE
#0072  BEGIN
#0073     "\r\n"
#0074     "\0"
#0075  END
#0076
#0077  ////////////////////////////////////////////////////////////////////
#0078  #endif    // APSTUDIO_INVOKED
#0079
#0080
#0081  #ifndef APSTUDIO_INVOKED
#0082  ////////////////////////////////////////////////////////////////////
#0083  //
#0084  // Generated from the TEXTINCLUDE 3 resource.
#0085  //
#0086
```

```
#0087
#0088  //////////////////////////////////////////////////////////////
#0089  #endif    // not APSTUDIO_INVOKED
```

程式列表 4.9　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by CUSTCNTL.RC
#0004  //
#0005  #define IDI_CUSTCNTL                102
#0006  #define IDD_CUSTCNTL                103
#0007  #define IDC_LEGEND0                 1000
#0008  #define IDC_LEGEND1                 1001
#0009  #define IDC_LEGEND2                 1002
#0010  #define IDC_LEGEND3                 1003
#0011  #define IDC_LEGEND4                 1004
#0012  #define IDC_BARGRAPH                1005
#0013  #define IDC_USER                    1006
#0014  #define IDC_GDI                     1007
#0015  #define IDC_KERNEL                  1008
#0016  #define IDC_APILABEL                1009
#0017  #define IDC_BUTTON1                 1010
#0018  #define IDC_STATIC                  -1
#0019
#0020  // Next default values for new objects
#0021  //
#0022  #ifdef APSTUDIO_INVOKED
#0023  #ifndef APSTUDIO_READONLY_SYMBOLS
#0024  #define _APS_NEXT_RESOURCE_VALUE      104
#0025  #define _APS_NEXT_COMMAND_VALUE       40001
#0026  #define _APS_NEXT_CONTROL_VALUE       1011
#0027  #define _APS_NEXT_SYMED_VALUE         101
#0028  #endif
#0029  #endif
```

第５章

Window Subclassing 和 Window Superclassing

Microsoft Windows 提供了許多現成好用的視窗類別，像是 listbox、combobox、scrollbar 等等。這些控制元件通常可直接使用，不需再經特別處理。一般來說它們已足夠程式員使用。然而有些時候，你可能想要稍微做些改變以符合你自己的需求。

解決這問題的方法之一就是設計你自己的控制元件（請參閱第４章）。這可是件大工程；如果微軟公司能將各控制元件的原始碼公開，那真是我們莫大的福音啊！不幸地是，微軟並沒有這樣做。所以我們必須另尋它法來解決這個問題。啊，我們可以使用 Windows 所提供的 subclassing 和 superclassing 來幫助我們達成目的。

Subclassing 和 superclassing 也可應用在你自己建立的視窗類別上，像是程式的視窗類別和你的訂製型控制元件（custom control）等等。然而通常沒有這個必要，因爲你可以輕易取得並修改你自己的原始碼，做你愛做的事。

Window Subclassing 如何運作

所謂註冊視窗類別，即是將類別資料填入你的 WNDCLASSEX 結構各欄位中，再將此結構的位址交給 RegisterClassEx 函式去註冊。當然，在使用 RegisterClassEx 函式之前，你必須指定 WNDCLASSEX 結構的 lpfnWndProc 欄位，使它指向此類別之視窗函式位址。

這個函式處理所有由此類別所產生之視窗的所有訊息。在 RegisterClassEx 函式回返之前，系統會配置一塊記憶體，記錄著這個視窗類別的相關資訊 -- 當然也記錄了視窗函式的位址。

每當產生一個新視窗，系統便會配置一塊新的視窗記憶體區塊，用以記錄新視窗的相關資訊。當系統爲此視窗配置記憶體並初始化之後，便從類別的記憶體區塊中複製其視窗函式的位址到視窗的記憶體區塊中。當有任何訊息被分派（dispatch）到視窗函式時，系統會依據「視窗記憶體區塊」中所記載的視窗函式位址來呼叫視窗函式，而不是依據「類別記憶體區塊」中所記載的視窗函式位址來呼叫視窗函式。類別記憶體區塊中所記載的視窗函式位址只做複製之用，並無其它用途。

爲了 subclassing 一個視窗，你必須將視窗記憶體區塊中的視窗函式位址改變，使它指向一個新的視窗函式。而由於這個位址的改變只是發生在某一個視窗記憶體區塊中，所以不會影響同一類別所衍生的其它視窗。如果系統沒有讓每一個視窗實體擁有自己的一份視窗函式位址，當類別之中的視窗函式位址改變，可就會影響到由此類別所產生之所有視窗的行爲了！果真如此，如果我們對某個 edit 控制元件做 subclassing 動作，使它不接受數字輸入，將會導致系統中所有的 edit 控制元件都不接受數字輸入。這並不是我們想要的結果。

一旦你改變了視窗記憶體區塊中的視窗函式位址，所有送給視窗的訊息，都會交由這個新位址所指向的視窗函式去處理。這個函式的原型必須和標準視窗函式一樣。換句話說，所有視窗函式的原型是一致的：

```
LRESULT WINAPI SubClassWndProc (HWND hwnd, UINT uMsg,
                                WPARAM wParam, LPARAM lParam);
```

一旦原本要流往「原視窗函式」的訊息流到了你的視窗函式，你可能會做這些事情：

- 原封不動地傳給原視窗函式。對於大部份訊息，我們都是這麼處理，原因是 subclassing 技術通常只對視窗的行爲稍做修改，所以大部份訊息還是傳給其原視窗函式，以執行其預設動作。

- 限制只有某些訊息才可以傳入原視窗函式。舉個例，假如你希望 edit 控制元件停止接受數字輸入，你必須處理 WM_CHAR 訊息，檢查看看是否輸入字元（wParam 參數）介於 '0' 到 '9' 之間。如果是，什麼也不做就立即回返；如果不是，才將訊息送給原視窗函式去處理。稍後的一個範例程式就是這樣做的。
- 將訊息交給原視窗函式之前，先做修改。如果你希望 combobox 控制元件只接受大寫字母，你應該處理每一個 WM_CHAR 訊息，將其 wParam 轉會爲大寫字母（使用 CharUpper 函式），然後再將此訊息送至原視窗函式。

如何 subclassing 一個視窗？以下是一個實例：

```
static LPCSTR g_szSubclassWndProcOrig = "SubclassWndProcOrig";

int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   HWND hwndEdit = CreateWindowEx(0, "EDIT", "", WS_CHILD,
      10, 20, 100, 16, hWndParent, NULL, hInstance, OL);

   // Change the window procedure address associated with this one edit control.
   pfnWndProcOrig = SubclassWindow(hwndEdit, Edit_SubClassWndProc);

   // Associate the original window procedure address with the window.
   SetProp(hwndEdit, g_szSubclassWndProcOrig, (HANDLE) (DWORD) pfnWndProcOrig);

   // All messages destined for hwndEdit are sent to
   // Edit_SubClassWndProc instead.
}
.
.
.
LRESULT WINAPI Edit_SubClassWndProc (HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam) {

   LRESULT lResult = 0;
   BOOL fCallOrigWndProc = TRUE;

   switch (uMsg) {

      // Some cases set fCallOrigWndProc to FALSE.
```

```
      .
      .
      .
   }
   if (fCallOrigWndProc)
      lResult = CallWindowProc(
        (WNDPROC) (DWORD) GetProp(hwnd, g_szSubclassWndProcOrig),
        hwnd, uMsg, wParam, lParam);

    return(lResult);
}
```

要對某個視窗做 subclassing 動作，唯一需要的資料就是其視窗代碼。注意，我使用 SubClassWindow 巨集來執行視窗的 subclassing 動作，這個巨集在 WindowsX.H 檔中定義如下：

```
#define SubclassWindow(hwnd, lpfn) \
      ((WNDPROC)SetWindowLong((hwnd), GWL_WNDPROC, \
      (LPARAM)(WNDPROC)(lpfn)))
```

你可以看到，除了型別轉換之外，這個巨集其實只是呼叫 SetWindowLong 來改變視窗函式的位址而已。當然，指定給 SubclassWindow 巨集做爲第二個參數的函式位址必須確實存在。在 Win32 環境下，你不可能 subclassing 一個視窗，卻使用一個位於另一行程位址空間中的視窗函式[1]。SetWindowLong 巨集的傳回值是原視窗函式的位址。這個視窗函式的位址被儲存在「被 subclassed 視窗」的 property 之中（侯俊傑註：這只是本書作者所選擇的一種作法，並非一定要如此）。

新視窗函式的形式和一般視窗函式是一樣的，唯一的差別是，它不再把訊息交給 DefWindowProc，而是交給原視窗函式。我們必須呼叫 CallWindowProc，才能夠將訊息傳給原視窗函式，以執行其正常運作。呼叫此函式時，需傳入原視窗函式位址、視窗代

[1] 如果你想知道如何能夠以另一程式之函式來“subclassing”你的某個視窗，請參閱我所寫的“Advanced Windows”一書（Microsoft Press，1995 年發行）中的“Breaking Through Process Boundary Walls”一節。

碼（hwnd）、訊息（uMsg）和兩個標準參數（wParam 和 lParam）。

注意，我們可以使用 GetProp 函式取得視窗的 property，於是獲得原視窗函式的位址。在我建議你使用視窗的 property 來存放原視窗函式位址之前，我反覆思索了很久。我想沒有其他更好的地方可以用來存放原視窗函式位址了。這裡我要列出我的一些想法，並解釋每一個想法的優缺點：

- 當你 subclassing 一個個視窗時，將每一個視窗的「原視窗函式位址」和「視窗本身」產生密切關聯，是很合理的。有一個想法是，將原視窗函式位址存放在視窗的 GWL_USERDATA 位元組中。你是可以這麼做，只不過它有個缺點。由於你正在 subclassing 某個視窗，你不應該使用 GWL_USERDATA 位元組，因爲你不知道原視窗函式有沒有用到這些位元組。本章稍後的「取得 subclassed 視窗的額外資料」一節，會對此做更多討論。
- 我的下一個想法是，將原視窗函式的位址存放在全域變數中。這很容易但缺乏彈性。舉個例子，如果你有一些視窗要被 subclassed，於是你必須爲每一個視窗準備一個全域變數。如果你要對同一類別所衍生的多個視窗做 subclassing 動作，你其實只需要儲存原視窗函式一次即可。我覺得全域變數是罪惡的淵藪，能避免就盡量避免。
- 當我撰寫本章時，我曾經想建議以下的方法：

```
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   HWND hwndEdit = CreateWindowEx(0, "EDIT", "", WS_CHILD,
      10, 20, 100, 16, hWndParent, NULL, hInstance, OL);

   // Change the window procedure address associated with this one edit control.
   SubclassWindow(hwndEdit, Edit_SubClassWndProc);

   // All messages destined for hwndEdit are sent to
   // Edit_SubClassWndProc instead.
}
.
.
.
LRESULT WINAPI Edit_SubClassWndProc (HWND hwnd, UINT uMsg,
```

```
    WPARAM wParam, LPARAM lparam) {

    LRESULT lResult = 0;
    BOOL fCallOrigWndProc = TRUE;

    switch (uMsg) {

        // Some cases set fCallOrigWndProc to FALSE.
        .
        .
        .
    }
    if (fCallOrigWndProc)
       lResult = CallWindowProc(GetClassLong(hwnd, GCL_WNDPROC),
         hwnd, uMsg, wParam, lParam);

     return(lResult);
}
```

注意，這個方法並沒有用到全域變數，而是引用 GCL_WNDPROC 識別碼來呼叫 GetClassLong，取得記錄在類別結構中的原視窗函式位址，再利用 CallWindowProc 函式呼叫之。還記得嗎，SubclassWindow 巨集是使用 SetWindowLong 函式來改變視窗函式位址。你當然還是可以藉由類別結構中的資訊來取得原視窗函式位址，然而，我決定不採用這個方法，原因是本章所有的技術檢閱人員都憎恨這個方法。他們之所以憎恨，因爲現今市場上一些工具程式已經 subclassing 各式各樣的視窗，造成系統中可能有些視窗被 subclassed 許多次 -- Windows 並不阻止這樣的事情發生。但如果任何 subclass 視窗函式都使用這種技術的話，後來添加的 subclass 視窗函式就沒有機會處理訊息了（侯俊傑註：也就是說，不能夠形成一個 subclass 串鏈）。

回顧這些想法，很明顯地，與 subclass 視窗函式溝通的最好方法還是使用視窗的 property，唯一缺點就是存取速度較慢。我曾經詢問過幾位 Windows 95 和 Windows NT 開發人員有關於這方面的問題，他們告訴我說使用 properties 不會壞到那裡去，他們也告訴我說像 MFC、control 3-D Library 和 common dialog boxes 都是使用 properties 來存放視窗的原視窗函式位址。如果你擔心執行效率的話，你可以使用較短的 property 名稱或使用 atom 值來呼叫 SetProp 函式和 GetProp 函式。

另一個問題是，subclassing 串鏈很容易被打斷。舉個例，一個視窗可能先被 DLL-A 做 subclassing 動作，而後這個視窗又被 DLL-B 做 subclassing 動作。Windows 允許這樣的技術，而且運作良好。現在，當一個訊息發送到視窗，DLL-B 的 subclass 視窗函式會收到這個訊息。當它處理完畢，此訊息會傳給 DLL-A 的 subclass 視窗函式。之所以會這樣，是因爲當 DLL-B 呼叫 SubclassWindow（譯註：前數頁出現過的一個巨集），獲得的是 DLL-A 的 subclass 視窗函式位址。當然，如果 DLL-B 沒有儲存原視窗函式的位址，而是使用前述三種方法中的最後一種的話，串鏈就被打斷了，DLL-A 中的 subclass 視窗函式就不會被呼叫。這就是爲什麼我不鼓勵你使用最後一種方法的原因。

現在讓我們看看，如果 DLL-A 想要 “un-subclassing” 一個視窗，會發生什麼事情。當程式意圖 un-subclassing 一個視窗，它應該呼叫 SubclassWindow 巨集（譯註：前數頁曾出現過，還記得嗎？），傳入 DLL-A 所儲存的原視窗函式位址，這個位址就是視窗類別的視窗函式位址。從此以後，所有屬於該視窗的訊息皆會直接傳送給原視窗函式。這其實不是我們想要的 -- DLL-B 仍然想要 subclassing 這個視窗，現在卻無能爲力了，因爲 subclass 串鏈已被破壞。

不幸的是，我們無法解決這個問題。這個問題需要作業系統提供更多的支援，允許「發動 subclassing 動作」的那個程式能夠優雅地從串鏈中移除它們自己。

送 subclassed 視窗私有新的視窗訊息

通常，當我們 subclassing 一個視窗，我們會想要定義一些新的視窗訊息，交給視窗處理。譬如說我們產生一個 edit 視窗，只接受數字（使用 ES_NUMBER 視窗風格）。而後，我們 subclassing 這個視窗，將它的行爲稍做改變。我們或許會想定義一個新的訊息 EM_ADDNUM，送給這個被 subclassed 的 edit 視窗，讓它先將數字加上一個值再顯示出來。

我們如何定義這個 EM_ADDNUM 訊息呢？一般而言，如果我們自己建立了一個視窗類別，我們可以定義自己的類別專屬訊息，編號從 WM_USER 開始。舉個例子，我們可以定義 EM_ADDNUM 訊息如下：

```
#define EM_ADDNUM    (WM_USER + 0)
```

但是，當我們 subclassing 一個視窗，而其視窗類別不是由我們自己定義的時候，我們不能夠從 WM_USER 開始定義新的視窗訊息。所有新的通用控制元件（common controls），像是 trackbar 等等，都有它們自己的類別專屬訊息，而且都是從 WM_USER 開始定義。例如 trackbar 類別的 TBM_GETPOS 訊息定義如下：

```
#define TBM_GETPOS    (WM_USER)
```

如果我們要 subclassing 這個 trackbar 視窗並且從 WM_USER+0 開始定義我們自己的訊息，我們的 subclass 視窗函式就會攔截這個訊息並以我們自己的方法來處理這個訊息。結果呢，程式原本期望送出 TBM_GETPOS 訊息以取得尺寸相關資訊，卻獲得一個大驚奇！

好，你可以將你的訊息從 WM_USER+500 開始定義。但這並不代表就安全了，因爲我們不知道微軟公司是否爲這個 trackbar 控制元件定義了一些未公開訊息（從 WM_USER+x 開始定義）。如果我們從 WM_USER+500 開始定義自己的訊息，說不定還是會與那些未公開訊息相衝。事實上，選擇任何值都是不安全的，你只能聽天由命！

很幸運地，有兩種方法可以解決這個問題。第一種方法是使用第 1 章曾介紹過的 RegisterWindowMessage 函式。這個函式會產生一個新的訊息，而且其編碼獨一無二，只要傳入此函式的字串（代表訊息名稱）是獨一無二的就行囉！每當我們呼叫 RegisterWindowMessage，它會傳回一個整數值，做爲訊息的識別碼，它一定不會和系統所定義的任何訊息相衝突。

第二種方法較爲簡單。從 Windows 95 開始，微軟公司便宣告說，所有協力廠商（third-party）所製作的視窗類別的視窗函式必須忽略 WM_APP ~ 0XBFFFF 之間的應用程式專屬訊息，其中 WM_APP 定義於 WinUser.H 檔：

```
#define WM_APP    0x8000
```

這表示微軟公司保證所有的 ”system global” 視窗類別（譯註：第１章討論過的主題）和所有新的通用控制元件（common controls）都會忽略這個範圍內的所有訊息。這也意味其它廠商所製造出來的控制元件也應該忽略這個範圍內的訊息。所有你所使用的「由協力廠商提供的控制元件」，你都應該實際檢查看看，確保它們不會處理 WM_APP 以上的任何訊息。

由於這些類別會忽略上述範圍內的訊息，所以我們可以安心使用，爲我們的「subclassed edit 視窗」定義 EM_ADDNUM 訊息如下：

```
#define EM_ADDNUM    (WM_APP + 0)
```

現在，當我們送這個訊息到「subclassed edit 視窗」，它就會知道我們想要做的事：將數值再加上一個值，然後才顯現到視窗中。

取得 subclassed 視窗的額外資料

subclassing 視窗所帶來的另一個問題是，使用視窗的「額外位元組」是很困難的。舉個例子，我們可能會想要在我們的 subclassed edit 視窗中設定一個「有效數值範圍」。由於我們並沒有註冊這個 edit 視窗類別，所以我們沒辦法增加「類別的額外位元組」或「視窗的額外位元組」。此外，由於 subclassing 動作的先決條件是必須先存在一個視窗，所以增加該視窗之「類別額外位元組」或「視窗額外位元組」也是不可能的。

將一筆資料和一個 subclassed 視窗關連在一起，最好的方法還是使用視窗的 properties -- 我在第１章曾介紹過這項技術。你也可以利用 GWL_USERDATA 位元組儲存一個資料結構的位址，並在該資料結構中存放一些附加資料，但這並不是好方法。要知道，GWL_USERDATA 位元組通常是給那個「產生並操控此一視窗」之應用程式所用，由於你的 subclass 函式並不是視窗的產生者，所以你不應動用 GWL_USERDATA 位元組。視窗的 properties 才是你的最佳選擇。

當然，如果你知道被你 subclassed 的那個視窗沒有用到 GWL_USERDATA，你就可以放

心大膽地使用它，但是這麼一來你會犧牲掉一些未來的擴充彈性。

NoDigits 應用實例

NoDigits 程式（NoDigits.EXE）的原始碼顯示於**程式列表 5.1 ~ 5.4**，示範如何 subclassing 兩個 edit 視窗。當你執行此程式，主視窗如**圖 5.1**。這個視窗內含三個 edit 控制元件和兩個 buttons。第一個 edit 控制元件使用標準的、內建的視窗函式，另兩個 edit 控制元件則被 subclassed 過；其「subclass 視窗函式」會攔截所有原本要送到 edit 控制元件的字元。如果字元是數字的話，「subclass 視窗函式」不會呼叫原來的 edit 視窗函式，而會發出嗶嗶聲並直接回返（return）。

此程式所需用到的檔案，列在**表 5.1** 中。

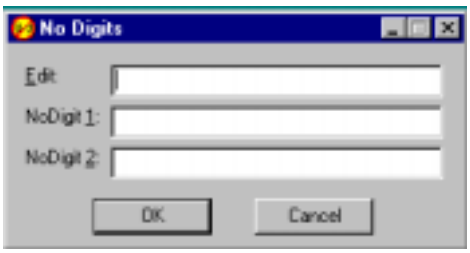


圖 5.1　NoDigits 程式

表 5.1　NoDigits 程式所需檔案

檔案	說明
NoDigits.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
NoDigCls.C	內含 subclass 視窗函式。
NoDigits.RC	內含主視窗的對話盒面板（template）和其圖示（icon）。
Resource.H	內含 NoDigits.RC 檔中所有資源的 ID。
NoDigits.ICO	主視窗圖示（icon）。
NoDigits.MAK	Visual C++ 的 MAK 檔。

當 NoDigits.EXE 一開始執行，會產生一個對話盒，內含三個 edit 控制元件和兩個 buttons。當對話盒函式收到 WM_INITDIALOG 訊息，NoDigits_OnInitDialog 函式便呼叫 NoDigitsClass_ConvertEdit 函式，對兩個 edit 控制元件做 subclassing 動作：

```
NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS1));
NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS2));
```

NoDigitsClass_ConvertEdit 函式實作於 NoDigCls.C 檔中，函式原型則宣告在 NoDigits.C 檔。此函式使用 SubclassWindow 巨集將原視窗函式的位址儲存在 subclassed 視窗的 property 之中。

「Subclass 視窗函式」，也就是 NoDigitsClass_WndProc，只攔截 WM_CHAR 訊息，其它所有訊息皆藉由呼叫 NoDigitsClass_CallOrigWndProc 函式交給原 edit 控制元件的視窗函式處理之。

```
LRESULT WINAPI NoDigitsClass_WndProc (HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam) {

   switch (uMsg) {
      HANDLE_MSG(hwnd, WM_CHAR, NoDigitsClass_OnChar);
   }
   return(NoDigitsClass_CallOrigWndProc(hwnd, uMsg, wParam, lParam));
}
```

NoDigitsClass_CallOrigWndProc 函式動作如下：

```
LRESULT NoDigitsClass_CallOrigWndProc (HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam) {

   // Call the base class's window procedure. It was saved as a window
   // property by the NoDigitsClass_ConvertEdit function.
   return(CallWindowProc(
           (WNDPROC) (DWORD) GetProp(hwnd, g_szNoDigitsClassWndProcOrig),
           hwnd, uMsg, wParam, lParam));
}
```

它的函式原型和一般視窗函式相同，但是它會呼叫 CallWindowProc 函式，將訊息傳給原視窗函式去處理。CallWindowProc 函式的第一個參數表示我們所希望的訊息轉交對象（一個視窗函式位址）。NoDigitsClass_WndProc 函式很容易就可以獲得 edit 控制元件的視窗函式位址 -- 它可以呼叫 GetProp 並放入一個 property 名稱，此名稱（一個字串）與在 NoDigitsClass_ConvertEdit 函式中呼叫 SetProp 時所使用的字串相同。

現在，我們來討論 subclassed edit 控制元件如何阻止數字的輸入。稍早我曾提過，我們的這個 subclass 視窗函式僅對 WM_CHAR 訊息做了一些額外處理。每次當 NoDigitsClass_WndProc 函式收到 WM_CHAR 訊息，便呼叫 NoDigitsClass_OnChar 函式：

```
void NoDigitsClass_OnChar (HWND hwnd, TCHAR ch, int cRepeat) {

   if (adgINRANGE(_TEXT('0'), ch, _TEXT('9'))) {

      // Beep when a digit is received.
      MessageBeep(0);
   } else {

       // Allow nondigits to pass through to the original window procedure.
       FORWARD_WM_CHAR(hwnd, ch, cRepeat, NoDigitsClass_CallOrigWndProc);
   }
}
```

這個函式所做的第一件事是判斷傳入的字元是否爲數字。假如是的話，就呼叫 MessageBeep，意思是告訴使用者說此 edit 視窗不接受數字輸入。如果輸入的不是數字，就使用 FORWARD_WM_CHAR 巨集將 WM_CHAR 訊息轉交給 edit 控制元件的原視窗函式去處理。此巨集的最後一個參數是我們想要將訊息轉交過去的對象（一個視窗函式位址），本例爲 NoDigitsClass_CallOrigWndProc。

譯註：這個 NoDigits 程式雖然不允許使用者在 subclassed edit 視窗中輸入數字，但卻接受剪貼簿的 copy-past 動作，將數字放進去。如果連剪貼簿的 copy-past 動作你也想檢查並攔阻，那麼你必須做更多的處理。

NoDigits.ICO

程式列表 5.1 NoDigits.C

```
#0001  /******************************************************************
#0002  Module name: NoDigits.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates how to subclass a window
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <Windows.h>
#0011  #include <WindowsX.h>
#0012  #pragma warning(disable: 4001)    /* Single-line comment */
#0013  #include "resource.h"
#0014
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017
#0018
#0019  // Function that converts an edit window to a NoDigitsClass window
#0020  // by subclassing the edit window
#0021  BOOL WINAPI NoDigitsClass_ConvertEdit (HWND hwnd, BOOL fSubclass);
#0022
#0023
#0024  ////////////////////////////////////////////////////////////////
#0025
#0026
#0027  BOOL NoDigits_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0028
#0029     adgSETDLGICONS(hwnd, IDI_SUBCLASS, IDI_SUBCLASS);
#0030
#0031     // Turn the regular edit windows into NoDigits windows by subclassing them.
#0032     NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS1), TRUE);
#0033     NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS2), TRUE);
#0034
#0035     return(TRUE);  // Accepts default focus window.
#0036  }
#0037
#0038
#0039  ////////////////////////////////////////////////////////////////
#0040
#0041
#0042  void NoDigits_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0043
#0044     switch (id) {
```

```
#0045       case IDCANCEL:            // Allows dialog box to close.
#0046
#0047          // Turn the NoDigits windows back into regular Edit windows.
#0048          NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS1), FALSE);
#0049          NoDigitsClass_ConvertEdit(GetDlgItem(hwnd, IDC_NODIGITS2), FALSE);
#0050          EndDialog(hwnd, id);
#0051          break;
#0052    }
#0053  }
#0054
#0055
#0056  ////////////////////////////////////////////////////////////////
#0057
#0058
#0059  BOOL WINAPI NoDigits_DlgProc (HWND hwnd, UINT uMsg,
#0060    WPARAM wParam, LPARAM lParam) {
#0061
#0062    switch (uMsg) {
#0063
#0064       // Standard Window's messages
#0065       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, NoDigits_OnInitDialog);
#0066       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    NoDigits_OnCommand);
#0067    }
#0068    return(FALSE);               // We didn't process the message.
#0069  }
#0070
#0071
#0072  ////////////////////////////////////////////////////////////////
#0073
#0074
#0075  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0076    LPSTR lpszCmdLine, int nCmdShow) {
#0077
#0078    adgWARNIFUNICODEUNDERWIN95();
#0079    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_NODIGITS),
#0080       NULL, NoDigits_DlgProc));
#0081
#0082    return(0);
#0083  }
#0084
#0085
#0086  ////////////////////////// End of File ///////////////////////////
```

程式列表 5.2 NoDigCls.C

```
#0001 /******************************************************************
#0002 Module name: NoDigCls.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: NoDigits subclass child control implementation file
#0006 ******************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010 #include <Windows.h>
#0011 #include <WindowsX.h>
#0012 #pragma warning(disable: 4001)   /* Single-line comment */
#0013 #include "resource.h"
#0014
#0015
#0016 ////////////////////////////////////////////////////////////////
#0017
#0018
#0019 static LPCTSTR g_szNoDigitsClassWndProcOrig =
#0020    __TEXT("NoDigitsClassWndProcOrig");
#0021
#0022
#0023 ////////////////////////////////////////////////////////////////
#0024
#0025
#0026 // The sole purpose of this function is to call the base class's window
#0027 // procedure. When using the FORWARD_* message crackers, a function like
#0028 // this one is necessary because the last parameter to a FORWARD_* message cracker
#0029 // is a function with the standard WNDPROC function prototype, whereas
#0030 // CallWindowProc requires a fifth parameter - the address of the window
#0031 // procedure to call.
#0032 LRESULT NoDigitsClass_CallOrigWndProc (HWND hwnd, UINT uMsg,
#0033    WPARAM wParam, LPARAM lParam) {
#0034
#0035    // Call the base class's window procedure.  It was saved as a window
#0036    // property by the NoDigitsClass_ConvertEdit function.
#0037    return(CallWindowProc(
#0038      (WNDPROC) (DWORD) GetProp(hwnd, g_szNoDigitsClassWndProcOrig),
#0039      hwnd, uMsg, wParam, lParam));
#0040 }
#0041
#0042
#0043 ////////////////////////////////////////////////////////////////
#0044
```

```
#0045
#0046  void NoDigitsClass_OnChar (HWND hwnd, TCHAR ch, int cRepeat)  {
#0047
#0048     if (adgINRANGE(__TEXT('0'), ch, __TEXT('9'))) {
#0049
#0050        // Beep when a digit is received.
#0051        MessageBeep(0);
#0052     } else {
#0053
#0054        // Allow nondigits to pass through to the original window procedure.
#0055        FORWARD_WM_CHAR(hwnd, ch, cRepeat, NoDigitsClass_CallOrigWndProc);
#0056     }
#0057  }
#0058
#0059
#0060  ////////////////////////////////////////////////////////////////
#0061
#0062
#0063  // This function processes all messages sent to the NoDigits windows.
#0064  LRESULT WINAPI NoDigitsClass_WndProc (HWND hwnd, UINT uMsg,
#0065     WPARAM wParam, LPARAM lParam) {
#0066
#0067     switch (uMsg) {
#0068        HANDLE_MSG(hwnd, WM_CHAR, NoDigitsClass_OnChar);
#0069     }
#0070     return(NoDigitsClass_CallOrigWndProc(hwnd, uMsg, wParam, lParam));
#0071  }
#0072
#0073
#0074  ////////////////////////////////////////////////////////////////
#0075
#0076
#0077  // Function that converts an edit window to a NoDigitsClass window
#0078  // by subclassing the edit window.
#0079  BOOL WINAPI NoDigitsClass_ConvertEdit (HWND hwnd, BOOL fSubclass) {
#0080
#0081     BOOL fOk = FALSE;
#0082
#0083     if (fSubclass) {
#0084        fOk = SetProp(hwnd, g_szNoDigitsClassWndProcOrig,
#0085          (HANDLE) (DWORD) SubclassWindow(hwnd, NoDigitsClass_WndProc));
#0086     } else {
#0087        WNDPROC wp = (WNDPROC) (DWORD)
#0088           RemoveProp(hwnd, g_szNoDigitsClassWndProcOrig);
#0089       SubclassWindow(hwnd, wp);
#0090       fOk = (wp != NULL);
```

```
#0091    }
#0092    return(fOk);
#0093  }
#0094
#0095
#0096  ////////////////////// End of File ////////////////////////////
```

程式列表 5.3 NoDigits.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017  //
#0018  // Icon
#0019  //
#0020
#0021  IDI_SUBCLASS          ICON    DISCARDABLE     "NoDigits.Ico"
#0022
#0023  ////////////////////////////////////////////////////////////////
#0024  //
#0025  // Dialog
#0026  //
#0027
#0028  IDD_SUBCLASS DIALOG DISCARDABLE  -32768, 5, 192, 79
#0029  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0030  CAPTION "No Digits"
#0031  FONT 8, "MS Sans Serif"
#0032  BEGIN
#0033      LTEXT          "&Edit:",IDC_STATIC,7,8,15,8
#0034      EDITTEXT       IDC_EDIT,43,8,140,13,ES_AUTOHSCROLL
#0035      LTEXT          "NoDigit &1:",IDC_STATIC,7,24,34,8
#0036      EDITTEXT       IDC_NODIGITS1,43,24,140,13,ES_AUTOHSCROLL
```

```
#0037     LTEXT          "NoDigit &2:",IDC_STATIC,7,40,34,8
#0038     EDITTEXT       IDC_NODIGITS2,43,40,140,13,ES_AUTOHSCROLL
#0039     DEFPUSHBUTTON   "OK",1,36,60,50,14
#0040     PUSHBUTTON     "Cancel",2,104,60,50,14
#0041  END
#0042
#0043
#0044  #ifdef APSTUDIO_INVOKED
#0045  ////////////////////////////////////////////////////////////////
#0046  //
#0047  // TEXTINCLUDE
#0048  //
#0049
#0050  1 TEXTINCLUDE DISCARDABLE
#0051  BEGIN
#0052     "resource.h\0"
#0053  END
#0054
#0055  2 TEXTINCLUDE DISCARDABLE
#0056  BEGIN
#0057     "#include ""windows.h""\r\n"
#0058     "\0"
#0059  END
#0060
#0061  3 TEXTINCLUDE DISCARDABLE
#0062  BEGIN
#0063     "\r\n"
#0064     "\0"
#0065  END
#0066
#0067  ////////////////////////////////////////////////////////////////
#0068  #endif    // APSTUDIO_INVOKED
#0069
#0070
#0071  #ifndef APSTUDIO_INVOKED
#0072  ////////////////////////////////////////////////////////////////
#0073  //
#0074  // Generated from the TEXTINCLUDE 3 resource.
#0075  //
#0076
#0077
#0078  ////////////////////////////////////////////////////////////////
#0079  #endif    // not APSTUDIO_INVOKED
```

程式列表 5.4　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by NoDigits.RC
#0004  //
#0005  #define IDD_NODIGITS                104
#0006  #define IDD_SUBCLASS                104
#0007  #define IDI_NODIGITS                105
#0008  #define IDI_SUBCLASS                105
#0009  #define IDC_EDIT                   1000
#0010  #define IDC_NODIGITS1               1004
#0011  #define IDC_NODIGITS2               1005
#0012  #define IDC_STATIC                  -1
#0013
#0014  // Next default values for new objects
#0015  //
#0016  #ifdef APSTUDIO_INVOKED
#0017  #ifndef APSTUDIO_READONLY_SYMBOLS
#0018  #define _APS_NEXT_RESOURCE_VALUE       101
#0019  #define _APS_NEXT_COMMAND_VALUE        40001
#0020  #define _APS_NEXT_CONTROL_VALUE        1002
#0021  #define _APS_NEXT_SYMED_VALUE          101
#0022  #endif
#0023  #endif
```

Window Superclassing 如何運作

視窗的 superclassing 動作和 subclassing 動作有些類似，主要都是以一個新的視窗函式來取代原視窗函式，藉以改變視窗的部份行爲。然而，使用 superclassing 技術，你必須建立一個新的視窗類別，而不是像 subclassing 那樣只是改變單一視窗的行爲。如果你打算產生數個視窗，每一個視窗的行爲都需要稍做改變，那麼 superclassing 技術是你的最佳選擇。Subclassing 和 superclassing 有一些不同點，稍後我再詳加討論。

當你要對一個視窗做 superclassing 動作，你必須註冊一個新的視窗類別。舉個例，如果你的程式需要產生數個 edit 視窗，每一個 edit 視窗都只能夠接受字母（而非數字），你可以先產生所有的 edit 視窗，然後一一爲每個 edit 視窗做 subclassing 動作，達到你所想要的效果；或者，你可以使用 superclassing 技術來完成。

要 superclassing 一個視窗，你必須先建立一個「superclass 視窗函式」，它幾乎和「subclass 視窗函式」一樣。函式原型相同，攔截訊息的方法也相同。事實上，唯一不同的就是你如何呼叫原視窗函式。

這裡有一個例子教你如何建立一個 superclass：

```
// Store the address of the original window procedure.
WNDPROC g_pfnWndProcOrig;
.
.
.
int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
   LPSTR lpszCmdLine, int nCmdShow) {

   WNDCLASSEX wc;
   adgINITSTRUCT(wc, TRUE);

   // Get all the information about the original window class
   GetClassInfoEx(NULL, "EDIT", &wc);

   // Save the original window procedure address so that the
   // Edit_SuperClassWndProc can use it.
   g_pfnWndProcOrig = wc.lpfnWndProc;

   // Our new class must have a new class name.
   wc.lpszClassName = "NoDigits";

   // Our new class is registered by our module.
   wc.hInstance = hinstExe;

   // Our new class has a different window procedure address.
   wc.lpfnWndProc = NoDigitsClass_SuperClassWndProc;

   // Register our new window class.
   RegisterClassEx(&wc);

   // At this point we can create windows of the NoDigits window class.
   // All messages that go to these windows are sent to the
   // NoDigitsClass_SuperClassWndProc first where we can decide if we
   // want to pass them on to g_pfnWndProcOrig.
   .
   .
   .
```

```
}


LRESULT WINAPI NoDigitsClass_SuperClassWndProc (HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam) {

   LRESULT lResult = 0;
   BOOL fCallOrigWndProc = TRUE;


   switch (uMsg) {
      .
      .
      .
   }
   if (fCallOrigWndProc)
      lResult = CallWindowProc(g_pfnWndProcOrig,
         hwnd, uMsg, wParam, lParam);

    return(lResult);
}
```

建立一個 superclass 比建立一個 subclass 還要多出許多步驟。一開始要呼叫 GetClassInfoEx 函式，傳入基礎類別（base class）的名稱（例如 “edit”），以取得其視窗類別的相關資訊。這些資訊將會填入你準備的一個 WNDCLASSEX 結構中，於是你可以做適度修改，用以建立一個新的視窗類別。

一旦有了這基底類別的資訊，接下來你必須將其 lpfnWndProc 欄位的內容儲存起來（通常是存放在全域變數中，稍後我會介紹另一種方法），這個值就是你的基底類別的視窗函式位址。這個變數之後會在「superclass 視窗函式」中被用來作爲 CallWindowProc 函式的第一個參數，這樣才能夠將訊息轉交給基底類別的視窗函式處理。

再來你必許指定一個新的類別名稱，也就是將一個新類別名稱（字串）設定給 lpszClassName 欄位。你還必須將 hInstance 欄位設定爲 hinstExe 或 hinstDll -- 這是 WinMain 函式或 DllMain 函式的第一個參數。這個值主要是讓系統知道行程位址空間中到底是哪一個模組（EXE 或 DLL）正在註冊新的視窗類別。最後，WNDCLASSEX 結構中的 lpfnWndProc 欄位必須設定爲「superclass 視窗函式」的位址。

由於註冊了一個新的視窗類別，所以你可以增加 WNDCLASSEX 結構中的 cbClsExtra 和 cbWndExtra 欄位內容，於是你就可以在你的「superclass 視窗函式」中使用這些額外位元組，這會帶給你莫大的方便。但是，當你在「superclass 視窗函式」中使用類別額外位元組或視窗額外位元組時，要特別小心。基底類別視窗函式在撰寫之初就已做了一項規定，那就是：類別額外位元組從 0 到 cbClsExtra-1，以及視窗額外位元組從 0 到 cbWndExtra-1，歸它自己運用，你的「superclass 視窗函式」不應存取這個範圍內的類別額外位元組和視窗額外位元組 -- 除非你很清楚基底類別如何使用它們。

如果「superclass 視窗函式」要增加類別額外位元組和視窗額外位元組的大小，必須在改變這些值之前，先儲存原來 WNDCLASSEX 結構中的 cbClsExtra 和 cbWndExtra 欄位內容。通常我們把它們儲存在全域變數中。當「superclass 視窗函式」要存取視窗額外位元組時，它必須將索引值加上原 cbWndExtra 值，這樣一來才不會用到基底類別要使用的視窗額外位元組。這裡有一個範例，教你如何在你的「superclass 視窗函式」中存取視窗額外位元組：

```
// Global variables to save the number of class extra bytes, the
// window extra bytes, and the window procedure address of the
// listbox base class
int g_cbClsExtraOrig, g_cbWndExtraOrig;
WNDPROC g_pfnWndProcOrig;

// Index into window extra bytes where our edit data can be found.
// These data follow the data required by the base class.
#define GWL_NODIGITSDATA    (g_cbWndExtraOrig + 0)
.
.
.
ATOM WINAPI NoDigitsClass_RegisterClass (void) {

   WNDCLASSEX wc;
   GetClassInfoEx(NULL, "Edit", &wc);

   // Save the information we need later in global variables.
   g_cbClsExtraOrig = wc.cbClsExtra;
   g_cbWndExtraOrig = wc.cbWndExtra;
   g_pfnWndProcOrig = wc.lpfnWndProc;
```

```
   // Add four window extra bytes to account for our additional edit data.
   wc.cbWndExtra += sizeof(LONG);

   // Change the lpfnWndProc, lpszClassName, and hInstnce members, too.
   .
   .
   .
   // Register the new window class.
   return(RegisterClassEx(&wc));
}


LRESULT WINAPI NoDigitsClass_SuperClassWndProc (HWND hwnd, UINT uMsg,
   WPARAM wParam, LPARAM lParam) {

   int nNoDigitsData;
   .
   .
   .
   // Retrieve our data from the added window extra bytes.
   nNoDigitsData = GetWindowLong(hwnd, GWL_NODIGITSDATA);
   .
   .
   .
   // Call base class window procedure for remainder of processing.
   return(CallWindowProc(g_pfnWndProcOrig,
      hwnd, uMsg, wParam, lParam));
}
```

當然，你也可以使用視窗的 properties 來存放 superclassed 視窗的資料（像 cbClsExtra、cbWndExtra 和原視窗函式位址等等）。然而，使用視窗額外位元組來存放這些資訊比使用視窗 properties 來得好，因爲 properties 需要較多的空間，速度也比較慢。

WNDCLASSEX 結構中的 lpszMenuName 欄位也可以改變，使這個新類別有一份新的選單。如果要使用一份新選單，這份選單的選項識別碼（ID）應該和基底類別中的標準選單的選項識別碼（ID）一致。如果「superclass 視窗函式」完全處理掉所有的 WM_COMMAND 訊息，不將這類訊息轉交給基底類別的視窗函式去處理的話，那麼做出一個新的選單其實是沒有必要的。

WNDCLASSEX 結構中的其餘欄位如 style、hIcon、hCursor、hbrBackground 和 hIconSm

也可以被改變。如果你希望新的視窗類別使用不同的滑鼠游標或不同的圖示（icon），你可以改變 WNCLASEEEX 結構中 hCursor 和 hIcon 欄位。該改的都改了之後，呼叫 RegisterClassEx 註冊這個新類別，然後就可以依此類別建立新的視窗了。

Subclassing 和 superclassing 的主要差別在於：subclassing 改變的是單一視窗的行爲，而 superclassing 改變的是一個類別所屬的所有視窗的行爲。當你意圖修改一個類別並根據它產生出數個視窗，而其視窗行爲與目前存在的這個視窗類別有些許不同的話，你最好是使用 superclassing。是呀，給一個新名稱，註冊新類別，再依此類別產生所有你想要的視窗，比起你使用 SetWindowLong 函式或 SubclassWindow 巨集來修改每一個視窗函式的位址，容易多了！

最常見的運用就是，產生一個對話盒，其中包含數個被 superclassed 的控制元件。當你產生對話盒時，CreateDialogIndirectParam 函式會檢查對話盒面板（template），並根據對話盒面板中的每一個 CONTROL 定義產生對應的視窗。如果面板中有許多 listbox，而其行爲需要修改，我們寧可在每個 CONTROL 敘述句中指定 "NewListBox" 還容易些。如果你選擇使用 subclassing 技術，在你要 subclassing 這些視窗之前，視窗必須都已產生，而且你得在 WM_INITDIALOG 訊息發生時，一次一個地處理那些視窗。過程相當繁瑣。

Superclassing 的另一個好處是，「superclass 視窗函式」可以執行自己的視窗初始化動作。這是因爲在產生視窗之前 Windows 會從視窗類別的記憶體區塊中取得「superclass 視窗函式」，一旦視窗產生，「superclass 視窗函式」便會收到 WM_NCCREATE 和 WM_CREATE 訊息。面對這些訊息，「superclass 視窗函式」可以設定它的類別額外位元組或視窗額位元組，或做任何想做的事情。

不論 superclass 視窗函式是否有處理 WM_NCCREATE 和 WM_CREATE，這兩個訊息都應該被傳遞給基礎類別（base class）的視窗函式處理。Windows 必須爲每一個視窗執行初始化動作以回應 WM_NCCREATE 訊息。如果這個訊息沒有被轉交給原視窗函式，DefWindowProc 函式就不會被呼叫，於是視窗就不會被適當地初始化。藉著將 WM_NCCREATE 訊息交給原視窗函式，我們可以確定 DefWindowProc 函式最後一定會

被呼叫。同樣地，WM_CREATE 訊息也是如此。

不幸的是，在 superclassed 視窗中定義新的視窗訊息，會遭遇先前在 subclassing 時遭遇的相同問題。如果要定義新的視窗訊息，你可以使用 RegisterWindowMessage 函式，要不就是將你想定義的視窗訊息從 WM_APP 開始編碼。

表 **5.2** 概略的說明了 subclassing 和 superclassing 的差異性。

表 5.2　subclassing 和 superclassing 的差別

Subclassing	Superclassing
只有少數視窗需要修改時才使用此一技術。	需要改變同類別的多個視窗行爲時，使用之。
不需註冊新類別。	必須註冊新的視窗類別。
Subclass 視窗函式不能使用任何類別額外位元組或視窗額外位元組。	Superclass 視窗函式可以使用新增的類別額外位元組和視窗額外位元組。
在 subclassing 之前，必須先產生視窗。	superclassing 之前，不需先產生視窗。
Subclassed 視窗無法攔截視窗初始化訊息（WM_NCCREATE 和 WM_CREATE）。	Superclassed 視窗可以攔截視窗初始化訊息（WM_NCCREATE 和 WM_CREATE）。
如果你使用視窗的 properties 來儲存原視窗函式位址，則其執行速度較慢且耗用較多記憶體。	如果你使用全域變數或類別額外位元組來儲存原視窗函式位址，則其執行速度較快並且使用的記憶體比較少。

Arcade 範例實例

Arcade 程式（Arcade.EXE）的原始碼顯示於**程式列表 5.5 ~ 5.12** 中，示範如何藉由 superclassing 標準按鈕類別而建立一個擁有動畫的按鈕類別：AniBtn，。這個新的 AniBtn 類別擁有幾乎和標準按鈕類別一樣的行爲，但它利用一個新的 imagelist 控制元件（譯註：Windows 95 新增）來產生一個有動畫圖示的按鈕，取代原本十分沈悶的文字。當你執行此一程式，主視窗如**圖 5.2** 所示。當然，紙張上的圖不可能顯示動畫效果，你必須執行此程式才能看到動畫。

圖 5.2 Arcade 程式

這個視窗允許使用者選擇一個遊戲來玩。只有兩個遊戲可供選擇：碰碰球（Pong）和俄羅斯方塊（Tetris）。按鈕上的動畫會讓使用者在選擇遊戲之前，就對遊戲有初步的認識。

建立這個程式所需的檔案，列在表 **5.3** 中。

表 5.3　Arcade 程式所需檔案

檔案	說明
Arcade.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
AniBtn.C	內含 superclass 視窗函式。
AniBtn.H	內含 AniBtn.C 中的函式原型宣告，以及 AniBtn 控制元件的訊息巨集。
AniBtn.MSG	MsgCrack 的輸入檔，用來產生 AniBtn.H（請參閱附錄 B）。
SuperCls.C	內含 superclassing 動作的相關函式。你可以將這些函式運用在自己發展的程式中。
SuperCls.H	SuperCls.C 檔中的函式原型的宣告。
Arcade.RC	內含主視窗的對話盒面板（template）和其圖示（icon）。
Resource.H	內含 Arcade.RC 檔中所有資源的 ID。
Arcade.ICO	主視窗圖示（icon）。
Pong1.ICO ~ Pong7.ICO	碰碰球（Pong）動畫按鈕的所有圖示（icons）。
Tetris1.ICO ~ Tetris7.ICO	俄羅斯方塊（Tetris）動畫按鈕的所有圖示（icons）。
AniBtn.MAK	Visual C++ 的 MAK 檔。

當 Arcade.EXE 檔執行起來，WinMain（在 Arcade.C 檔中）首先呼叫 AniBtn_RegisterClass（在 AniBtn.C 檔中）以註冊 AniBtn 視窗類別。Arcade 程式不需要知道 AniBtn 視窗類別實際上是一個系統內建之按鈕類別的 superclass。以這種方法來註冊 AniBtn 類別似乎就像我們在第４章註冊訂製型控制元件一樣。

AniBtn_RegisterClass 函式會傳回新註冊之視窗類別的 atom 值。WinMain 函式會檢查看看這個值是否是 INVALID_ATOM。如果是的話，結束行程。反之假如 AniBtn 類別註冊成功的話，WinMain 會呼叫 DialogBox 來產生程式的使用者介面：一個對話盒。

當你使用 Visual C++，Arcade 程式的對話盒如圖 **5.3** 所示。這個對話盒面板之中包含了三個控制元件：一個 static 控制元件和兩個 AniBtn 控制元件。由於 AniBtn 控制元件是我們自定的視窗類別，所以 Visual C++ 無法顯示太多細節資訊給你。在圖 **5.3** 中，它們是以一個深灰色的矩形表示。當你看到這個自定類別的屬性對話盒（property windows），你會發現其中僅有少許設定（如圖 **5.4**），尤其視窗風格（style 欄）竟以一個 32 位元的十六進位值表示。在 Visual C++ 中使用一個非標準視窗類別，會比較不方便些，不過，至少可以用就是了。

現在，當對話盒及其所有子視窗都產生出來之後，對話盒函式會收到一個 WM_INITDIALOG 訊息，我們以 Arcade_OnInitDialog 函式處理之。在這個函式中我將建立一個影像串列（imagelist），內含 Pong 和 Tetris 的所有圖示（icons），用來讓 Pong 和 Tetris 按鈕產生動畫效果：

```
BOOL Arcade_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lparam) {

  // Create and set the imagelists for both buttons.
  // NOTE: When an AniBtn is destroyed, it destroys the imagelist.
  HINSTANCE hinstRes = GetWindowInstance(hwnd);
  HIMAGELIST himl;

  // Initialize the Pong AniBtn.
  himl = Arcade_MakeImageList(hinstRes, IDI_PONG1, IDI_PONG2,
    IDI_PONG3, IDI_PONG4, IDI_PONG5, IDI_PONG6, 0);
  adgASSERT(himl != NULL);
  AniBtn_SetImageList(GetDlgItem(hwnd, IDC_PONG), himl);
```

```
   AniBtn_SetTimer(GetDlgItem(hwnd, IDC_PONG), 250);

   // Initialize the Tetris AniBtn.
   himl = Arcade_MakeImageList(hinstRes, IDI_TETRIS1, IDI_TETRIS2,
      IDI_TETRIS3, IDI_TETRIS4, IDI_TETRIS5, IDI_TETRIS6, IDI_TETRIS7, 0);
   adgASSERT(himl != NULL);
   AniBtn_SetImageList(GetDlgItem(hwnd, IDI_TETRIS), himl);
   AniBtn_SetTimer(GetDlgItem(hwnd, IDC_TETRIS), 500);

   adgSETDLGICONS(hwnd, IDI_ARCADE, IDI_ARCADE);

   return(TRUE);   // Accept default focus window.
}
```

圖 5.3　Arcade 對話盒面板（template）

圖 5.4　使用者自定（user-defined）類別的屬性視窗（Visual c++ 提供）

建立和初始化影像串列（imagelist）的大部份工作由 Arcade_MakeImageList 函式完成。這是一個參數個數不定的函式，第一個參數是個 HINSTANCE，表示我們將從那裡取得所有圖示（icons）。其餘的參數代表圖示的識別碼（ID），如果 ID 爲 0，表示最後一個。函式會呼叫 ImageList_Craete 來產生 imagelist 控制元件，並根據 Arcade_MakeImageList 參數中的 ID，每次呼叫 LoadIcon 及 ImageList_AddIcon 函式，將圖示（icons）一個個加到影像串列（imagelist）之中。最後，再將影像串列的代碼（handle）傳回給 Arcade_OnInitDialog 函式：

```
HIMAGELIST Arcade_MakeImageList (HINSTANCE hinstRes, int idi, ...) {

   va_list args;
   HIMAGELIST himl = ImageList_Create(
      GetSystemMetrics(SM_CXICON), GetSystemMetrics(SM_CYICON),
      ILC_MASK, 5, 5);
   adgASSERT(himl != NULL);

   va_start(args, idi);
   while (idi != 0) {
      ImageList_AddIcon(himl, LoadIcon(hinstRes, MAKEINTRESOURCE(idi)));
      idi = va_arg(args, int);
   }
   va_end(args);
   return(himl);
}
```

現在，我們可以藉由 AniBtn_SetImageList 巨集（定義在 AniBtn.H 檔中）送出 ABM_SETIMAGELIST 訊息給 AniBtn 視窗，要求將影像串列的 handle 設定給對話盒中的一個控制元件（本例爲 IDC_PONG 或 IDC_TERIS）。同時，我們也必須將圖形交換速度告訴 AniBtn 視窗：我們可以利用 AniBtn_SetTimer 巨集送出 ABM_SETTIMER 訊息，設定影像交換時間。

Arcad.C 之中剩下的唯一一個有趣的函式是 Arcade_OnCommand 函式。當使用者選按 Pong 或 Tetris 動畫按鈕，程式便會呼叫這個函式，並執行使用者所選擇的遊戲。不過 Arcade 程式並沒有設計真正的遊戲，只是顯現出一個訊息視窗（如圖 **5.5**）。

圖 5.5　如果你選擇的是 Pong，會出現這個訊息視窗。

AniBtn 視窗類別

現在讓我們將注意力移轉到 AniBtn 類別及其實現方法。程式碼在 AniBtn.C 檔中，訊息與巨集則定義在 AniBtn.H 檔中。

每一個 AniBtn 視窗都需要一些額外的資料。具體而言是，每一個 AniBtn 視窗需要儲存三項資料：imagelist 代碼（handle）、影像串列中即將顯示的下一個圖示的索引值、影像交換時間的間隔值。如果我們是從頭設計 AniBtn 類別，而不是去 superclassing 一個 button 控制元件，那麼我們所儲存的資料不會有任何問題：我們可以把它們儲存到視窗額外位元組之中。

還記得稍早我曾經說過嗎，當我們 superclassing 一個視窗，我們可以增加類別額外位元組和視窗額外位元組的大小。說來容易做來難呀，你必須絕對確保你不會破壞到任何由基礎類別（base clase）所配置和使用的類別額外位元組以及視窗額外位元組。這可不是件容易的事。不過，我已經寫了一個函式庫，讓這件事情變得比較容易一些。這些函式定義在 SuperCls.C 檔中，宣告於 SuperCls.H 檔中。當你看到一個以 SuperCls_ 開頭的函式，就應該知道這個函式是在 SuperCls.C 檔中。下一節我再詳細討論這個函式庫。

我們在 AniBtn.C 檔中第一個看到的是一個資料結構，定義我們將儲存在 AniBtn「視窗額外位元組」中的資料：

```
typedef struct {
   HIMAGELIST himl;    // Handle of image lList containing images
   int iImage;        // Index of image in himl to show next
   int nTimeout;      // Image change interval
} ANIBTN_WNDEXTRABYTES;
```

Arcade.C 檔中的 WinMain 函式會呼叫 AniBtn_RegisterClass 函式，註冊 AniBtn 視窗類別：

```
ATOM WINAPI AniBtn_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass) {

   WNDCLASSEX wc;
   adgINITSTRUCT(wc, TRUE);
   if (!GetClassInfoEx(NULL, __TEXT("BUTTON"), &wc))
      return(INVALID_ATOM);

   // Give our new class a new name.
   wc.lpszClassName = WC_ANIBTN;

   // Our module is registering the class.
   wc.hInstance = hinst;

   // Make the new class a global class if the user desires.
   if (fGlobalClass)
      wc.style |= CS_GLOBALCLASS;

   // The following WNDCLASSEX members are not changed for AniBtn;
   //     hIcon, hIconSm, hCursor, hbrBackground, lpszMenuName.

   // Register the new window superclass.
   return(SuperCls_RegisterClassEx(&wc, AniBtn_WndProc,
      0, sizeof(ANIBTN_WNDEXTRABYTES)));
}
```

你會發現，這個函式首先呼叫 GetClassInfoEx 以獲得 Windows 內建的 button 視窗類別相關資訊。然後，我們改變類別名稱和 HINSTANCE，再重新註冊這個新的視窗類別。我們也可以加上 CS_GLOBALCLASS 類別風格，讓 AniBtn 成爲一個 ”application global” 類別（如第４章所述）。一切準備就緒後，我們便可以呼叫 SuperCls_RegisterClassEx （定義於 SuperCls.C 函式庫檔案中），註冊 AniBtn 視窗類別。

SuperCls_RegisterClassEx 函式和 Win32 的 RegisterClassEx 函式很類似，只不過它會在註冊前先做一些處理工作，使 AniBtn 程式碼可以輕易使用 superclass 的類別額外位元組和視窗額外位元組。當你呼叫 SuperCls_RegisterClassEx，你必須傳入 WNDCLASSEX 結構體的位址、superclass 視窗函式位址（本例爲 AniBtn_WndProc）、superclass 類別額外位元組和視窗額外位元組的大小。AniBtn 類別不需要任何類別額外位元組（class extra

bytes），但需要一些視窗額外位元組（window extra bytes）-- 大小爲 ANIBTN_WNDEXTRABYTES 資料結構的大小。就像 Win32 的 RegisterClassEx 函式一樣，SuperCls_RegisterClassEx 會傳回被註冊類別的 atom 值；如果註冊不成功的話，則傳回 INVALID_ATOM。傳回值會傳回給呼叫者（本例爲 Arcade 的 WinMain 函式）。

現在，只要產生一個 AniBtn 視窗，所有傳給此視窗的訊息皆會直接交由「superclass 視窗函式」AniBtn_WndProc 來處理。這個函式攔截三個標準視窗訊息：WM_CREATE、WM_DESTROY 和 WM_TIMER。

ANiBtn_OnCreate 函式負責初始化 AniBtn 視窗：

```
BOOL AniBtn_OnCreate (HWND hwnd, LPCREATESTRUCT lpCreateStruct) {

   AniBtn_SetTimer(hwnd, 250);    // Defaults to .25 seconds.

   // Make sure that the button base class is given the opportunity to
   // perform any needed cleanup.

   // NOTE: The return value for WM_CREATE is -1 if the window should not be
   // created and 0 if it should be created. However, the OnCreate message
   // cracker expects TRUE to be returned if the window should be created and
   // FALSE if the window should not be created.
   return(-1 != FORWARD_WM_CREATE(hwnd, lpCreateStruct, AniBtn_CallBaseClass));
}
```

首先它呼叫 AniBtn_SetTimer 函式，使 AniBtn 視窗可以週期性地收到 WM_TIMER 訊息（表示要更換影像）。預設情況是讓計時器每 0.25 秒觸發一次，不過 AniBtn 視窗的使用者也可以經由送出 ABM_SETTIMER 訊息而改變影像更換速度（稍後討論之）。

計時器產生之後，AniBtn 必須完成剩餘的初始化工作。記住，AniBtn 類別是我們將系統的 button 類別經過 superclassing 動作之後得來的，所以大部份的初始化工作還是得交由 button 類別的視窗函式去處理。我們可以利用 FORWARD_WM_CREATE 巨集送出 WM_CREATE 訊息給 AniBtn_CallBaseClass 函式，完成其餘的初始化工作：

```
LRESULT AniBtn_CallBaseClass (HWND hwnd, UINT uMsg,
```

```
   WPARAM wParam, LPARaM lParam) {

   return(CallWindowProc(SuperCls_GetWndProcBaseCls(hwnd),
      hwnd, uMsg, wParam, lParam));
}
```

這個函式的原型必須和一般的視窗函式一樣，因爲 WindowsX.H 所定義的所有 FORWARD_* 巨集都會把你交給它們的四個參數傳給此函式。我們在函式中利用 CallWindowProc 呼叫 button 類別的原視窗函式。其第一個參數是視窗函式位址 -- 再一次，我使用 SuperCls.C 函式庫中的 SuperCls_GetWndProcBaseClass 來幫助我獲得 button 類別的視窗函式位址。而，CallWindowProc 函式其餘的四個參數是標準視窗函式的參數。

當 AniBtn 視窗被摧毀，AniBtn_OnDestroy 函式負責了善後清除的工作：

```
void AniBtn_OnDestroy (HWND hwnd) {

   // If there is an imagelist associated with the button, destroy it.
   HIMAGELIST himl = AniBtn_GetImageList(hwnd);
   if (himl != NULL)
      ImageList_Destroy(himl);

   // Kill the timer associated with the button.
   KillTimer(hwnd, 1);

   // Make sure that the button base class is given the opportunity to
   // perform any needed cleanup.
   FORWARD_WM_DESTROY(hwnd, AniBtn_CallBaseClass);
}
```

只有三樣東西需要清理。第一，如果 AniBtn 含有一個影像串列（imagelist），就呼叫 ImageList_Destroy 函式將它清除掉。第二，將計時器清除掉。第三，呼叫基礎類別（本例爲 button 類別）的視窗函式，做它該做的清理工作。AniBtn_OnTimer 函式負責變更按鈕中的圖示以形成動畫效果。每當計時器被觸發，它就會被呼叫。

```
void AniBtn_OnTimer (HWND hwnd, UINT id) {

   HIMAGELIST himl = AniBtn_GetImageList(hwnd);
```

```
   if (himl != NULL) {

      // Get the AniBtn's image index.
      int iImage = SuperCls_GetWindowLong(hwnd,
         adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, iImage));

      // Change the AniBtn's icon.
      HICON hiconOld = (HICON) SendMessage(hwnd, BM_SETIMAGE, IMAGE_ICON,
         (LONG) ImageList_GetIcon(himl, iImage, ILD_NORMAL));

      // Destroy any old icon that is displayed.
      if (hiconOld != NULL)
         DestroyIcon(hiconOld);

      iImage = ++iImage % ImageList_GetImageCount(himl);
      SuperCls_SetWindowLong(hwnd,
         adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, iImage), iImage);
   }
}
```

這個函式首先取得 AniBtn 中的影像串列（imagelist）代碼（handle）。如果 handle 不是 NULL，它就到 AniBtn 的視窗額外位元組中取得接下來要顯示的圖示的索引。取得視窗額外位元組的值是利用 SuperCls_GetWindowLong 函式（稍後會介紹）完成，我們應該傳給它視窗代碼（handle）以及我們所感興趣的 ANIBTN_WNDEXTRABYTES 結構欄位的偏移位置（offset）。

然後，根據前面獲得的圖示索引，我們呼叫 ImageList_GetIcon 函式取得圖示內容，並送出 BM_SETIMAGE 訊息，要求按鈕顯示出這個新圖示。送出這個訊息給按鈕（button），會傳回前一個（目前正在顯示的）圖示代碼，我們可以將它刪除（如果不是 NULL 的話）。

最後，我們增加影像的索引值（如有必要則令它爲 0，以獲得循環顯示的效果），並呼叫 SuperCls_SetWindowLong 將新的索引值儲存到 AniBtn 的「視窗額外位元組」中。

除了攔截三個標準視窗訊息外，還有 AniBtn 類別所定義的四個類別專屬訊息也要處理，它們是：ABM_SETTIMER、ABM_GETTIMER、ABM_SETIMAGELIST 和 ABM_GETIMAGELIST。這些訊息定義在 AniBtn.MSG 檔中。這個檔案需作爲 MsgCrack 工具軟體（請參考附錄 B）的輸入檔，用以產生訊息 API 和訊息剖析器，放到 AniBtn.H

檔中。所有這些訊息都以 WM_APP 爲基底開始定義，都有相關的訊息處理函式，都是使用 SuperCls_SetWindowLong 或 SuperCls_GetWindowLong 函式來改變視窗的相關資料。你可以從程式碼中驗證其細節。

Window Superclassing 函式庫

在 SuperCls.C 檔中，有許多和 superclassing 技術相關的函式，你可以輕易將它們應用在你自己的程式中。當你要 superclassing 一個視窗類別，這些函式會幫助你註冊新的 superclass，協助你增加類別額外位元組和視窗額外位元組。

處理 superclassed 視窗時，必須用到從基礎類別（base class）取得的某些資訊，包括類別額外位元組及視窗額外位元組的大小，以及基礎類別的視窗函式位址。SuperCls_RegisterClassEx 函式爲 superclass 的類別額外位元組增加了 12 個位元組，並將上述三項資料存在那裡（而不是存放在全域變數中）。下列資料結構代表這份資訊：

```
typedef struct {
   WNDPROC pfnWndProcBaseCls;     // Base class's window procedure
   int     cbClsExtraBaseCls;     // Class extra bytes for base class
   int     cbWndExtraBaseCls;     // Window extra bytes for base class
} SUPERCLS_BASECLSINFO;
```

SuperCls_RegisterClassEx 函式如下：

```
ATOM WINAPI SuperCls_RegisterClassEx (
   PWNDCLASSEX pwc,            // lpfnWndClass member is base class WndProc
   WNDPROC pfnWndProcSuperCls,   // Address of superclass WndProc function
   int cbClsAdditional,         // # of class extra bytes for super class
   int cbWndAdditional) {       // # of window extra bytes for super class

   HWND hwnd;
   ATOM atomSuperClass;
   WNDPROC pfnWndProcBaseCls = pwc->lpfnWndProc;
   int cbClsExtraBaseCls     = pwc->cbClsExtra;
   int cbWndExtraBaseCls     = pwc->cbWndExtra;

   // Add to the cbClsExtra member the number of class extra bytes
```

```
// desired by the superclass plus the number of extra bytes
// required by the SUPERCLS_BASECLSINFO structure.
pwc->cbClsExtra += cbClsAdditional + sizeof(SUPERCLS_BASECLSINFO);

// Add to the cbWndExtra member the number of window extra bytes
// desired by the superclass.
pwc->cbWndExtra += cbWndAdditional;

// In order for the superclass window procedure to manipulate
// any of the superclass's class and window extra bytes successfully, the
// SUPERCLS_BASECLSINFO data members in the superclass's class extra bytes
// must be initialized first.  Because there is no Win32 function that
// allows us to change a class's extra bytes without creating a window
// first, we need to hack the solution.  This is done by registering the
// class using DefWindowProc as the window procedure.  After the class
// extra bytes have been initialized, DefWindowProc is replaced by the
// desired superclass window procedure.
pwc->lpfnWndProc = DefWindowProc;

// Register the superclass.
atomSuperClass = RegisterClassEx(pwc);
if (atomSuperClass == INVALID_ATOM)
   return(atomSuperClass);

// Now, we must complete the class's initialization by setting its
// class extra bytes.  Unfortunately, we must have a window handle
// to change a class's extra bytes.  So, we create a window of the class.
// This is a dummy window whose messages are processed by DefWindowProc.
hwnd = CreateWindowEx(0, MAKEINTATOM(atomSuperClass), NULL,
   0, 0, 0, 0, 0, NULL, NULL, pwc->hInstance, NULL);

if (!IsWindow(hwnd)) {

   // If the window could not be created, unregister the superclass
   // and return INVALID_ATOM to the caller.
   UnregisterClass(MAKEINTATOM(atomSuperClass), pwc->hInstance);
   atomSuperClass = INVALID_ATOM;
} else {

   // Initialize the data represented by the SUPERCLS_BASECLSINFO structure.
   SetClassLong(hwnd,
      SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, pfnWndProcBaseCls),
         (LONG) pfnWndProcBaseCls);
   SetClassLong(hwnd,
      SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, cbClsExtraBaseCls),
         cbClsExtraBaseCls);
```

```
      SetClassLong(hwnd,
         SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, cbWndExtraBaseCls),
            cbWndExtraBaseCls);

      // Now, we can set the class's window procedure to point to the
      // desired superclass window procedure.
      SetClassLong(hwnd, GCL_WNDPROC, (LONG) pfnWndProcSuperCls);

      // NOTE: At this point, any windows of this class that are created
      // will have their messages processed by the proper superclass WndProc.

      // Because all the class extra bytes are set, we can destroy the dummy
      // window.  The WM_DESTROY/WM_NCDESTROY messages will be sent to
      // DefWindowProc because the call to SetClassLong above does NOT affect
      // the window procedure address that was associated with the dummy
      // window when it was created.
      DestroyWindow(hwnd);
   }

   // Return the atom of the registered class to the caller.
   return(atomSuperClass);
}
```

要註冊一個新的 superclass，你的應用程式首先必須呼叫 GetClassInfoEx 函式，獲得一個 WNDCLASSEX 結構。然後對它做適度改變（只有 cbClsExtra、cbWndExtra 和 lpfnWndProc 欄位不改，因爲這些值會由 SuperCls_RegisterClassEx 函式來處理）。當 SuperCls_RegisterClassEx 函式被呼叫，第一個參數是 WNDCLASSEX 結構位址，緊跟著的是 superclass 的視窗函式位址，再來是附加的類別額外位元組和視窗額外位元組的大小。

註冊新視窗類別之前，SuperCls_RegisterClassEx 函式先修改 WNDCLASSEX 結構中的 cbClsExtra 和 cbClsExtra 成員，增加你想要增加的位元組個數。此外，爲了儲存 SUPERCLS_BASECLSINFO 結構，類別額外位元組會再加 12 個位元組。這些位元組可從類別額外位元組的尾端開始計算。由於所有以此類別所產生出來的視窗，其類別資訊完全一樣，所以將資訊儲存在類別額外位元組（class extra bytes）中比儲存在視窗額外位元組（window extra bytes）中更有效率。

現在，我們要做的是將類別額外位元組中的 SUPERCLS_BASECLSINFO 結構初始化。我們必須在 superclass 的視窗產生之前，將這些值設定完成。這真是一個問題，因爲 Win32 並沒有提供任何一個函式允許你在尚未產生任何視窗之前就更改其類別的額外位元組。我們必須自己想辦法。

爲了適當初始化類別的額外位元組，在呼叫 Win32 的 RegisterClassEx 函式之前，SuperCls_RegisterClassEx 函式會設定 WNDCLASSEX 結構中的 lpfnWndProc 欄位爲 DefWindowProc。我們都知道，DefWindowProc 函式是一個安全的視窗函式，其能力足以支援視窗的最基本功能，也因此我們知道它不可能去處理任何額外位元組。

SuperCls_RegisterClassEx 函式註冊了視窗類別後，我們便可以呼叫 CreateWindowEx 產生一個傀儡視窗。技術上，這個以 superclass 視窗類別產生出來的視窗，並不會呼叫 superclass 視窗函式來處理訊息，而是交由 DefWindowProc 函式處理。如果這個視窗成功地被產生出來，SuperCls_RegisterClassEx 函式會呼叫三次 SetWindowLong，設定類別額外位元組中的 SUPERCLS_BASECLSINFO 結構。

此時，所有類別額外位元組都已設定完成，但是 superclass 的視窗函式仍然是 DefWindowProc。我們需要將類別的視窗函式位址指向一個 superclass 視窗函式 -- 我們可以呼叫 SetClassLong 並指定 GCL_WNDPROC 以完成這件事情。現在，superclass 的一些前置處理動作都已完成，接下來我們必須將先前所產生的傀儡視窗摧毀掉，並將 superclass 的 atom 值傳回給呼叫者。

SuperCls.C 檔中的其它函式主要用來存取這些初始化過之後的類別額外位元組。舉個例，SuperCls_GetWndProcBaseCls 函式就是用來取得基礎類別的視窗函式位址，動作如下：

譯註：原書第 342 頁，將 SuperCls_GetWndProcBaseCls 函式名稱誤寫爲 SuperCls_GetWndProcBaseClass，這是不對的。我們可從 SuperCls.C 檔中得知正確名稱。

1. 呼叫 GetClassLong，並指定 GCL_CBCLSEXTRA 識別碼，用以取得類別額外位元組的大小。

2. 取得 SUPERCLS_BASECLSINFO 結構在類別額外位元組中的起始偏移位置。由於這個結構放在類別額外位元組的尾端，所以我們可以將步驟 1 中所取得的類別額外位元組大小減去此結構的大小，即獲得 SUPERCLS_BASECLSINFO 結構在類別額外位元組中的起始偏移位置。
3. 取得 SUPERCLS_BASECLSINFO 結構中的 pfnWndProcBaseCls 欄位的偏移位置，並與步驟 2 所取得的偏移位置相加。
4. 呼叫 GetClassLong，傳入步驟 3 相加後的結果，於是獲得基礎類別的視窗函式位址。

存取 superclass 的「類別額外位元組」和「視窗額外位元組」的過程，是相當類似的。每一個可用函式都列在下面：

```
WORD  WINAPI SuperCls_SetClassWord(HWND hwnd, int nIndex, WORD wNewWord);
WORD  WINAPI SuperCls_GetClassWord(HWND hwnd, int nIndex);
DWORD WINAPI SuperCls_SetClassLong(HWND hwnd, int nIndex, DWORD dwNewLong);
DWORD WINAPI SuperCls_GetClassLong(HWND hwnd, int nIndex);
WORD  WINAPI SuperCls_SetWindowWord(HWND hwnd, int nIndex, WORD wNewWord);
WORD  WINAPI SuperCls_GetWindowWord(HWND hwnd, int nIndex);
DWORD WINAPI SuperCls_SetWindowLong(HWND hwnd, int nIndex, DWORD dwNewLong);
DWORD WINAPI SuperCls_GetWindowLong(HWND hwnd, int nIndex);
```

首先我們必須檢驗類別的 SUPERCLS_BASECLSINFO 結構內容，取出類別或視窗的額外位元組的偏移量。這個偏移量會跳過那些保留給基礎視窗類別使用的額外位元組（不管是「類別額外位元組」或是「視窗額外位元組」）。然後，便可以這個偏移量爲基礎，呼叫 Win32 函式來取得和設定額外位元組，並將結果傳回給呼叫者。

上述那些函式大部份僅只是在增減偏移量，所以我不打算在此做太詳細的介紹。

Arcade.ICO

程式列表 5.5　Arcade.C

```
#0001  /*****************************************************************
#0002  Module name: Arcade.c
```

```
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: AniBtn superclass child control implementation file.
#0006  ************************************************************/
#0007
#0008  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0009  #include <Windows.h>
#0010  #include <Windowsx.h>
#0011  #pragma warning(disable: 4001)   /* Single- line comment */
#0012  #include <commctrl.h>
#0013  #include <stdarg.h>
#0014  #include "AniBtn.h"
#0015  #include "resource.h"
#0016
#0017
#0018  ////////////////////////////////////////////////////////////
#0019
#0020
#0021  HIMAGELIST Arcade_MakeImageList (HINSTANCE hinstRes, int idi, ...) {
#0022
#0023     va_list args;
#0024     HIMAGELIST himl = ImageList_Create(
#0025        GetSystemMetrics(SM_CXICON), GetSystemMetrics(SM_CYICON),
#0026        ILC_MASK, 5, 5);
#0027     adgASSERT(himl != NULL);
#0028
#0029     va_start(args, idi);
#0030     while (idi != 0) {
#0031        ImageList_AddIcon(himl, LoadIcon(hinstRes, MAKEINTRESOURCE(idi)));
#0032        idi = va_arg(args, int);
#0033     }
#0034     va_end(args);
#0035     return(himl);
#0036  }
#0037
#0038  ////////////////////////////////////////////////////////////
#0039
#0040
#0041  BOOL Arcade_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0042
#0043     // Create and set the imagelists for both buttons.
#0044     // NOTE: When an AniBtn is destroyed, it destroys the imagelist.
#0045     HINSTANCE hinstRes = GetWindowInstance(hwnd);
#0046     HIMAGELIST himl;
#0047
#0048     // Initialize the Pong AniBtn.
```

```
#0049    himl = Arcade_MakeImageList(hinstRes, IDI_PONG1, IDI_PONG2,
#0050       IDI_PONG3, IDI_PONG4, IDI_PONG5, IDI_PONG6, 0);
#0051    adgASSERT(himl != NULL);
#0052    AniBtn_SetImageList(GetDlgItem(hwnd, IDC_PONG), himl);
#0053    AniBtn_SetTimer(GetDlgItem(hwnd, IDC_PONG), 250);
#0054
#0055    // Initialize the Tetris AniBtn.
#0056    himl = Arcade_MakeImageList(hinstRes, IDI_TETRIS1, IDI_TETRIS2,
#0057       IDI_TETRIS3, IDI_TETRIS4, IDI_TETRIS5, IDI_TETRIS6, IDI_TETRIS7, 0);
#0058    adgASSERT(himl != NULL);
#0059    AniBtn_SetImageList(GetDlgItem(hwnd, IDC_TETRIS), himl);
#0060    AniBtn_SetTimer(GetDlgItem(hwnd, IDC_TETRIS), 500);
#0061
#0062    adgSETDLGICONS(hwnd, IDI_ARCADE, IDI_ARCADE);
#0063
#0064    return(TRUE);  // Accepts default focus window.
#0065 }
#0066
#0067
#0068 ////////////////////////////////////////////////////////////////
#0069
#0070
#0071 void Arcade_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0072
#0073    switch (id) {
#0074
#0075       case IDCANCEL:            // Allows dialog box to close.
#0076          EndDialog(hwnd, id);
#0077          break;
#0078
#0079       case IDC_PONG:
#0080          adgMB(__TEXT("Imagine Pong running here."));
#0081          break;
#0082
#0083       case IDC_TETRIS:
#0084          adgMB(__TEXT("Imagine Tetris running here."));
#0085          break;
#0086    }
#0087 }
#0088
#0089
#0090 ////////////////////////////////////////////////////////////////
#0091
#0092
#0093 BOOL WINAPI Arcade_DlgProc (HWND hwnd, UINT uMsg,
#0094    WPARAM wParam, LPARAM lParam) {
```

```
#0095
#0096    switch (uMsg) {
#0097       // Standard Window's messages
#0098       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, Arcade_OnInitDialog);
#0099       adgHANDLE_DLGMSG(hwnd, WM_COMMAND, Arcade_OnCommand);
#0100    }
#0101    return(FALSE);
#0102 }
#0103
#0104
#0105
#0106 ////////////////////////////////////////////////////////////////
#0107
#0108
#0109 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0110    LPSTR lpszCmdLine, int nCmdShow) {
#0111
#0112
#0113 #ifndef WINDOWSNT_COMPATIBILITY
#0114    OSVERSIONINFO VerInfo;
#0115    adgINITSTRUCT(VerInfo, TRUE);
#0116    GetVersionEx(&VerInfo);
#0117    if ((VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT) &&
#0118        (VerInfo.dwMajorVersion <= 3 && VerInfo.dwMinorVersion <= 51)) {
#0119       adgMB(__TEXT("This program may not run properly on Windows NT because ")
#0120          __TEXT("it requires that the BS_BITMAP style be supported by the ")
#0121          __TEXT("Windows' Button class."));
#0122    }
#0123 #endif
#0124
#0125    adgWARNIFUNICODEUNDERWIN95();
#0126
#0127    // Register the AniBtn control class.
#0128    if (AniBtn_RegisterClass(hinstExe, FALSE) != INVALID_ATOM) {
#0129
#0130       adgVERIFY(-1 != DialogBox(hinstExe,
#0131          MAKEINTRESOURCE(IDD_ARCADE), NULL, Arcade_DlgProc));
#0132
#0133       // Unregister window classes.
#0134       AniBtn_UnregisterClass(hinstExe);
#0135    }
#0136    return(0);
#0137 }
#0138
#0139
#0140 ////////////////////////// End of File ///////////////////////////
```

程式列表 5.6 AniBtn.C

```
#0001 /****************************************************************
#0002 Module name: AniBtn.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: AniBtn superclass child control implementation file
#0006 ****************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)   /* Single-line comment */
#0013 #include <commctrl.h>
#0014 #include "SuperCls.h"
#0015 #include "AniBtn.h"
#0016
#0017
#0018 ////////////////////////////////////////////////////////////////
#0019
#0020
#0021 // The imagelist code requires the ComCtl32.lib library.  Because
#0022 // VC++ 2.x doesn't link with ComCtl32.lib by default, I must force it.
#0023 #pragma comment(lib, "ComCtl32.lib")
#0024
#0025
#0026 ////////////////////////////////////////////////////////////////
#0027
#0028
#0029 typedef struct {
#0030    HIMAGELIST himl;            // Handle of imagelist containing images
#0031    int iImage;                // Index of image in himl to show next
#0032    int nTimeout;               // Image change interval
#0033 } ANIBTN_WNDEXTRABYTES;
#0034
#0035
#0036 ////////////////////////////////////////////////////////////////
#0037
#0038
#0039 void WINAPI AniBtn_OnSetTimer (HWND hwnd, int nTimeout) {
#0040
#0041    SuperCls_SetWindowLong(hwnd,
#0042       adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, nTimeout), nTimeout);
```

```
#0043    SetTimer(hwnd, 1, nTimeout, NULL);
#0044 }
#0045
#0046
#0047 ////////////////////////////////////////////////////////////////
#0048
#0049
#0050 int WINAPI AniBtn_OnGetTimer (HWND hwnd) {
#0051
#0052    return(SuperCls_GetWindowLong(hwnd,
#0053       adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, nTimeout)));
#0054 }
#0055
#0056
#0057
#0058 ////////////////////////////////////////////////////////////////
#0059
#0060
#0061 void WINAPI AniBtn_OnSetImageList (HWND hwnd, HIMAGELIST himl) {
#0062
#0063    SuperCls_SetWindowLong(hwnd,
#0064       adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, himl), (LONG) himl);
#0065    SuperCls_SetWindowLong(hwnd,
#0066       adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, iImage), 0);
#0067 }
#0068
#0069
#0070 ////////////////////////////////////////////////////////////////
#0071
#0072
#0073 HIMAGELIST WINAPI AniBtn_OnGetImageList (HWND hwnd) {
#0074
#0075    return((HIMAGELIST) SuperCls_GetWindowLong(hwnd,
#0076       adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, himl)));
#0077 }
#0078
#0079
#0080 ////////////////////////////////////////////////////////////////
#0081
#0082
#0083 LRESULT AniBtn_CallBaseClass (HWND hwnd, UINT uMsg,
#0084    WPARAM wParam, LPARAM lParam) {
#0085
#0086    return(CallWindowProc(SuperCls_GetWndProcBaseCls(hwnd),
#0087       hwnd, uMsg, wParam, lParam));
#0088 }
```

```
#0089
#0090
#0091 ////////////////////////////////////////////////////////////////
#0092
#0093
#0094 BOOL AniBtn_OnCreate (HWND hwnd, LPCREATESTRUCT lpCreateStruct) {
#0095
#0096    AniBtn_SetTimer(hwnd, 250);    // Default to .25 seconds.
#0097
#0098    // Make sure that the button base class is given the opportunity to
#0099    // perform any initialization that it needs to.
#0100    // NOTE: The return value for WM_CREATE is -1 if the window should not be
#0101    // created and 0 if it should be created.  However, the OnCreate message
#0102    // cracker expects TRUE to be returned if the window should be created and
#0103    // FALSE if the window should not be created.
#0104    return(-1 != FORWARD_WM_CREATE(hwnd, lpCreateStruct, AniBtn_CallBaseClass));
#0105 }
#0106
#0107
#0108 ////////////////////////////////////////////////////////////////
#0109
#0110
#0111 void AniBtn_OnDestroy (HWND hwnd) {
#0112
#0113    // If there is an imagelist associated with the button, destroy it.
#0114    HIMAGELIST himl = AniBtn_GetImageList(hwnd);
#0115    if (himl != NULL)
#0116       ImageList_Destroy(himl);
#0117
#0118    // Kill the timer associated with the button.
#0119    KillTimer(hwnd, 1);
#0120
#0121    // Make sure that the button base class is given the opportunity to
#0122    // perform any needed clean up.
#0123    FORWARD_WM_DESTROY(hwnd, AniBtn_CallBaseClass);
#0124 }
#0125
#0126
#0127 ////////////////////////////////////////////////////////////////
#0128
#0129
#0130 void AniBtn_OnTimer (HWND hwnd, UINT id) {
#0131
#0132    HIMAGELIST himl = AniBtn_GetImageList(hwnd);
#0133
#0134    if (himl != NULL) {
```

```
#0135
#0136       // Get the AniBtn's image index.
#0137       int iImage = SuperCls_GetWindowLong(hwnd,
#0138          adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, iImage));
#0139
#0140       // Change the AniBtn's icon.
#0141       HICON hiconOld = (HICON) SendMessage(hwnd, BM_SETIMAGE, IMAGE_ICON,
#0142          (LONG) ImageList_GetIcon(himl, iImage, ILD_NORMAL));
#0143
#0144       // Destroy any old icon that is displayed.
#0145       if (hiconOld != NULL)
#0146          DestroyIcon(hiconOld);
#0147
#0148       iImage = ++iImage % ImageList_GetImageCount(himl);
#0149       SuperCls_SetWindowLong(hwnd,
#0150          adgMEMBEROFFSET(ANIBTN_WNDEXTRABYTES, iImage), iImage);
#0151    }
#0152 }
#0153
#0154
#0155 ////////////////////////////////////////////////////////////////
#0156
#0157
#0158 LRESULT WINAPI AniBtn_WndProc (HWND hwnd, UINT uMsg,
#0159    WPARAM wParam, LPARAM lParam) {
#0160
#0161    switch (uMsg) {
#0162
#0163       // Standard window messages
#0164       HANDLE_MSG(hwnd, WM_CREATE,       AniBtn_OnCreate);
#0165       HANDLE_MSG(hwnd, WM_DESTROY,      AniBtn_OnDestroy);
#0166       HANDLE_MSG(hwnd, WM_TIMER,        AniBtn_OnTimer);
#0167
#0168       // Control-specific messages
#0169       HANDLE_MSG(hwnd, ABM_SETTIMER,     AniBtn_OnSetTimer);
#0170       HANDLE_MSG(hwnd, ABM_GETTIMER,     AniBtn_OnGetTimer);
#0171       HANDLE_MSG(hwnd, ABM_SETIMAGELIST, AniBtn_OnSetImageList);
#0172       HANDLE_MSG(hwnd, ABM_GETIMAGELIST, AniBtn_OnGetImageList);
#0173    }
#0174
#0175    return(AniBtn_CallBaseClass(hwnd, uMsg, wParam, lParam));
#0176 }
#0177
#0178
#0179 ////////////////////////////////////////////////////////////////
#0180
```

```
#0181
#0182 ATOM WINAPI AniBtn_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass) {
#0183
#0184    WNDCLASSEX wc;
#0185    adgINITSTRUCT(wc, TRUE);
#0186    if (!GetClassInfoEx(NULL, __TEXT("BUTTON"), &wc))
#0187       return(INVALID_ATOM);
#0188
#0189    // Calling InitCommonControls isn't strictly necessary because ImageLists
#0190    // are not actually registered window classes, but for "correctness" we
#0191    // call this function here anyway.
#0192    InitCommonControls();
#0193
#0194    // Give our new class a new name.
#0195    wc.lpszClassName = WC_ANIBTN;
#0196
#0197    // Our module is registering the class.
#0198    wc.hInstance = hinst;
#0199
#0200    // Make the new class a global class if the user desires.
#0201    if (fGlobalClass)
#0202       wc.style |= CS_GLOBALCLASS;
#0203
#0204    // The following WNDCLASSEX members are not changed for AniBtn:
#0205    //     hIcon, hIconSm, hCursor, hbrBackground, lpszMenuName.
#0206
#0207    // Register the new window superclass.
#0208    return(SuperCls_RegisterClassEx(&wc, AniBtn_WndProc,
#0209       0, sizeof(ANIBTN_WNDEXTRABYTES)));
#0210 }
#0211
#0212
#0213 ////////////////////////////////////////////////////////////////
#0214
#0215
#0216 BOOL WINAPI AniBtn_UnregisterClass (HINSTANCE hinst) {
#0217
#0218    return(UnregisterClass(WC_ANIBTN, hinst));
#0219 }
#0220
#0221
#0222 ////////////////////// End of File ///////////////////////////
```

程式列表 5.7 AniBtn.H

```
#0001 /*****************************************************************
```

```
#0002  Module name: AniBtn.h
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: AniBtn superclass child control header file
#0006  ******************************************************************/
#0007
#0008
#0009  ////////////////////////// Class Name //////////////////////////////
#0010
#0011
#0012  #define WC_ANIBTNA  "AniBtn"
#0013  #define WC_ANIBTNW L"AniBtn"
#0014
#0015  #ifdef UNICODE
#0016  #define WC_ANIBTN WC_ANIBTNW
#0017  #else
#0018  #define WC_ANIBTN WC_ANIBTNA
#0019  #endif
#0020
#0021
#0022  ////////////////////// Class Registration ///////////////////////////
#0023
#0024
#0025  // Register and unregister the AniBtn window class.
#0026  ATOM WINAPI AniBtn_RegisterClass (HINSTANCE hinst, BOOL fGlobalClass);
#0027  BOOL WINAPI AniBtn_UnregisterClass (HINSTANCE hinst);
#0028
#0029
#0030  ////////////////// Class-Specific Window Styles ////////////////////
#0031
#0032
#0033  // NOTE: Superclassed classes cannot have any class-specific window styles.
#0034
#0035
#0036  ////////////////// Parent Notification Codes ////////////////////
#0037
#0038
#0039  // AniBtn controls have no class-specific notification codes at this time.
#0040
#0041
#0042  ///////////////// Class-Specific Window Messages /////////////////
#0043
#0044
#0045  // NOTE: We must select a value for ABM_FIRSTMSG that is guaranteed not to
#0046  // conflict with other messages.  If the application defines other WM_APP
#0047  // messages, the following line will have to change.  Similarly, if the
```

```
#0048  // application uses other superclasses or subclasses, their message ranges
#0049  // must not overlap!
#0050  #define ABM_FIRSTMSG   (WM_APP + 0)
#0051
#0052  //{{adgMSGCRACK_MESSAGES
#0053
#0054  // Purpose: Sets the timer interval between image changes.
#0055  // wParam:  int nTimeout - New timer interval in milliseconds
#0056  // lParam:  N/A
#0057  // Returns: void
#0058  #define ABM_SETTIMER       (ABM_FIRSTMSG + 0)
#0059
#0060  // Purpose: Gets the timer interval between image changes.
#0061  // wParam:  N/A
#0062  // lParam:  N/A
#0063  // Returns: int - Current timer interval in milliseconds
#0064  #define ABM_GETTIMER       (ABM_FIRSTMSG + 1)
#0065
#0066  // Purpose: Sets the imagelist that contains the set of images.
#0067  // wParam:  HIMAGELIST himl - New imagelist containing images
#0068  // lParam:  N/A
#0069  // Returns: void
#0070  #define ABM_SETIMAGELIST    (ABM_FIRSTMSG + 2)
#0071
#0072  // Purpose: Gets the imagelist that contains the set of images.
#0073  // wParam:  N/A
#0074  // lParam:  N/A
#0075  // Returns: HIMAGELIST - Current imagelist containing images
#0076  #define ABM_GETIMAGELIST    (ABM_FIRSTMSG + 3)
#0077
#0078  //}}adgMSGCRACK_MESSAGES
#0079
#0080
#0081  ////////////////////////// Message APIs //////////////////////////
#0082
#0083
#0084  //{{adgMSGCRACK_APIS
#0085
#0086  #define AniBtn_SetTimer(hwnd, nTimeout) \
#0087    ((void)SendMessage((hwnd), ABM_SETTIMER, (WPARAM)(DWORD)(nTimeout), 0))
#0088
#0089  #define AniBtn_GetTimer(hwnd) \
#0090    ((int )SendMessage((hwnd), ABM_GETTIMER, 0, 0))
#0091
#0092  #define AniBtn_SetImageList(hwnd, himl) \
#0093    ((void)SendMessage((hwnd), ABM_SETIMAGELIST, (WPARAM)(DWORD)(himl), 0))
```

```
#0094
#0095  #define AniBtn_GetImageList(hwnd) \
#0096     ((HIMAGELIST )SendMessage((hwnd), ABM_GETIMAGELIST, 0, 0))
#0097
#0098  //}}adgMSGCRACK_APIS
#0099
#0100
#0101  ////////////////////// Message Crackers ////////////////////////
#0102
#0103
#0104  // The following message crackers are available for use in subclassing and
#0105  // superclassing the AniBtn control.
#0106
#0107  //{{adgMSGCRACK_CRACKERS
#0108
#0109  // void Cls_OnSetTimer (HWND hwnd, int nTimeout)
#0110  #define HANDLE_ABM_SETTIMER(hwnd, wParam, lParam, fn) \
#0111     ((fn)((hwnd), (int)(wParam)), 0)
#0112  #define FORWARD_ABM_SETTIMER(hwnd, nTimeout, fn) \
#0113     (void)((fn)((hwnd), ABM_SETTIMER, (WPARAM)(DWORD)(nTimeout), 0))
#0114
#0115  // int  Cls_OnGetTimer (HWND hwnd)
#0116  #define HANDLE_ABM_GETTIMER(hwnd, wParam, lParam, fn) \
#0117     (LRESULT)(fn)((hwnd))
#0118  #define FORWARD_ABM_GETTIMER(hwnd, fn) \
#0119     (int )((fn)((hwnd), ABM_GETTIMER, 0, 0))
#0120
#0121  // void Cls_OnSetImageList (HWND hwnd, HIMAGELIST himl)
#0122  #define HANDLE_ABM_SETIMAGELIST(hwnd, wParam, lParam, fn) \
#0123     ((fn)((hwnd), (HIMAGELIST)(wParam)), 0)
#0124  #define FORWARD_ABM_SETIMAGELIST(hwnd, himl, fn) \
#0125     (void)((fn)((hwnd), ABM_SETIMAGELIST, (WPARAM)(DWORD)(himl), 0))
#0126
#0127  // HIMAGELIST  Cls_OnGetImageList (HWND hwnd)
#0128  #define HANDLE_ABM_GETIMAGELIST(hwnd, wParam, lParam, fn) \
#0129     (LRESULT)(fn)((hwnd))
#0130  #define FORWARD_ABM_GETIMAGELIST(hwnd, fn) \
#0131     (HIMAGELIST )((fn)((hwnd), ABM_GETIMAGELIST, 0, 0))
#0132
#0133  //}}adgMSGCRACK_CRACKERS
#0134
#0135
#0136  ////////////////////////// End of File //////////////////////////
```

程式列表 5.8　AniBtn.MSG

```
#0001  // Module name: AniBtn.msg
#0002  // Written by: Jeffrey Richter
#0003  // Notices: Copyright (c) 1995 Jeffrey Richter
#0004  // Purpose: 'MsgCrack' input file for AniBtn
#0005
#0006  MessageBase ABM_FIRSTMSG
#0007  MessageClass AniBtn
#0008
#0009  Message ABM_SETTIMER SetTimer \
#0010   - Sets the timer interval between image changes.
#0011  wParam int nTimeout - New timer interval in milliseconds
#0012  .
#0013
#0014  Message ABM_GETTIMER GetTimer \
#0015   - Gets the timer interval between image changes.
#0016  Returns int - Current timer interval in milliseconds
#0017  .
#0018
#0019  Message ABM_SETIMAGELIST SetImageList \
#0020   - Sets the imagelist that contains the set of images.
#0021  wParam HIMAGELIST himl - New imagelist containing images
#0022  .
#0023
#0024  Message ABM_GETIMAGELIST GetImageList \
#0025   - Gets the imagelist that contains the set of images.
#0026  Returns HIMAGELIST - Current imagelist containing images
#0027  .
```

程式列表 5.9 SuperCls.C

```
#0001  /*****************************************************************
#0002  Module name: SuperCls.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Utility functions to help with window superclassing.
#0006  *****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <Windows.h>
#0011  #include <WindowsX.h>
#0012  #pragma warning(disable: 4001)   /* Single-line comment */
#0013  #include "SuperCls.h"
#0014
#0015
```

```
#0016 ////////////////////////////////////////////////////////////////
#0017
#0018
#0019 // Enough class extra bytes are added to the superclass to accommodate
#0020 // the data represented by the following structure. These additional data make it
#0021 // easier for a superclass to add class and window extra bytes of its own to
#0022 // the superclass. After the superclass has been registered, using the
#0023 // functions in this module, the layout of the superclass's extra bytes is as
#0024 // follows:
#0025 //
#0026 // Range:   0 to (cbClsExtraBaseCls - 1)
#0027 // Contents: Base class's class extra bytes
#0028 // Range:   cbClsExtraBaseCls to (GCL_CBCLSEXTRA - SUPERCLS_BASECLSINFO - 1)
#0029 // Contents: Additional class extra bytes desired by superclass
#0030 // Range:   (GCL_CBCLSEXTRA - SUPERCLS_BASECLSINFO) to (GCL_CBCLSEXTTA - 1)
#0031 // Contents: Data represented by the following SUPERCLS_BASECLSINFO structure
#0032 //
#0033 typedef struct {
#0034    WNDPROC pfnWndProcBaseCls;    // Base class's window procedure
#0035    int    cbClsExtraBaseCls;    // Class extra bytes for base class
#0036    int    cbWndExtraBaseCls;    // Window extra bytes for base class
#0037 } SUPERCLS_BASECLSINFO;
#0038
#0039
#0040 ////////////////////////////////////////////////////////////////
#0041
#0042
#0043 // Helper macro is used to get the offset of a SUPERCLS_BASECLSINFO member from
#0044 // within the class's extra bytes.
#0045 #define SuperCls_BaseClsInfoIndex(cbClsTotal, member)   \
#0046    (cbClsTotal - sizeof(SUPERCLS_BASECLSINFO) +       \
#0047    adgMEMBEROFFSET(SUPERCLS_BASECLSINFO, member))
#0048
#0049
#0050 ////////////////////////////////////////////////////////////////
#0051
#0052
#0053 ATOM WINAPI SuperCls_RegisterClassEx (
#0054    PWNDCLASSEX pwc,            // lpfnWndClass member is base class WndProc.
#0055    WNDPROC pfnWndProcSuperCls,   // Address of superclass WndProc function
#0056    int cbClsAdditional,         // # of class extra bytes for super class
#0057    int cbWndAdditional) {       // # of window extra bytes for super class
#0058
#0059    HWND hwnd;
#0060    ATOM atomSuperClass;
#0061    WNDPROC pfnWndProcBaseCls = pwc->lpfnWndProc;
```

```
#0062    int cbClsExtraBaseCls    = pwc->cbClsExtra;
#0063    int cbWndExtraBaseCls    = pwc->cbWndExtra;
#0064
#0065    // Add to the cbClsExtra member the number of class extra bytes
#0066    // desired by the superclass plus the number of extra bytes
#0067    // required by the SUPERCLS_BASECLSINFO structure.
#0068    pwc->cbClsExtra += cbClsAdditional + sizeof(SUPERCLS_BASECLSINFO);
#0069
#0070    // Add to the cbWndExtra member the number of window extra bytes
#0071    // desired by the superclass.
#0072    pwc->cbWndExtra += cbWndAdditional;
#0073
#0074    // In order for the superclass window procedure to manipulate
#0075    // any of the superclass's class and window extra bytes successfully, the
#0076    // SUPERCLS_BASECLSINFO data members in the superclass's class extra bytes
#0077    // must be initialized first.  Because there is no Win32 function that
#0078    // allows us to change a class's extra bytes without creating a window
#0079    // first, we need to hack the solution.  This is done by registering the
#0080    // class using DefWindowProc as the window procedure.  After the class
#0081    // extra bytes have been initialized, DefWindowProc is replaced by the
#0082    // desired superclass window procedure.
#0083    pwc->lpfnWndProc = DefWindowProc;
#0084
#0085    // Register the superclass.
#0086    atomSuperClass = RegisterClassEx(pwc);
#0087    if (atomSuperClass == INVALID_ATOM)
#0088       return(atomSuperClass);
#0089
#0090    // Now, we must complete the class's initialization by setting its
#0091    // class extra bytes.  Unfortunately, we must have a window handle
#0092    // to change a class's extra bytes.  So, we create a window of the class.
#0093    // This is a dummy window whose messages are processed by DefWindowProc.
#0094    hwnd = CreateWindowEx(0, MAKEINTATOM(atomSuperClass), NULL,
#0095       0, 0, 0, 0, 0, NULL, NULL, pwc->hInstance, NULL);
#0096
#0097    if (!IsWindow(hwnd)) {
#0098
#0099       // If the window could not be created, unregister the superclass
#0100       // and return INVALID_ATOM to the caller.
#0101       UnregisterClass(MAKEINTATOM(atomSuperClass), pwc->hInstance);
#0102       atomSuperClass = INVALID_ATOM;
#0103    } else {
#0104
#0105       // Initialize the data represented by the SUPERCLS_BASECLSINFO structure.
#0106       SetClassLong(hwnd,
#0107          SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, pfnWndProcBaseCls),
```

```
#0108            (LONG) pfnWndProcBaseCls);
#0109       SetClassLong(hwnd,
#0110          SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, cbClsExtraBaseCls),
#0111             cbClsExtraBaseCls);
#0112       SetClassLong(hwnd,
#0113          SuperCls_BaseClsInfoIndex(pwc->cbClsExtra, cbWndExtraBaseCls),
#0114             cbWndExtraBaseCls);
#0115
#0116       // Now, we can set the class's window procedure to point to the
#0117       // desired superclass window procedure.
#0118       SetClassLong(hwnd, GCL_WNDPROC, (LONG) pfnWndProcSuperCls);
#0119
#0120       // NOTE: At this point, any windows of this class that are created
#0121       // will have their messages processed by the proper superclass WndProc.
#0122
#0123       // Because all the class extra bytes are set, we can destroy the dummy
#0124       // window.  The WM_DESTROY/WM_NCDESTROY messages will be sent to
#0125       // DefWindowProc because the call to SetClassLong above does NOT affect
#0126       // the window procedure address that was associated with the dummy
#0127       // window when it was created.
#0128       DestroyWindow(hwnd);
#0129    }
#0130
#0131    // Return the atom of the registered class to the caller.
#0132    return(atomSuperClass);
#0133 }
#0134
#0135
#0136 ////////////////////////////////////////////////////////////////
#0137
#0138
#0139 WNDPROC WINAPI SuperCls_GetWndProcBaseCls (HWND hwnd) {
#0140
#0141    // Return the address of the base class's WndProc from the superclass's
#0142    // extra bytes.
#0143    return((WNDPROC) GetClassLong(hwnd,
#0144       SuperCls_BaseClsInfoIndex(GetClassLong(hwnd, GCL_CBCLSEXTRA),
#0145          pfnWndProcBaseCls)));
#0146 }
#0147
#0148
#0149 ////////////////////////////////////////////////////////////////
#0150
#0151
#0152 // Function used internally by Get/SetClassWord/Long functions
#0153 static int SuperCls_ClassByteIndex (HWND hwnd, int nIndex) {
```

```
#0154
#0155    int cbClsExtraIndexBaseCls, cbClsExtraBaseCls;
#0156
#0157    // If nIndex is negative, the caller wants a Win32 predefined class
#0158    // word/long value.
#0159    if (nIndex < 0)
#0160       return(nIndex);
#0161
#0162    // Retrieve index into class extra bytes for the number of class
extra
#0163    // bytes used by the base class.
#0164    cbClsExtraIndexBaseCls =
#0165       SuperCls_BaseClsInfoIndex(GetClassLong(hwnd, GCL_CBCLSEXTRA),
#0166          cbClsExtraBaseCls);
#0167
#0168    // Retrieve number of class extra bytes used by the base class.
#0169    cbClsExtraBaseCls = GetClassWord(hwnd, cbClsExtraIndexBaseCls);
#0170
#0171    // Return (index + number) of class extra bytes used by base class.
#0172    return(nIndex + cbClsExtraBaseCls);
#0173 }
#0174
#0175
#0176 ////////////////// Set/GetClassWord/Long Functions /////////////////
#0177
#0178
#0179 WORD WINAPI SuperCls_SetClassWord (HWND hwnd, int nIndex, WORD wNewWord) {
#0180
#0181    return(SetClassWord(hwnd,
#0182       SuperCls_ClassByteIndex(hwnd, nIndex), wNewWord));
#0183 }
#0184
#0185
#0186 WORD WINAPI SuperCls_GetClassWord (HWND hwnd, int nIndex) {
#0187
#0188    return(GetClassWord(hwnd,
#0189       SuperCls_ClassByteIndex(hwnd, nIndex)));
#0190 }
#0191
#0192
#0193 DWORD WINAPI SuperCls_SetClassLong (HWND hwnd, int nIndex, DWORD dwNewLong) {
#0194
#0195    return(SetClassLong(hwnd,
#0196       SuperCls_ClassByteIndex(hwnd, nIndex), dwNewLong));
#0197 }
#0198
```

```
#0199
#0200  DWORD WINAPI SuperCls_GetClassLong (HWND hwnd, int nIndex) {
#0201
#0202     return(GetClassLong(hwnd,
#0203        SuperCls_ClassByteIndex(hwnd, nIndex)));
#0204  }
#0205
#0206
#0207  ////////////////////////////////////////////////////////////
#0208
#0209
#0210  // Function used internally by Get/SetWindowWord/Long functions.
#0211  static int SuperCls_WindowByteIndex (HWND hwnd, int nIndex) {
#0212
#0213     int cbWndExtraIndexBaseCls, cbWndExtraBaseCls;
#0214
#0215     // If nIndex is negative, the caller wants a Win32 predefined window
#0216     // word/long value.
#0217     if (nIndex < 0)
#0218        return(nIndex);
#0219
#0220     // Retrieve index into class extra bytes for the number of window
extra
#0221     // bytes used by the base class.
#0222     cbWndExtraIndexBaseCls =
#0223        SuperCls_BaseClsInfoIndex(GetClassLong(hwnd, GCL_CBCLSEXTRA),
#0224           cbWndExtraBaseCls);
#0225
#0226     // Retrieve number of window extra bytes used by base class.
#0227     cbWndExtraBaseCls = GetClassWord(hwnd, cbWndExtraIndexBaseCls);
#0228
#0229     // Return (index + number) of window extra bytes used by base class.
#0230     return(nIndex + cbWndExtraBaseCls);
#0231  }
#0232
#0233
#0234  ////////////////// Set/GetWindowWord/Long Functions ////////////////
#0235
#0236
#0237  WORD WINAPI SuperCls_SetWindowWord (HWND hwnd, int nIndex, WORD wNewWord) {
#0238
#0239     return(SetWindowWord(hwnd,
#0240        SuperCls_WindowByteIndex(hwnd, nIndex), wNewWord));
#0241  }
#0242
#0243
```

```
#0244  WORD WINAPI SuperCls_GetWindowWord (HWND hwnd, int nIndex) {
#0245
#0246     return(GetWindowWord(hwnd,
#0247        SuperCls_WindowByteIndex(hwnd, nIndex)));
#0248  }
#0249
#0250
#0251  DWORD WINAPI SuperCls_SetWindowLong (HWND hwnd, int nIndex, DWORD dwNewLong) {
#0252
#0253     return(SetWindowLong(hwnd,
#0254        SuperCls_WindowByteIndex(hwnd, nIndex), dwNewLong));
#0255  }
#0256
#0257
#0258  DWORD WINAPI SuperCls_GetWindowLong (HWND hwnd, int nIndex) {
#0259
#0260     return(GetWindowLong(hwnd,
#0261        SuperCls_WindowByteIndex(hwnd, nIndex)));
#0262  }
#0263
#0264
#0265  ////////////////////////// End of File ///////////////////////////
```

程式列表 5.10 SuperCls.H

```
#0001  /*****************************************************************
#0002  Module name: SuperCls.h
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Utility functions to help with window superclassing
#0006  *****************************************************************/
#0007
#0008
#0009  // Function that registers a superclass
#0010  ATOM WINAPI SuperCls_RegisterClassEx (
#0011     PWNDCLASSEX pwc,            // lpfnWndClass member is base class WndProc.
#0012     WNDPROC pfnWndProcSuperCls,   // Address of superclass WndProc function
#0013     int cbClsAdditional,         // # of class extra bytes for superclass
#0014     int cbWndAdditional);        // # of window extra bytes for superclass
#0015
#0016
#0017  // Function to retrieve the WndProc address of the base class
#0018  WNDPROC WINAPI SuperCls_GetWndProcBaseCls (HWND hwnd);
#0019
#0020
```

```
#0021  // Functions to manipulate a superclass's class extra bytes without affecting
#0022  // the base class's class extra bytes
#0023  WORD  WINAPI SuperCls_SetClassWord (HWND hwnd, int nIndex, WORD wNewWord);
#0024  WORD  WINAPI SuperCls_GetClassWord (HWND hwnd, int nIndex);
#0025  DWORD WINAPI SuperCls_SetClassLong (HWND hwnd, int nIndex, DWORD dwNewLong);
#0026  DWORD WINAPI SuperCls_GetClassLong (HWND hwnd, int nIndex);
#0027
#0028
#0029  // Functions to manipulate a superclass's window extra bytes without affecting
#0030  // the base class's window extra bytes.
#0031  WORD  WINAPI SuperCls_SetWindowWord (HWND hwnd, int nIndex, WORD wNewWord);
#0032  WORD  WINAPI SuperCls_GetWindowWord (HWND hwnd, int nIndex);
#0033  DWORD WINAPI SuperCls_SetWindowLong (HWND hwnd, int nIndex, DWORD dwNewLong);
#0034  DWORD WINAPI SuperCls_GetWindowLong (HWND hwnd, int nIndex);
#0035
#0036
#0037  /////////////////////////// End of File /////////////////////////////
```

Pong1.ICO

Pong2.ICO

Pong3.ICO

Pong4.ICO

Pong5.ICO

程式列表 5.11　Arcade.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
```


Pong6.ICO


Tetris1.ICO


Tetris2.ICO


Tetris3.ICO


Tetris4.ICO


Tetris5.ICO


Tetris6.ICO


Tetris7.ICO

```
#0024     "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029     "#include ""windows.h""\r\n"
#0030     "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035     "\r\n"
#0036     "\0"
#0037  END
#0038
#0039  /////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  /////////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_ARCADE DIALOG DISCARDABLE  -32768, 5, 176, 59
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Arcade"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053     CTEXT          "Please select the game that you would like to play:",
#0054                    IDC_STATIC,4,4,168,12
#0055     CONTROL        "Pong",IDC_PONG,"AniBtn",WS_TABSTOP | 0x40,34,21,39,30
#0056     CONTROL        "Tetris",IDC_TETRIS,"AniBtn",WS_TABSTOP | 0x40,102,21,39,
#0057                    30
#0058  END
#0059
#0060
#0061  /////////////////////////////////////////////////////////////////
#0062  //
#0063  // Icon
#0064  //
#0065
#0066  IDI_ARCADE            ICON    DISCARDABLE     "Arcade.ico"
#0067  IDI_PONG1             ICON    DISCARDABLE     "Pong1.ico"
#0068  IDI_PONG2             ICON    DISCARDABLE     "Pong2.ico"
#0069  IDI_PONG3             ICON    DISCARDABLE     "Pong3.ico"
```

```
#0070  IDI_PONG4            ICON    DISCARDABLE    "Pong4.ico"
#0071  IDI_PONG5            ICON    DISCARDABLE    "Pong5.ico"
#0072  IDI_PONG6            ICON    DISCARDABLE    "Pong6.ico"
#0073  IDI_TETRIS1           ICON    DISCARDABLE    "Tetris1.ico"
#0074  IDI_TETRIS2           ICON    DISCARDABLE    "Tetris2.ico"
#0075  IDI_TETRIS3           ICON    DISCARDABLE    "Tetris3.ico"
#0076  IDI_TETRIS4           ICON    DISCARDABLE    "Tetris4.ico"
#0077  IDI_TETRIS5           ICON    DISCARDABLE    "Tetris5.ico"
#0078  IDI_TETRIS6           ICON    DISCARDABLE    "Tetris6.ico"
#0079  IDI_TETRIS7           ICON    DISCARDABLE    "Tetris7.ico"
#0080
#0081  #ifndef APSTUDIO_INVOKED
#0082  ////////////////////////////////////////////////////////////////
#0083  //
#0084  // Generated from the TEXTINCLUDE 3 resource.
#0085  //
#0086
#0087
#0088  ////////////////////////////////////////////////////////////////
#0089  #endif    // not APSTUDIO_INVOKED
```

程式列表 5.12　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by Arcade.rc
#0004  //
#0005  #define IDC_STATIC                  -1
#0006  #define IDD_ARCADE                 105
#0007  #define IDI_ARCADE                 106
#0008  #define IDI_PONG1                  110
#0009  #define IDI_PONG2                  111
#0010  #define IDI_PONG3                  112
#0011  #define IDI_PONG4                  113
#0012  #define IDI_PONG5                  114
#0013  #define IDI_PONG6                  115
#0014  #define IDI_TETRIS1                 130
#0015  #define IDI_TETRIS2                 131
#0016  #define IDI_TETRIS3                 132
#0017  #define IDI_TETRIS4                 133
#0018  #define IDI_TETRIS5                 134
#0019  #define IDI_TETRIS6                 135
#0020  #define IDI_TETRIS7                 136
#0021  #define IDC_PONG                   1010
#0022  #define IDC_TETRIS                 1011
```

```
#0023
#0024  // Next default values for new objects
#0025  //
#0026  #ifdef APSTUDIO_INVOKED
#0027  #ifndef APSTUDIO_READONLY_SYMBOLS
#0028  #define _APS_NEXT_RESOURCE_VALUE       140
#0029  #define _APS_NEXT_COMMAND_VALUE        40001
#0030  #define _APS_NEXT_CONTROL_VALUE        1011
#0031  #define _APS_NEXT_SYMED_VALUE          140
#0032  #endif
#0033  #endif
```

第 6 章

訊息攔截（Hooks）

我認爲 Hook 是 Windows 中最具威力的特性之一，它提供程式一個攔截 events 的能力 -- 不論是針對個別的一個執行緒（thread），或是針對整個 Windows 系統而言。事實上，不只是當某一個 event 發生時程式可以獲得通告，甚至對大部份 hooks 而言，程式還可以告訴 Windows 停止處理這個 event。

Local Hooks 和 Remote Hooks

大致上 hooks 可分爲兩類：local hooks 和 remote hooks。Local hooks 所攔截的是原本預定要給「這個行程中設定此一 hook 之執行緒」的 events，而 remote hooks 所攔截的則是原本要給「其它行程之執行緒」的 events。Local hooks 一次僅能安裝在一個執行緒身上。Remote hooks 有兩種形式：一是針對特定執行緒（thread specific），一是針對整個系統（system-wide）。Thread specific remote hooks 所攔截的是「其它行程中的某個特定執行緒」的 events，而 system wide hooks 所攔截的是「整個系統中所有行程位址空間中的所有執行緒」的 events。

安裝（掛上）local hooks 非常容易，對系統造成的影響極少。然而，remote hooks 就會帶來一些值得注意的不利點：

- **System-wide remote hooks 會影響系統的效率** 任何一個相關 event 發生，Windows 都會通知 system-wide hooks，不管原本該是誰（執行緒）來處理這個 event 。Thread-specific hooks 會對系統造成些許影響，但是其影響遠不及 system-wide hooks 所造成的大。除非真有必要，否則還是儘可能少用 system-wide hooks 爲妙。
- **System-wide hooks 會影響其它程式的穩定性** 如果你寫的是一個有臭蟲的 hook，你會破壞你的行程。如果你寫的是一個有臭蟲的 system-wide hook，你會破壞其它行程。你一定以爲在 32 位元環境下這是不可能的事，因爲一個 Win32 行程不可能影響另一個 Win32 行程，但是 system-wide hooks 卻能夠突破這道防線，讓你所寫的 hook 碼得以在其他行程之內執行。因此，撰寫 system-wide hooks 時，必須格外小心。
- **Remote hooks 必須以 DLL 的方式來撰寫** 所有的 remote hook procedure 都必須存在於 DLL 之中，如此一來 hook 函式碼才能夠被作業系統注射（inject）到其它行程的位址空間中。唯有當 hook 函式碼在 DLL 之中，系統才能夠這麼做。不幸的是，如果只爲了一個小小的 hook 函式就建立一個 DLL，似乎有點小題大作[1] 。

系統呼叫 hook filter function 之前一刻，DLL 的注射動作才會發生，而不是在你呼叫 SetWindowsHookEx 函式時就發生。然而只要你一呼叫 UnhookWindowsHookEx 函式，系統就會準備釋放 DLL，不過還是得等待所有的執行緒完成了在 filter function 中的執行動作，DLL 才會被釋放[2]。

---

[1] 即使你可以將你的 hook 函式放到一個原已存在的 DLL 中，但其實分開來放還是比較好些，也就是說另外產生一個 DLL。這是因爲，就拿 system-wide hooks 的情況來說，包含 hook 函式的整個 DLL（而不單單僅是其中的 hook 函式）都會被注入到系統所有的每一個行程位址空間中，所以你的 hook DLL 愈小愈好。

[2] 關於注射「內含 hook 之 DLL」的動作，請參閱 ***Advanced Windows*** 一書第 16 章， "Breaking Through Process Boundary Wall"。

安裝（掛上）Hooks

Windows 提供了 14 種不同的 hook 型態，每一種都可以是 local 或 remote。但 journal record（WH_JOURNALRECORD）和 journal playback（WH_JOURNALPLAYBACK）兩種 hook 型態除外，它們是 system-wide local hooks，它們不會被注射到任何行程位址空間中。也因此，journal hook 的 filter functions 不一定得放在 DLL 之中。然而，不論是 local hooks 或 remote hooks，安裝和卸除的方法都是一樣的。稍後本章將會詳細探討每一種 hook。

應用程式必須呼叫 SetWindowsHookEx 以安裝一個 hook：

```
HHOOK SetWindowsHookEx(int idHook, HOOKPROC hkprc,
  HINSTANCE hmod, DWORD dwThreadID);
```

當這個函式被呼叫，Windows 會配置一塊記憶體，存放一個內部資料結構，描述即將被安裝的這個新的 hook。所有這些資料結構都會被鏈接在一起，成爲一個串列（linked list），每當有新的節點（新的資料結構）加入，就會被系統放在串列的前端。節點被加上去之後，SetWindowsHookEx 傳回此節點的一個代碼（handle），我們必須將此代碼儲存起來，以備日後參考此一 hook 時所需。稍後我們會看到這樣的例子。

SetWindowsHookEx 函式的第一個參數 idHook，用來表示要掛上哪一種 hook，它必須是表 **6.1** 中的其中一種型態：

表 6.1　Windows 提供的 14 種 hook

Hook 識別字	Windows 呼叫 hook procedure 的時機
1. WH_CALLWNDPROC	每當 SendMessage 送出一個訊息至某個視窗身上時。
2. WH_CALLWNDPROCRET	每當 SendMessage 送出一個訊息後回返（return）時。
3. WH_GETMESSAGE	每當 GetMessage 或 PeekMessage 從已掛上 hook 的執行緒的訊息佇列中取得一個訊息時。
4. WH_KEYBOARD	每當 GetMessage 或 PeekMessage 從已掛上 hook 的執行緒的訊息佇列中取得一個 WM_KEYDOWN 或

Hook 識別字	Windows 呼叫 hook procedure 的時機
	WM_KEYUP 訊息時。
5. WH_MOUSE	每當 GetMessage 或 PeekMessage 從已掛上 hook 的執行緒的訊息佇列中取得滑鼠訊息時。
6. WH_HARDWARE	每當 GetMessage 或 PeekMessage 從已掛上 hook 的執行緒的訊息佇列中取得一個硬體訊息（但不是鍵盤或滑鼠相關訊息）時。
7. WH_MSGFILTER	掛上這個 hook 的執行緒中每當有對話盒、選單或捲動軸正要處理一個被 posted 過來的訊息時。
8. WH_SYSMSGFILTER	任何一個執行緒中每當有對話盒、選單或捲動軸正要處理一個被 posted 的訊息時。
9. WH_JOURNALRECORD	每當有一個訊息從 system's hardware input queue 中被取得（retrieved）時。
10. WH_JOURNALPLAYBACK	每當 Windows 需要自 system's hardware input queue 中提取（requested）一個訊息時。
11. WH_SHELL	一共有５種情況：1. 只要有一個 top-level、unowned 視窗被產生、起作用、或是被摧毀；2. 當 Taskbar 需要重畫某個按鈕；3. 當系統需要「顯示於 Taskbar 上的一個程式」的最小化四方形；4. 當目前的鍵盤佈局狀態改變；5. 當使用者壓下 Ctrl+Esc 去執行 Task Manager（或同級軟體）。
12. WH_CBT	一共有５種情況：1. 每當有一個視窗被 actived、被產生、被摧毀、被最小化、被最大化、被移動或是被改變大小；2. 在完成 system command 之前；3. 在從 system's hardware input queue 中取走滑鼠或鍵盤的 event 之前；4. 在設定輸入焦點或取得 WM_QUEUESYNC 訊息之前。
13. WH_FOREGROUNDIDLE	當前景執行緒呼叫 GetMessage 函式並且佇列中沒有訊息時 -- 也就是說執行緒即將進入睡眠狀態時。
14. WH_DEBUG	每當任何 hook filter function 被呼叫時。

SetWindowsHookEx 函式的第二個參數 hkprc，是 filter function 的位址，用來處理所指定的 hook 型態的訊息。如果你掛上的是一個 remote hook，hkprc 必須位於一個 DLL 之中。

第三個參數 hmod，是 DLL 的模組代碼，此 DLL 是存放 filter function 的地方（如果你掛的是個 remote hook 的話）。如果你掛的是一個 local hook，則 hmod 應爲 NULL。注意，SetWindowsHookEx 函式原型中宣告 hmod 爲一個 HINSTANCE，而不是一個 HMODULE，這雖然看起來有點混亂，但在 Win32 之下不是問題，因爲在 Win32 環境中 HINSTANCEs 和 HMODULEs 沒有差別，是同樣的東西。

最後一個參數 dwThreadID，表示你希望爲此執行緒掛上這個 hook。此參數可用來判斷你的 hook 是 local 還是 remote。如果你所指定的執行緒位在你自己的行程位址空間中，那麼你的 hook 就是 local，否則它就是 remote。如果你要爲目前的執行緒掛上 hook，可以指定 GetCurrentThreadId 函式的傳回值做爲此一參數內容。如果你想掛的是一個 system-wide hook，就請將 dwThreadID 設爲 0（這不是一個有效的執行緒 ID），那麼 hook function 就會攔截系統中所有程式的訊息。如果你想爲特定某個視窗所隸屬的執行緒掛上 hook，你可以將這個參數設定爲 GetWindowThreadProcessId 的傳回值：

```
DWORD GetWindowThreadProcessId(HWND hwnd, LPDWORD lpdwProcessId);
```

當你指定一個視窗代碼，此函式會傳回視窗產生者（一個執行緒）的 ID。你也可以傳入一個 DWORD 的位址做爲第二參數，於是此函式將在該位址上放置擁有此執行緒之行程的 ID。通常我們對行程 ID 不感興趣，所以我們都是將 lpdwProcessID 設爲 NULL。下面這個例子教你如何爲「產生某一視窗」之執行緒掛上一個 hook：

```
DWORD dwThreadID = GetWindowThreadProcessId(hwnd, NULL);
SetWindowsHookEx(WH_KEYBOARD, hkprc, hmod, dwThreadID);
```

大部份 hook 型態，既可以是 local hooks 也可以是 remote hooks。但是有兩個例外：第一，WH_JOURNALRECORD 和 WH_JOURNALPLAYBACK 都是 local system-wide hooks，它們的 hook function（也就是所謂的 filter function）不需放在 DLL 之中。第二，如果 dwProcessID 爲 0，WH_MSGFILTER 其實就是 WH_SYSMSGFILTER（也就是說它們兩個都是 system-wide hook）。當你想要掛上一個 WH_JOURNALRECORD 或是 WH_JOURNALPLAYBACK 或是 WH_SYSMSGFILTER，你必須指定 dwThreadID 爲 0，

否則無法成功掛上 hook。

每當 events 發生，並且與你所掛上的 hook 型別有關聯，Windows 就會呼叫你在 SetWindowsHookEX 函式中所指定的 filter function（hook function）。不論哪一種 hook，filter function 的函式原型都是這樣：

```
LRESULT WINAPI FilterFunc (int nCode, WPARAM wParam, LPARAM lParam);
```

函式名稱 FilterFunc 可以替換爲你所想要的任何名稱。第一個參數 nCode，表示 hook code，其數值範圍依所掛上的 hook 型別而定。傳回值 LRESULT 和參數 wParam 以及 lParam 的意義，視 hook 型別以及 nCode 而定。

Hook filter function 的架構其實和一般的視窗函式很類似。nCode 就像訊息一樣，代表動作的型態，也就是用來通知 filter function 該做什麼樣的處理。wParam 和 lParam 參數的意義則依據 nCode 的不同而不同。稍後我們會探討 nCode 和其 wParam 和 lPram 所代表的意義。

下面的程式片段顯示 WH_KEYBOARD hook 的 filter function 的基本骨幹：

```
// The "g_hhook" global variable will be set to the return value from
// the call to the SetWindowsHookEx function.
static HHOOK g_hhook = NULL;
   .
   .
   .
LRESULT WINAPI Example_KybdHook (int nCode, WPARAM wParam, LPARAM lParam)
{

   LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);

   switch (nCode) {

      case HC_ACTION:

         // Do HC_ACTION processing...
         break;
```

```
      case HC_NOREMOVE:

         // Do HC_NOREMOVE processing...
         break;
   }
   return(lResult);
}
```

卸除 Hook

當程式不再需要 hook，可以利用 UnhookWindowsHookEx　函式卸除之：

```
BOOL UnhookWindowsHookEx(HHOOK hhook);
```

hhook 參數表示要卸除的 hook 的代碼（handle）。這個代碼也就是先前我們呼叫 SetWindowsHookEx 時的傳回值。UnhookWindowsHookEx 函式的傳回值表示 hook 是否成功卸除。如果成功，會傳回一個非 0 值。

Windows 並不要求 hook filter function 的卸除順序一定得和安裝次序相反。每當有一個 hook 被卸除，Windows 便釋放其記憶體區塊，並更新整個 hook 串鏈（hook chain）。

如果有一個執行緒掛上了一個 hook，但是在尚未卸除 hook 之前就結束了，那麼系統會自動爲它做卸除 hook 的動作。

Hook 串鏈（Hook Chains）

當許多程式都安裝了某種型態的 hook 時，就會形成一個 filter-function chain。一旦特定的 event 發生，Windows 會呼叫該型態中最新掛上的 hook filter function。舉個例，如果程式 A 掛上了一個 system-wide WH_KEYBOARD hook，每當有任何執行緒取得鍵盤訊息，Windows 就會呼叫這個 filter function。如果程式 B 也掛上了一個 system-wide WH_KEYBOARD hook，那麼當 event 發生，Windows 不再呼叫程式 A 的 filter function，改呼叫程式 B 的 filter function。這也意味每一個 filter function 有責任確保先前掛上的 filter

function 被呼叫（也就是維護串鏈的完整性）。

SetWindowsHookEx 函式會將新掛上的 hook filter function 的代碼傳回。任何程式只要掛上一個新的 filter function 就必須儲存這個代碼（通常存放在全域變數中）：

```
static HHOOK g_hhook = NULL;
   .
   .
   .
g_hhook = SetWindowsHookEx(WH_KEYBOARD, Example_kybdHook, hinst, NULL);
   .
   .
   .
```

如果有錯誤發生，SetWindowsHookEx 函式會傳回 NULL。

如果你希望 hook chain 中的其它 filter functions 也能夠執行，你可以在你的 filter function 中呼叫 CallNextHookEx 函式（或許你已經在先前的 Example_KybdHook 函式片段中注意到了）[3]：

```
LRESULT CallNextHookEx(HHOOK hhook, int nCode, WPARAM wParam, LPARAM lParam);
```

這個函式會呼叫 filter-function chain 的下一個 filter function，並傳入相同的 nCode、wParam 和 lParam。下一個 filter function 結束之前，應該也遵循這個規則去呼叫 CallNextHookEx 函式，並再次將 hook 代碼（通常那是被放在全域變數中）傳入。CallNextHookEx 函式利用這個 hook 代碼，走訪整個串鏈，決定哪一個 filter function 是下一個呼叫目標。如果 CallNextHookEx 函式發現已經沒有下一個 filter function 可以呼叫（走到串鏈盡頭了），它會傳回 0；否則它就傳回「下一個 filter function 執行後的傳回值」。

[3] 你可能會在許多文件（包括 SDK 文件）中發現一個有關 CallNextHookEx 函式的過氣警告：「如果 nCode 小於 0，則 hook 函式應該不做任何處理，直接將它交給 CallNextHookEx 函式，並傳回 CallNextHookEx 函式的回返值」。這並不是真的，而且自 Windows 3.0 以來（那時還在使用舊版的 SetWindowsHook 函式）就已經不是真的了！撰寫程式時，你可以完全不理會這項警告。

有些時候你可能不希望呼叫其他的 filter functions，這種情況下你只要不在你的 filter function 中呼叫 CallNextHookEx 函式即可。只要不將 CallNextHookEx 函式放到你的 filter function 中，你就不會呼叫其他的 filter functions，而你也因此可以指定你自己的傳回值。不幸的是，這裡埋伏著一個陷阱：另一個執行緒可能也爲你安裝了一個 hook，新的 filter function 於是比你的 filter function 更早被喚起，而它可能不呼叫你的 filter function，完蛋了！這個問題沒有一般性的解決方案，如果你先將自己的 hook 卸除，然後再重新掛上，那麼你的 filter function 就成爲最新的一個，會最先被呼叫。沒錯，但你不能夠保證其他人不會依樣畫葫蘆。簡言之，hooks 是一個合作機制，沒有任何保障。

Windows 的 14 種 hooks

傳遞給 filter function 的參數的意義，視你所掛的 hook 型別的不同而不同。每一種 hook 有它自己最適合處理的一個問題領域。表 **6.1** 所列出的 14 種 Windows hook 將在本節中一一詳細介紹。

1. WH_CALLWNDPROC

2. WH_CALLWNDPROCRET

3. WH_GETMESSAGE

Visual C++ 提供的 Spy++ 可能是最有用的一個除錯工具 。這個工具允許你監看送給某一個視窗（或系統中所有視窗）的訊息。Spy++ 工具就是利用 WH_CALLWNDPROC 和 WH_CALLWNDPROCRET 兩種 hook 完成的，它可以攔截送往某一視窗的所有訊息和其傳回值，並將結果顯示於自己的視窗工作區內[4]。Spy++ 也能攔截「經由 GetMessage 或 PeekMessage 函式」「從執行緒訊息佇列中取得」的所有 posted 訊息 -- 這裡的執行緒是指「被監視視窗」的產生者。Spy++ 之所以能夠這麼做是因爲它掛上了 WH_GETMESSAGE hook。

當一個執行緒呼叫 SendMessage 函式，Windows 會先檢查看看是否有一個 WH_CALLWNDPROC hook 被掛在這個執行緒身上。如果有的話，Windows 會呼叫 hook chain 中最新掛上的 WH_CALLWNDPROC filter function。「呼叫 SendMessage」的那個執行緒也就是「執行 WH_CALLWNDPROC filter function」的那個執行緒。下表列出 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	如果不爲 0，表示訊息是由現行（current）執行緒送出。	指向一個 CWPSTRUCT 結構。	未使用，傳回 0。

HC_ACTION 的意思是，有一個訊息被送到某一個視窗。lParam 指向一個 CWPSTRUCT

[4] 目前的 Spy++ 版本其實並不是掛上 WH_CALLWNDPROCRET hook，因爲 Windows NT 3.5 和 Windows NT 3.51 不支援這個 hook。這個 hook 型態已被加入 Win32 API 之中，未來的 Spy++應該會用此一 hook 來完成。

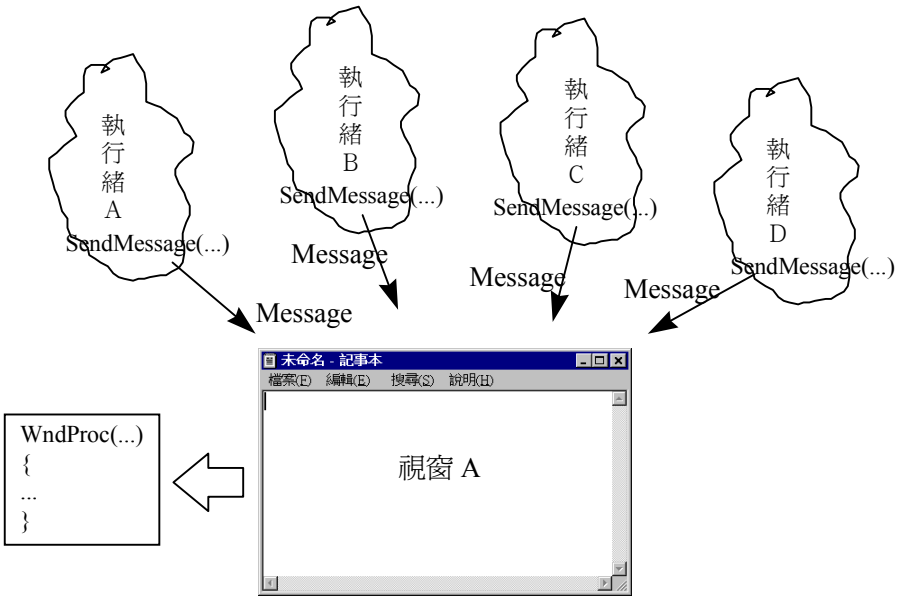
結構，定義於 WinUser.H：

```
typedef struct tagCWPSTRUCT {
    LPARAM  lParam;
    WPARAM  wParam;
    UINT    message;
    HWND    hwnd;
} CWPSTRUCT, *PCWPSTRUCT, NEAR *NPCWPSTRUCT, FAR *LPCWPSTRUCT;
```

由於 Windows 環境中訊息的發送非常頻繁，所以掛上 WH_CALLWNDPROC hooks 會對作業系統效率帶來嚴重的影響。也因爲這樣，所以 WH_CALLWNDPROC hooks 通常僅用於偵錯（debugging）目的。

有一件重要的事必須注意。WH_CALLWNDPROC 的 filter function 被系統呼叫的時機是在「被 hooked 之執行緒」呼叫 SendMessage 函式時，而不是在視窗函式被呼叫（以處理某個訊息）時！這有一點違反直覺！。如果你寫一個 Spy++ 程式，你很自然地會想掛上一個 WH_GETMESSAGE hook，使你能看到所有被 “posted” 給這個視窗的訊息。同時你也一定很自然地會想爲「產生該視窗之執行緒」掛上一個 WH_CALLWNDPROC hook 以便能夠看到所有被 “send” 給該視窗的訊息。然而如果你這麼做的話，你將只會看到視窗之產生者（執行緒）send 給該視窗的訊息。其它執行緒 send 給該視窗的訊息，你的 filter function 都看不到。爲了看到所有被 sent 至此視窗的訊息，你必須掛上一個 system-wide WH_CALLWNDPROC hook 並監看 CWPSTRUCT 結構中的 hwnd 欄位。WH_CALLWNDPROCRET hook 也是一樣。

侯俊傑註：還是讓我用圖來說明吧：



本圖中，執行緒A 是視窗 A 的產生者（所以視窗函式由執行緒 A 來執行）。由於 WH_CALLWNDPROC 的攔截點是在「執行緒呼叫 SendMessage 之時」，並且由於本例設立的是 local hook 而非 system-wide hook，所以圖中只有執行緒 A 送出的訊息會被攔截。如果 WH_CALLWNDPROC 的攔截點是在視窗函式被喚起時，那麼由於本圖每一個執行緒送出的訊息都會喚起視窗函式，所以即使是 local hook，四個訊息也都應該可以被攔截才是。

在 16 位元 Windows 環境中，WH_CALLWNDPROC 的 filter function 可以修改 CWPSTRUCT 結構中的欄位，視窗函式會收到這些修改過後的值，而非原來的參數值。Win32 不允許這樣的事情發生。在 Win32 中，即使 filter function 修改了結構內容，視窗函式仍然會收到原先送給它的參數值。

當 SendMessage 函式回返，Windows 就呼叫最新掛上的 WH_CALLWNDPROCRET filter。「呼叫 SendMessage」的執行緒和「執行 filter」的執行緒是同一個。下表列出此 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	如果不爲 0，表示訊息由現行（current）執行緒送出；否則表示訊息由其它執行緒送出。	指向一個 CWPRETSTRUCT 結構。	未使用，傳回 0。

HC_ACTION 通知 filter function 說，SendMessage 已經回返了。lParam 指向一個 CWPRETSTRUCT 結構，此結構定義在 WinUser.H 中：

```
typedef struct tagCWPRETSTRUCT {
    LRESULT lResult;
    LPARAM  lParam;
    WPARAM  wParam;
    UINT    message;
    HWND    hwnd;
} CWPRETSTRUCT, *PCWPRETSTRUCT, NEAR *NPCWPRETSTRUCT, FAR *LPCWPRETSTRUCT;
```

SendMessage 的傳回值會放在結構中的 lResult 欄位。和 WH_CALLWNDPROC hook 的 filter function 一樣，WH_CALLWNDPROCRET hook 的 filter function 也不能修改 CWPRETSTRUCT 結構的欄位內容。即使 filter function 修改了結構中 lResult 欄位，呼叫 SendMessage 的那個執行緒還是收到由視窗函式所傳回的值。

如果你想在「訊息從執行緒佇列中被取走（經由 GetMessage 或 PeekMessage）時」攔截它，你必須掛上一個 WH_GETMESSAGE hook。呼叫 GetMessage 或 PeekMessage 的那個執行緒就是執行 WH_GETMESSAGE filter function 的執行緒。下表列出 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	NULL	指向一個 MSG 結構。	未使用，傳回 0。

HC_ACTION 通知 filter function 說，訊息已從「被 hooked 之執行緒」的佇列中取得了。當 filter function 收到這項通知時，GetMessage 尚未回返，也因此 DispatchMessage 函式尚未被呼叫。「被 hooked 之執行緒」的訊息迴路中可以做一些處理，過濾掉某些訊息，使這些訊息不被視窗函式處理。像 Spy++ 這樣的程式，並不能夠判斷這個訊息是否已被視窗函式處理過了。

lParam 參數指向一個 MSG 結構，定義於 WinUser.H 檔中：

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

和 WH_CALLWNDPROC 和 WH_CALLWNDPROCRET 的 filter function 不一樣的是，WH_GETMESSAGE 的 filter function 可以修改 MSG 結構的內容。果真這麼做的話，交給「GetMessage 或 PeekMessage 函式呼叫者」的訊息會是改變後的訊息。

4. WH_KEYBOARD

5. WH_MOUSE

6. WH_HARDWARE

WH_KEYBOARD、WH_MOUSE 和 WH_HARDWARE 用來攔截硬體的 events 。圖 6.1 顯示硬體 events 流經系統及各式各樣相關的 hook filter functions 的路線。

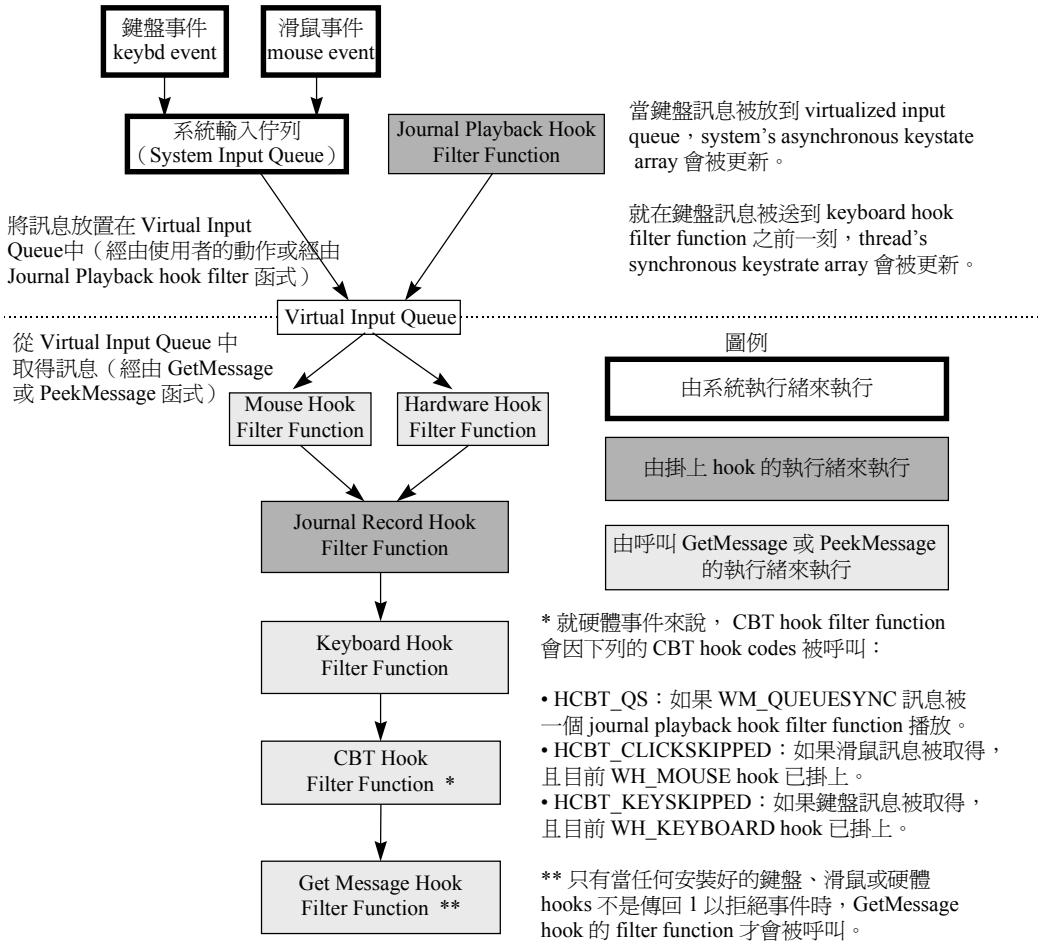


圖 6.1　硬體輸入訊息的流程

當執行緒呼叫 GetMessage 或 PeekMessage 並得到一個硬體輸入訊息，系統會呼叫適當的 hook 來處理它。「呼叫 GetMessage 或 PeekMessage」的執行緒就是「執行 filter function」的執行緒。三個 hooks 有大略相同的形式，不過 lParam 和 wParam 的意義隨著輸入訊息的不同而不同。

下表列出 WH_KEYBOARD filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION 或 HC_NOREMOVE	按鍵虛擬碼（virtual key）	和 WM_KEYDOWN 訊息的 lParam 參數一樣（下一頁）。	如果訊息應該被處理，則傳回 0；如果訊息應該被放棄，則傳回 1。

下表列出 WH_MOUSE filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION 或 HC_NOREMOVE	滑鼠訊息	指向 MOUSEHOOKSTRUCT 結構。	如果訊息應該被處理，則傳回 0；如果訊息應該被放棄，則傳回 1。

下表列出 WH_HARDWARE filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION 或 HC_NOREMOVE	未使用，NULL。	指向 HARDWAREHOOKSTRUCT 結構	如果訊息應該被處理，則傳回 0；如果訊息應該被放棄，則傳回 1。

如果執行緒經由 PeekMessage 函式（並指定 PM_NOREMOVE）取得一個 event ，那麼硬體輸入的 filter function 會收到一個 HC_NOREMOVE hook code。如果訊息真的從執行緒佇列中被移除，filter function 收到的就是 HC_ACTION hook code。

如果程式想要隨時檢驗按鍵狀態，可以利用 WH_KEYBOARD hook。舉個例，Windows

的巨集記錄器（macro recorder）就是靠這個 hook 完成的。這樣的工具程式讓使用者定義一個熱鍵（hot key），藉由熱鍵啓動一系列動作。執行起這個「記錄器」程式，它會監視鍵盤訊息，並將所有按鍵動作錄製成一個巨集。如果掛上去的是一個 system-wide hook，它就可以監看系統中所有的按鍵訊息，甚至即使其他程式正在前景工作。

WH_KEYBOARD 的 filter function，其 wParam 參數表示按鍵按下或放開的虛擬碼（virtual code），lParam 參數則包含了一些與按鍵 event 有關的訊息：

lParam 的各個位元（bits）的說明

0 到 15	重複次數（使用者按住鍵盤不放時，按鍵訊息被重複送出的次數。）
16 到 23	按鍵的掃瞄碼（scan code）。
24	如果按鍵是 extended 鍵，則此位元會被設立。所謂 Extended 鍵是指功能鍵或鍵盤右側的數字鍵。
25 到 26	未使用。
27 到 28	Windows 內部使用。
29	如 Alt 鍵被按住，再按下其他鍵，則這個位元會設爲 on。
30	如果訊息被送出前按下按鍵，則此位元會設爲 on。
31	如果鍵盤被放開，此位元爲 on。如果按鍵被按下，此位元爲 off。

在尚未允許執行緒去處理訊息之前，Windows 不允許 WH_KEYBOARD 的 filter function 任意修改 wParam 或 lParam 參數值。如果你想寫一個 hook filter function 轉換某個鍵盤訊息爲其它訊息的話，你可以使用 WH_GETMESSAGE hook，因爲這個 hook 允許其 filter function 修改訊息。

WH_MOUSE 被應用程式用來檢驗滑鼠移動和滑鼠按鈕狀態，甚至即使程式沒有抓到滑鼠的捕捉權（mouse capture）。舉個例，某公司想製作一項產品來教導使用者如何使用 Windows，當使用者執行這個程式，程式會自動掛上一個 WH_MOUSE hook，並在螢幕上顯示一個小小說明視窗。當使用者移動滑鼠至螢幕某個元素（例如系統選單、縮小盒、捲動軸、標題欄或桌面視窗），filter function 可以判斷滑鼠目前坐落的元素，並顯示一些

輔助文字到說明視窗中。這樣做並不會與正常的滑鼠動作的處理有任何衝突。

對於一個 WH_MOUSE filter function，其 wParam 參數表示滑鼠訊息（如 WM_MOUSEMOVE、WM_LBUTTONDOWN、WM_RBUTTONUP 等等），而 lParam 參數指向一個 MOUSEHOOKSTRUCT 結構，定義如下：

```
typedef struct tagMOUSEHOOKSTRUCT {
    POINT   pt;
    HWND    hwnd;
    UINT    wHitTestCode;
    DWORD   dwExtraInfo;
} MOUSEHOOKSTRUCT, FAR *LPMOUSEHOOKSTRUCT, *PMOUSEHOOKSTRUCT;
```

pt 欄位表示螢幕上的滑鼠座標。hwnd 表示欲接受滑鼠訊息的視窗代碼。wHitTestCode 內含一個 hit test code，用以表示滑鼠目前位於螢幕上的哪一個元素中[5]。

結構的最後一個參數是 dwExtraInfo，記錄著與滑鼠 event 有關的額外資訊，這項資訊通常以 mouse_event 函式來設定，可在視窗函式中以 GetMessageExtraInfo 函式取得。

當程式想要檢查滑鼠和鍵盤以外的一些硬體 event 時，可以使用 WH_HARDWARE hook。這個 hook 的存在主要是爲了筆式電腦（pen extension）。

WH_HARDWARE hook 的 filter function 的 lParam 參數是一個指向 HARDWAREHOOKSTRUCT 結構的指標。結構定義如下：

```
typedef struct tagHARDWAREHOOKSTRUCT {
    HWND        hwnd;
    UINT        message;
    WPARAM      wParam;
    LPARAM      lParam;
} HARDWAREHOOKSTRUCT, FAR *LPHARDWAREHOOKSTRUCT, *PHARDWAREHOOKSTRUCT;
```

[5] Hit test codes（HT*）是以 C 語言的列舉（enum）型態定義於 WinUser.H 表頭檔中。如欲更瞭解 hit test code，請參閱 SDK 文件中的 WM_NCHITTEST 訊息。

7. WH_MSGFILTER

8. WH_SYSMSGFILTER（譯註：這兩個合稱為 message filter hook）

一些系統元件（components）的內部有自己的訊息迴路。像對話盒、選單、捲動軸等等都有自己的訊息迴路以獲取訊息。由於這些訊息迴路並不是我們碰觸得到的，所以 Windows 提供了兩個 hooks（WH_MSGFILTER 和 WH_SYSMSGFILTER）幫助我們。每當有一個訊息要被某個系統元件處理了，Windows 就會呼叫最新安裝好的 WH_SYSMSGFILTER filter function。如果這個函式的傳回值是 0，系統就再呼叫最新被掛上的 WH_MSGFILTER filter function；如果前述傳回值不爲 0 的話，就不會去呼叫 WH_MSGFILTER filter function。

WH_MSGFILTER 和 WH_SYSMSGFILTER 兩個 hooks 之間有一點點不同。如果你企圖掛上一個 WH_SYSMSGFILTER hook，但是執行緒 ID 卻給一個非 0 值，那麼安裝動作不會成功，因爲 WH_SYSMSGFILTER 只以 system-wide 的形式出現。也就是說，它們總是 remote hooks，並且一定要寫成 DLL。

下表列出 WH_SYSMSGFILTER 和 WH_MSGFILTER filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
MSGF_DIALOGBOX MSGF_MESSAGEBOX MSGF_MENU MSGF_MOVE MSGF_SIZE MSGF_SCROLLBAR MSGF_NEXTWINDOW MSGF_MAINLOOP	NULL	指向 MSG 結構	如果訊息應該被處理，則傳回 0；如果訊息應該被放棄，則傳回 1。

WH_SYSMSGFILTER 和 WH_MSGFILTER 的 lParam 參數都是個指標，指向一個 MSG 結構，這個結構就是要被視窗函式處理的訊息。Filter function 可以在回返之前任意修改該結構的欄位內容。

過去，當使用者按下 Alt+Tab 或 Alt+Esc 鍵切換視窗時，WH_SYSMSGFILTER 的 filter

function 會收到一個 MSGF_NEXTWINDOW code。由於 Windows（譯註：作者指的是 Windows 95 和 Windows NT）的硬體輸入訊息的處理機制已改爲非同步模式，所以 MSGF_NEXTWINDOW code 也就不再被送出了，但它仍被我放在前一個表格中，因爲 Win32 技術文件中尚未將它除名。不過，是的，你再也不會收到 MSGF_NEXTWINDOW hook code 了。

先前表格中的最後一個項目是 MSGF_MAINLOOP，你可以利用它來告訴 WH_MSGFILTER filter function 說，這個訊息是來自程式的訊息迴路（通常此迴路實作於程式的 WinMain 函式內）。你可以利用 CallMsgFilter 函式辦到這一點：

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0)) {
   if (!CallMsgFilter(&msg, MSGF_MAINLOOP)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
}
```

CallMsgFilter 函式的第一個參數是 MSG 結構位址，其中包含了要被處理的訊息。這個參數會成爲任何已安裝好的 message filter function 的 lParam 參數。第二個參數 MSGF_MAINLOOP，會成爲每一個 message filter function 的 nCode 參數。

有時候，除了正常的訊息迴路（如上所示）之外，你可能還會在你的程式中建立一個 "modal message loop"。你可能也會想要在這個迴路中通知你所掛上的 message filter function。WinUser.H 表頭檔中定義了一個識別字 MSGF_USER，其值爲 4096。任何你想自行定義的其它 MSGF_* 識別字皆需以此值爲基值。不幸的是，如果許多程式都使用這個基值來定義它們自己的 MSGF_* 識別字，就會彼此衝突，於是 system-wide message filter function 將無法分辨到底是什麼型態的訊息迴路正在處理這個訊息。Win32 API 實在需要一個類似 RegisterWindowMessage 這樣的函式，可以讓程式註冊 filter codes。但由於這種函式並不存在，所以 WM_SYSMSGFILTER filter function 絕對不要企圖檢查 CallMsgFilter 函式送來的 user-defined filter code。如果你使用 CallMsgFilter 函式，那麼安裝一個 WH_SYSMSGFILTER hook 並不明智。你應該使用的是 WH_MSGFILTER hook，

才能確保你所收到的 hook code 獨一無二。

第 3 章的 DlgDyn 程式曾使用一項罕見的技巧，讓程式使用者能夠以鍵盤走訪 modeless 對話盒的所有控制元件。通常 modeless 對話盒要求你在訊息迴路中呼叫 IsDialogMessage 函式。這個函式能夠將鍵盤訊息解譯爲對話盒的活動（例如「以 Tab 鍵將焦點移至下一個控制元件」）。不幸的是，在 Windows 中要產生一個 modeless 對話盒並令其擁有者是 modal 對話盒，技術上很困難，因爲其中沒有可（被我們）操作的訊息迴路 -- Modal 對話盒的訊息迴路已經寫死在 DialogBox 函式中了。然而，如果使用 message filter hooks，就可以解決這個問題。以下程式碼（從 DlgDyn.C 檔中擷取出來）告訴你 DlgDyn 程式如何解決這個問題：

```
// This message filter hook function exists to enable the keyboard interface
// for the modeless dialog.
LRESULT WINAPI DlgDyn_MsgFilterProc (int nCode, WPARAM wParam, LPARAM lParam)
{

   // This variable prevents infinite recursion caused by the call to
   // IsDialogMessage below (IsDialogMessage calls any installed message
   // filter hooks as its first order of business).
   static BOOL fRecurse = FALSE;
   LRESULT lResult;

   // If we are called recursively due to IsDialogMessage, we simply return.
   if (fRecurse) {

      // Returning FALSE tells the system that we want it to do its normal
      // processing for this message.  Ie., let IsDialogMessage do its stuff.
      // NOTE: We are not calling CallNextHookEx because we don't want the
      // same message to go through the same hook chain twice (although this
      // hook function will see it twice).
      return(FALSE);
   }

   // Here we unhook our hook and then rehook ourselves to guarantee that our
   // hook procedure is at the front of the hook chain. This is necessary
   // because any hook procedure before us in the hook chain would see each
   // message twice. By installing ourselves at the front of the chain we get
   // the message first the second time it goes through the hook chain and
   // can simply stop the message from continuing through the hook chain when
   // we are recursing because of IsDialogMessage.
```

```
   adgVERIFY(UnhookWindowsHookEx(g_hhookMsgFilter));
   g_hhookMsgFilter = SetWindowsHookEx(WH_MSGFILTER, DlgDyn_MsgFilterProc,
      NULL, GetCurrentThreadId());
   adgASSERT(g_hhookMsgFilter != NULL);

   // Call any other hooks in the chain.  lResult is TRUE if the next hook
   // in the chain doesn't want any additional processing for the message.
   lResult = CallNextHookEx(g_hhookMsgFilter, nCode, wParam, lParam);

   // If the next hook in the chain wants to allow additional processing
   if (!lResult) {

      // Call IsDialogMessage, setting the static variable fRecurse to TRUE
      // to avoid infinite recursion.  If IsDialogMessage returns TRUE, the
      // message was processed and we DO NOT want normal processing for this
      // message.
      if (IsWindow(g_hwndModeless)) {
         fRecurse = TRUE;
         lResult = IsDialogMessage(g_hwndModeless, (PMSG) lParam);
         fRecurse = FALSE;
      }
   }
   return(lResult);
}
```

藉著爲 DlgDyn 程式的主執行緒掛上一個 WH_MSGFILTER hook，我們便可以攔截所有原本在正常情況下只能被 DialogBox 看到的訊息。然後我們便可以呼叫 IsDialogMessage 函式，傳入 modeless 對話盒代碼（儲存在全域變數 g_hwndModeless 之中），於是我們就可以使用鍵盤來控制 modeless 對話盒了。不幸的是這其中隱藏著更多的複雜性，因爲 IsDialogMessage 函式會呼叫任何一個已掛上的 WH_MSGFILTER filter function，導至無窮遞迴（infinite recursion）的情況。我們必須在 filter function 呼叫 IsDialogMessage 函式之前，設定一個靜態旗標值（fRecurse），我們必須在一開始進入 message filter function 時就檢查這個旗標。如果旗標設立，表示我們的 filter function 是因爲遞迴（由於我們呼叫 IsDialogMessage 函式）而引起，所以就直接回返。DlgDyn_MsgFilterProc 函式還採取了一個預警措施：在呼叫 IsDialogMessage 之前，先將自己的 hook 卸除，然後再立即掛上，於是我們的 hook 就跳至 hook chain 的最前端，如此一來就可以確定不會有其他的 hook functions 看到我們遞迴進入 DlgDyn_MsgFilterProc 函式（因爲這可能是它們從來沒有想過的）。

9. WH_JOURNALRECORD

10. WH_JOURNALPLAYBACK（譯註：這兩個合稱為 journal hook）

最常被使用的 hooks 大概就屬 WH_JOURNALRECORD 和 WH_JOURNALPLAYBACK 這兩種了。這些 hooks 可以用來爲應用程式加上巨集（macros）錄製功能。WH_JOURNALPLAYBACK hook 還可以用來播放（play back）你建構在記憶體中的 event 訊息。第 8 章的 SKDemo 程式便是使用這項技術，強迫其它程式接受某個按鍵。

還記得嗎，WH_JOURNALRECORD 和 WH_JOURNALPLAYBACK 兩者都是 local system-wide hooks，所以其 filter functions 不需寫在 DLL 之中。一般而言，當一個 system-wide hooks 被安裝妥當，「處理訊息」的執行緒也就是「執行 filter functions」的執行緒，然而這對於 journal hooks 來說卻不成立，journal hooks 的 filter function 只能被其安裝者（一個執行緒）執行之。這也就是爲什麼 journal hooks 的 filter functions 不需放在 DLL 的原因。

當系統中的任何一個執行緒索求（requested）一個硬體輸入訊息，系統便喚醒掛上 journal playback hook 的那個執行緒，並執行其 filter function。同樣道理，當系統中任何一個執行緒從其虛擬化輸入佇列（virtualized input queue）中取得（retrieved）一個硬體輸入訊息，系統首先會喚醒掛上 journal record hook 的那個執行緒，並執行其 filter function。一旦安裝 journal hook 的那個執行緒完成了它的處理工作，系統才允許取得訊息的那個執行緒接下去處理。

爲什麼對於這些 hooks，系統要設計成上述架構呢？因爲這些 hooks 所攔截的是一些低階的硬體 input events。一個 journal playback hook 事實上是模擬一個坐在電腦前面的使用者的硬體輸入。由於使用者一次只能輸入一個硬體 event ，所以這些 events 的播放也應當是一次一個。爲了確保輸入的順序不變，events 必須僅能被單一執行緒（掛上 journal playback hook 的那個執行緒）取得。同樣情況也適用於「記錄」動作：硬體的 input events 的處理也必須保持其順序，如此一來 hook 才知道這些 events 的正確誕生順序。

如果系統沒有強制 journal hooks 這麼運作的話，可能會有多個執行緒同時執行 journal hook filter function，以至於這個 filter function 沒辦法知道使用者所產生的各種硬體 input events 的順序。舉個例，如果有一個優先權較低的執行緒正準備開始處理按鍵訊息，在按鍵按下之後，它可能被一個優先權較高的執行緒岔斷，該執行緒準備處理一個滑鼠點選（click）訊息。如果有一個以上的執行緒可以執行 journal hook filter functions，可能上述較高優先權的執行緒會先呼叫 journal record filter function -- 儘管滑鼠訊息其實是在鍵盤訊息之後才產生的。

這樣的架構有一個一個潛在缺點。爲了讓「所有對 journal hook filter function 的呼叫動作」能夠被循序（serializing）執行，系統一次只能夠允許一個執行緒處理硬體的 input events。這可能會使整個系統的輸入機制在某種情況下動彈不得。想像一下，如果 Notepad 和 WordPad 兩個程式都在執行，Notepad 處於作用中（active）狀態。現在使用者按下 A 鍵，再按下 Alt+Tab，再按 B 鍵。可以想見當 Notepad 的執行緒收到 A 鍵便進入了一個無窮迴路。

下面就是沒有安裝 journal hook 時所發生的事情。Notepad 收到 A 鍵，便進入它的無窮迴路。然而，當使用者按下 Alt+Tab 鍵，系統的執行緒會介入並使 WordPad 呈現工作狀態（active）。於是 B 鍵訊息跑到 WordPad 的執行緒去了。注意，使用者還是可以使用執行中的任何程式，只有 Notepad 動彈不得，只有 Notepad 不能被使用。

現在，讓我們回顧相同的情節，但這一次假設我們掛上了一個 journal hook。再一次 Notepad 執行緒會收到 A 鍵並進入其無窮迴路。現在，使用者按下 Alt+Tab 鍵，但是 journal hook 並不會處理這個鍵，因爲 Notepad 的執行緒尚未處理完 A 鍵。所以 Alt+Tab 鍵將永遠在系統的輸入佇列中等候，使用者後來再按下的 B 鍵則排在 Alt+Tab 鍵之後。如果 Notepad 執行緒一直未跳出其迴路，系統的「輸入處理機制」最後可能會失效（disbled）。

顯然這是一個嚴重的問題，而微軟公司也爲此問題做了一些努力。當執行緒掛上 journal hook，系統會特別注意 Ctrl+Esc 鍵。如果使用者按下了 Ctrl+Esc 鍵，系統會自動卸除任何已掛上的 journal hooks，並將 WM_CANCELJOURNAL 訊息 "post" 到任何有掛上

journal hook 的執行緒訊息佇列中。如果你想更清楚瞭解 WM_CANCELJOURNAL 訊息與輸入的循序播放，請參閱本章的 Echo、Capture 和 SKDemo 範例程式。

下表列出 WH_JOURNALRECORD hook 的 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	NULL	指向一個 EVENTMSG 結構	未使用，傳回 0。
HC_SYSMODALON	NULL	NULL	未使用，傳回 0。
HC_SYSMODALOFF	NULL	NULL	未使用，傳回 0。

當 WH_JOURNALRECORD filter function 獲得的 hook code 是 HC_ACTION，其 lParam 參數將指向一個 EVENTMSG 結構[6]：

```
typedef struct tagEVENTMSG {
   UINT        message;
   UINT        paramL;
   UINT        paramH;
   DWORD       time;
   HWND        hwnd;
} EVENTMSG, *PEVENTMSGMSG, NEAR *NPEVENTMSGMSG, FAR *LPEVENTMSGMSG;

typedef struct tagEVENTMSG *PEVENTMSG, NEAR *NPEVENTMSG, FAR *LPEVENTMSG;
```

如果你要在程式中實作「巨集記錄」的功能，你的 WH_JOURNALRECORD filter function 應該把它所收到的每一個 EVENTMSG 結構附加到一塊記憶體中。我將在本章的 Echo 範例程式中說明這項技術。

HC_SYSMODALON 和 HC_SYSMODALOFF hook codes 用來通知一個

[6] 我實在無法解釋出現在 WinUser.H 檔案中的 *MSGMSG 這種名稱。這很明顯是一個瘋狂的人的凌亂無序的作品。EVENTMSG 這樣的名稱無疑是比較爲人所喜愛的。

WH_JOURNALRECORD filter function 說，現在有一個 system-modal 訊息盒顯現或取消。如果有一個 Win32 程式呼叫 MessageBox 函式並指定 MB_SYSTEMMODAL 旗標，在 Windows 95 和 Windows NT 之下產生出來的訊息盒會擁有 WS_EX_TOPMOST 視窗風格；但它並不是 system-modal，你可以輕易切換到其它 16 位元程式或 Win32 程式。然而 Windows 95 和 Windows NT 兩者在處理 16 位元程式所產生的 system-modal 訊息盒時，方式就大大不同了。在 Windows 95 之中，16 位元程式所產生的 system-modal 訊息盒就真的是 system-modal，它會暫停其它所有程式（包括 16 位元程式和 32 位元程式）的執行。在 Windows NT 之中，16 位元程式所產生的 system-modal 訊息盒只能夠暫停同一位址空間中的其它正在執行的 16 位元程式，它不會對 Win32 程式造成任何影響，也不會影響到不同位址空間中正在執行的 16 位元程式。

因爲上面的原因，除非 Windows 95 之中的一個 16 位元程式顯示一個 system-modal 訊息盒，否則一個 WM_JOURNALRECORD filter function 絕對不會收到 HC_SYSMODALON 或 HC_SYSMODALOFF hook codes。由於愈來愈多 16 位元程式移植到 Win32 環境，所以 system-modal 訊息盒的問題也就愈來愈少了，但你仍然應該重視它。你的 filter function 應該監視 WH_SYSMODALON hook code，並在它產生之時停止將 EVENTMSG 結構加到你用來記錄 events 的記憶體區塊中。然後關閉「記錄」功能，並通知使用者說現在有一個 system-modal 訊息盒出現了。本章的 Echo 範例將示範如何實現這個想法。

擁有一系列的 EVENTMSG 結構（或對等的 event 資訊）之後，你可以利用 WH_JOURNALPLAYBACK hook 播放串列中的所有 events。當程式掛上 WH_JOURNALPLAYBACK hook，Windows 會忽略來自使用者的所有滑鼠、鍵盤和筆式輸入，它會呼叫 WH_JOURNALPLAYBACK filter function 企圖取得硬體的 events。當程式掛上了 playback hook，滑鼠將不再能夠影響螢幕上的滑鼠游標位置。下表列出 WH_JOURNALPLAYBACK hook 的 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult

HC_GETNEXT	NULL	指向一個 EVENTMSG 結構。	處理這個訊息之前，系統要等待多少毫秒（milliseconds）
HC_SKIP	NULL	NULL	未使用，傳回 0。
HC_SYSMODALON	NULL	NULL	未使用，傳回 0。
HC_SYSMODALOFF	NULL	NULL	未使用，傳回 0。

Jornal playback filter function 中的 HC_SYSMODALON 和 HC_SYSMODALOFF hook code 的意義與用途和在 jornal record filter function 中是一樣的。

當 playback filter function 收到 HC_GETNEXT hook code，目前的（current）event 所對應的 EVENTMSG 資訊應該被複製到 lParam 參數所指向的那個 EVENTMSG 結構中。HC_GETNEXT hook code 的用途和其名稱並不相配，如果命名爲 “HC_GETCURRENT” 我認爲比較恰當，因爲它並不是要取得你的 events 串列中的下一個 event ，而是要取得目前的（currect）event 。

HC_SKIP hook code 用來通知 WH_JOURNALPLAYBACK filter function 說，Windows 已處理完目前的 event ，filter function 應該準備下一個 event 資訊（或許只要簡單地將 EVENTMSG 結構陣列的指標加 1 即可）。所以每當 Windows 送出 HC_GETNEXT hook code，filter function 應該傳回 events 串列中的 currect event 的資訊；然後當 Windows 送出 HC_SKIP hook code，filter function 應該將目前的指標移至 events 串列中的下一個 event 。當 filter function 收到 HC_SKIP hook code，並且確定所有被儲存起來的 events 都已播放過了，它就可以呼叫 UnhookWindowsHookEx 函式將它自己從 hook chain 中卸除。WH_JOURNALPLAYBACK filter function 不應該在 HC_GETNEXT hook code 處理期間卸除它自己。

當 filter function 在處理 HC_GETNEXT hook code 時回返，EVENTMSG 結構（由 lParam 參數指向之）中的 time 欄位必須內含 event 發生時的系統時間。爲完成這件事情，我建議在 WH_JOURNALRECORD 的 filter function 移除過後立即修改所有被儲存 起來的 events 結構中的 time 欄位。每一個 EVENTMSG 結構中的 time 欄位應該內含自從開始

「記錄」起的時間（以毫秒 milliseconds 爲單位）。下面的程式片段教你如何計算這個值：

```
PEVENTMSG pEventMsg;
int nEvents;
   .
   .
   .
while (nEvents >= 1)
   pEventMsg[--nEvents].time -= pEventMsg[0].time;  // 譯註：原文中爲 pEvent[ ]，應是手誤。
   .
   .
   .
```

其中的 pEventMsg 指向一個 EVENTMSG 結構陣列，而 nEvents 表示已記錄了多少 events。

WH_JOURNALPLAYBACK hook 一被掛上，程式必須立刻將目前的系統時間儲存下來：

```
HHOOK g_hhook;
DWORD g_dwPlaybackStartTime;
   .
   .
   .
g_hhook = SetWindowsHookEx(WH_JOURNALPLAYBACK,
   (HOOKPROC) JrnlPlayBackGook, GetModuleHandle(NULL), 0);
g_dwPlaybackStartTime = GetTickCount();
   .
   .
   .
```

然後，當 HC_GETNEXT hook code 被送到 filter function 時，filter function 可以複製目前的 EVENTMSG 結構到 lParam 參數所指的地方，並改變結構中的 time 欄位，使它成爲正確的播放時間：

```
*((PEVENTMSG) lParam) = *pMySavedEvent;
 ((PEVENTMSG) lParam)->time += g_dwPlaybackStartTime;
```

Filter function 的傳回值表示 Windows 在播放該 event 之前需要等待的時間（毫秒，milliseconds）。如果播放時間已過，filter function 應該傳回 0。下面的程式片段教你如何計算這個值：

```
LRESULT lResult;
.
.
.
switch (nCode) {

   case HC_GETNEXT:

      // Copy current event to the EVENTMSG structure pointed to by lParam.
      *((PEVENTMSG) lParam) = pEventMsg[wCurrentEvent];

      // Update time member in returned structure to desired system
      // playback time.
      ((PEVENTMSG) lParam)->time - GetTickCount():
      if (1Result < 0)
         1Result = 0;
      break;
      .
      .
      .
}
return(1Result);
```

如果你想全速播放被記錄下來的硬體 input events，而不是以記錄時的速度來播放的話，filter function 應該將 time 欄位修改爲目前的系統時間，並傳回 0：

```
LRESULT 1Resu1t;
   .
   .
   .
switch (nCode) (

   case HC_GETMEXT:
     // Copy current event to the EVENTMSG structure pointed to by 1Param.
     *((PEVENTMSG) lParam) = pEventMsg[wCurrentEvent];

     // Update time member in returned structure to desired system
```

```
    // playback time.
    ((PEVENTMSG) lParam)->time = GetTickCount();

    // Return milliseconds Windows should wait before processing the event.
    lResult = 0;
    break;
    .
    .
    .
}
return(lResult);
```

當你記錄或播放巨集時，必須僅記一些事情。首先，請儘量少用滑鼠，這是因爲 WH_JOURNALRECORD 所收到的 EVENTMSG 結構中的滑鼠位置是螢幕座標，而當播放巨集時，螢幕上的視窗佈局可能已經改變（包括位置和大小），導致滑鼠 events 所送往的視窗與原來的不同。此外，巨集也可能在不同的螢幕解析度下播放，於是也導致上述結果。最後一點，如果巨集是以鍵盤 events 被記錄下來，但是在播放時使用了不同的國別設定（鍵盤佈局不同），也會產生不相容的問題。

此外還有一些關於 journal record 和 journal playback hooks 的問題，我會在 Echo 和 Capture 範例程式中一一討論。第 8 章的 SKDemo 程式也會牽扯到這些問題。

11. WH_SHELL

WH_SHELL hook 通常被使用在 shell 程式（例如 Windows 95 的 Taskbar）中，用以得知程式的外觀或位置何時被改變。下表列出 WH_SHELL hook 的 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HSHELL_WINDOWCREATED	被產生的視窗的代碼。	NULL	未使用，傳回 0。
HSHELL_WINDOWDESTROYED	被摧毀的視窗的代碼。	NULL	未使用，傳回 0。
HSHELL_ACTIVATESHELLWINDOW	NULL	NULL	未使用，傳回 0。
HSHELL_WINDOWACTIVATED	被 activated 的視窗的代碼。	如果視窗即將成爲全螢幕，則爲 TRUE	未使用，傳回 0。
HSHELL_GETMINRECT	被最小化、被最大化或被還原的視窗的代碼。	RECT 結構位址，該處用來接收座標	未使用，傳回 0。
HSHELL_REDRAW	標題欄或圖示（icon）的視窗代碼	如果視窗標題被重畫，則爲 TRUE。	未使用，傳回 0。
HSHELL_TASKMAN	NULL	NULL	傳回 TRUE 以阻止系統顯示它自己的 “task list”。
HSHELL_LANGUAGE	NULL	NULL	未使用，傳回 0。

每當產生一個最上層的（top-level）、無人擁有（unowned）的視窗，CreateWindowEx 函式會去呼叫最新掛上的 WH_SHELL filter function 並傳入一個 hook code：HSHELL_WINDOWCREATED。每當有一個最上層的（top-level）、無人擁有的（unowned）視窗要被摧毀，DestroyWindow 函式便會去呼叫 WH_SHELL filter function，並傳入一個

hook code：HSHELL_WINDOWDESTROYED。除了這兩種情況之外，還有三種不常見的情況也會發出 HSHELL_WINDOW* hook code：

- 隱藏一個最上層（top-level）、無人擁有（unowned）的視窗，會發出 HSHELL_WINDOWDESTROYED hook code。
- 顯示一個最上層（top-level）、無人擁有（unowned）的視窗，會發出 HSHELL_WINDOWCREATED filter code。
- 改變一個最上層（top-level）視窗的父視窗（不論是使用 SetParent 函式，或是設定 GWL_HWNDPARENT），會先發出 HSHELL_WINDOWDESTROYED hook code，接著再發出 HSHELL_WINDOWCREATED hook code。

被產生或被摧毀的視窗的視窗代碼放在 wParam 參數中，如果你喜歡，你可以 "send" 一個訊息給該視窗。本章稍後的 AppLog 程式就是掛上了一個 system-wide WH_SHELL hook，然後利用 HSHELL_WINDOWCREATED 和 HSHELL_WINDOWDESTROYED 兩個 hook codes，追蹤記錄你所啓動或摧毀的任何程式。

HSHELL_ACTIVATESHELLWINDOW 是 16 位元 Windows 遺留下來的 hook code，在 32 位元環境中你將不會再收到它。

其餘的 shell hook codes 目前只使用於 Windows 95 環境下，Windows NT 3.51 並不支援這些 hook codes。然而未來的 Windows NT 版本將會支援它們。

當視窗被最小化、被最大化或恢復原狀時，系統會將 HSHELL_GETMINRECT hook code 傳給最新掛上的 WH_SHELL filter function，表示系統需要最小化視窗的矩形座標（例如工作列上的一個按鈕的矩形）。wParam 內含視窗代碼，lParam 內含一個 RECT 結構的位址，該 RECT 結構用來接收座標。當一個視窗要被最小化、最大化或還原時，系統會將最小化矩形座標傳給 DrawAnimatedRects 函式，用以產生視窗放大縮小的效果（zoom effect）。

HSHELL_WINDOWACTIVED hook code 表示說「作用中的視窗」已被改變至另一個最上

層（top-level）、無人擁有（unowned）的視窗，其 wParam 參數內含視窗代碼。Windows 95 的工作列（Taskbar）會注意這個 hook code，於是才能夠知道要壓下工作列中的某個按鈕，用以表示目前工作中（active）的視窗。

當一個視窗的圖示（icon）或標題欄（caption）改變，Windows 95 的工作列（Taskbar）必須重畫對應的按鈕。HSHELL_REDRAW hook code 用來通知 shell 程式這樣的情況，其 wParam 參數內含「即將重畫標題欄或改變圖示」的視窗代碼。

HSHELL_TASKMAN hook code 用來表示使用者按下 Ctrl+Esc 以啓動「開始」選單或執行 Task Manager（TaskMan.EXE）。一個提供 task list 的 Shell 程式可以傳回 TRUE，用以阻止 Windows 95 顯示其 task list。

HSHELL_LANGUAGE hook code 被用來表示一個新被載入的鍵盤佈局（layout）[7]。Windows 95 利用這個 hook 來更新鍵盤識別碼（identifier），此碼顯示在工作列（Taskbar）中的時鐘旁邊。如果你想要載入一個新的鍵盤佈局，但又不想呼叫 HSHELL_LANGUAGE hook，你可以設定 LoadKeyboardLayout 函式的 fuFlags 參數爲 KLF_NOTELLSHELL。如果你的程式一次想載入多個鍵盤佈局，這個技術將非常有用，因爲你可以延遲 shell 對於 HSHELL_LANGUAGE 的處理，直到最後一個鍵盤佈局被載入爲止。關於 LoadKeyboardLayout 函式的更多細節，請參閱 Win32 SDK 文件。

[7] 你可以使用鍵盤的屬性對話盒（Keyboard Properties dialog）中的【Language】附頁來增加或改變你鍵盤佈局。

12. WH_CBT

WH_CBT 通常被一個應用程式用來提供 "computer-base training"（CBT）。當這個 hook 被掛上，Windows 會在產生、摧毀、極大化、極小化、移動、縮放一個視窗之前，或是在令一個視窗成爲作用視窗（activated window）之前，呼叫對應的 filter function。Windows 同時也會在完成一個系統命令（system command）之前，或是從執行緒的硬體輸入佇列（hardware input queue）中移除滑鼠或鍵盤 events 之前，或是在設定輸入焦點之前，或是在取得 WM_QUEUESYNC 訊息之前，呼叫對應的 filter function。這個 hook 可以讓程式監視器一個使用者的演變發展，可以因爲自動執行某些工作而輔助使用者。

舉個例子，一個應用程式可以在使用者剛開始接觸它時，給予適度的指導，使他能正確完成某些操作。程式可以告訴使用者如何進行，CBT hook 可以用來監視使用者是否成功地執行了那些步驟。此外，許多提供 CBT hook 的程式往往會專注在某些對使用者而言十分新奇的觀念，而強迫程式很快執行一些不重要的勞務。例如使用者可能想要學習如何在一個文書處理軟體中將一份文件格式化，一個好的文書處理軟體一定不希望使用者先得輸入一段文字然後才能顯示文件的格式化功能。這種情況下，WH_CBT hook 可以掛上 WH_JOURNALPLAYBACK hook，強迫產生按鍵動作，輸入文字，給使用者做爲操作對象。等到所有的按鍵都播放完畢，再卸除 WH_JOURNALPLAYBACK hook。

下表列出 WH_CBT hook 的 filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HCBT_ACTIVATE	被 activated 之視窗的代碼。	指向 CBTACTIVATESTRUCT 結構。	如果視窗應該被 activated，傳回 0。傳回 1 表示阻止。
HCBT_CLICKSKIPPED	從佇列中移除的滑鼠訊息。	指向 MOUSEHOOKSTRUCT 結構。	未使用，傳回 0。
HCBT_CREATEWND	被產生之視窗的代碼。	指向 CBT_CREATEWND 結構。	如果視窗應該被產生，則傳回 0。傳回 1 表示阻止。

nCode	wParam	lParam	lResult
HCBT_DESTROYWND	被摧毀之視窗的代碼。	NULL	如果視窗應該被摧毀，則傳回 0。傳回 1 表示阻止。
HCBT_KEYSKIPPED	從佇列中移除的虛擬鍵盤碼。	與視窗函式收到 WM_KEYDOWN 訊息時的 lParam 參數相同。	未使用，傳回 0。
HCBT_MINMAX	被最大化或最小化的視窗的代碼。	LOWORD 中指定一個 ShowWindow 函式所使用的識別字（例如 SW_SHOWMAXIMIZED ）	如果視窗應被最小化或最大化，則傳回 0；如欲避免其發生，則傳回 1。
HCBT_MOVESIZE	被移動或改變大小的視窗的代碼。	指向一個 RECT 結構。	如果視窗應被移動或調整大小，則傳回 0；如欲避免其發生，則傳回 1。
HCBT_QS	NULL	NULL	未使用，傳回 0。
HCBT_SETFOCUS	獲得輸入焦點的視窗的代碼。	失去輸入焦點的視窗的代碼。	如果輸入焦點應被改變，則傳回 0；如欲避免其發生，則傳回 1。
HCBT_SYSCOMMAND	指定使用者所選擇的系統命令（例如 SC_CLOSE）。	與 WM_SYSCOMMAND 訊息的 lParam 參數相同。	如果系統命令應被執行，則傳回 0；如欲避免其發生，則傳回 1。

不論何時只要有一個視窗將被 activated，Windows 就會呼叫最新被掛上的 WH_CBT filter function，並傳給它 HCBT_ACTIVATE hook code。wParam 參數存放著正要被 activated 的視窗的代碼，lParam 參數是個指向 CBTACTIVATESTRUCT 結構的指標：

```
typedef struct tagCBTACTIVATESTRUCT {
   BOOL    fMouse;
   HWND    hWndActive;
} CBTACTIVATESTRUCT, *LPCBTACTIVATESTRUCT;
```

如果視窗之所以被 activated 是因爲滑鼠點選（click）的結果，fMouse 欄位會是 TRUE。hWndActive 欄位表示目前作用中（active）的視窗的代碼。

當一個 CBT 程式想知道使用者何時在某一個視窗上按下滑鼠按鈕，它可以藉著 HCBT_CLICKSKIPPED hook code 得知。爲達此目的，程式可以掛上 WH_JOURNALPLAYBACK hook，強迫硬體的 input events 送入系統輸入佇列（system's input queue）中。當滑鼠點選（click）訊息從佇列中被移除並且此時安裝有一個 WH_MOUSE hook，Windows 會呼叫最新掛上的 WH_CBT filter function 並傳入 HCBT_CLICKSKIPPED hook code。這時候，接收 HCBT_CLICKSKIPPED hook code 的 WH_CBT filter function 便可以掛上 WH_JOURNALPLAYBACK hook 來播放 events 。

舉個例子，一個試算表軟體可能希望知道使用者是否在某一個儲存格（cell）上按下滑鼠按鈕做點選（click）動作。它可以首先在 WH_MOUSE filter function 中做檢查，看看是否滑鼠游標落在目標儲存格之上。如果是的話，filter function 傳回 1，通知 Windows 說此訊息不應該被程式繼續處理。這時候 Windows 會呼叫 WH_CBT filter function，並傳入 HCBT_CLICKSKIPPED hook code，於是程式可以掛上 WH_JOURNALPLAYBACK hook 來重複播放一連串的 events，像是行列加總運算等等。

HCBT_CLICKSKIPPED 的 wParam 參數用來識別發生的滑鼠訊息，lParam 參數則是指向一個 MOUSEHOOKSTRUCTURE 結構，此結構與被送至 WH_MOUSE filter function 的結構一樣，請參考本章稍早之前有關於 WH_MOUSE 的討論。

HCBT_KEYSKIPPED hook code 的工作方式和 HCBT_CLICKSKIPPED 是一樣的，只不過 HCBT_KEYSKIPPED 和鍵盤輸入訊息有關，而不是和滑鼠輸入訊息有關。

每當一個新視窗即將誕生，Windows 會呼叫最新掛上的 WH_CBT filter function，並傳入 HCBT_CREATEWND hook code，其 wParam 參數爲即將誕生之視窗的視窗代碼。此時 Windows 已經在記憶體中建立起視窗的資料結構，但是還未送出任何訊息給視窗（諸如 WM_NCCREATE 或 WM_CREATE 等等）。如果 filter function 傳回值是 1，視窗將會被摧毀，而 CreateWindowEx 函式會傳回 NULL 給企圖產生視窗的那個執行緒。如果 filter

function 傳回的是 0，視窗會順利誕生，而且其視窗函式會順利收到 WM_NCCREATE 訊息和 WM_CREATE 訊息。

當 filter function 收到 HCBT_CREATEWND hook code 時，雖然視窗函式尚未收到 WM_NCCREATE 和 WM_CREATE 訊息，但是 filter function 可以送出其他一些訊息給視窗。你得當心，這時候的視窗尚未被完全初始化，你送過去的訊息可能會出問題。所以儘可能不要這麼做。

當 filter function 收到 HCBT_CREATEWND hook code，其 lParam 參數指向一個 CBT_CREATEWND 結構：

```
typedef struct tagCBT_CREATEWNDA {
    struct tagCREATESTRUCTA *lpcs;
    HWND          hwndInsertAfter;
} CBT_CREATEWNDA, *LPCBT_CREATEWNDA;

typedef struct tagCBT_CREATEWNDW {
    struct tagCREATESTRUCTW *lpcs;
    HWND          hwndInsertAfter;
} CBT_CREATEWNDW, *LPCBT_CREATEWNDW;

#ifdef UNICODE
typedef CBT_CREATEWNDW CBT_CREATEWND;
typedef LPCBT_CREATEWNDW LPCBT_CREATEWND;
#else
typedef CBT_CREATEWNDA CBT_CREATEWND;
typedef LPCBT_CREATEWNDA LPCBT_CREATEWND;
#endif // UNICODE
```

其中 lpcs 欄位是一個指向 CREATESTRUCT 結構的指標，此結構所存放的是當執行緒企圖產生視窗時，傳給 CreateWindowEx 函式的所有參數。結構中的 hwndInsertAfter 欄位記錄的是有關於新視窗將安插的位置（以 z-order 而言。也就是說新視窗將安全插在 hwndInsertAfter 所代表的視窗之後）。當 filter function 收到 HCBT_CREATEWND 通知，它可以檢驗這些內容並加以修改。

當一個 CBT 程式掛上一個 journal playback hook，它需要知道 events 的播放何時停止。

WM_QUEUESYNC 訊息可以提供這樣的資訊。在你的 journal playback hook 中，當 EVENTMSGs 結構播放完畢，你可以將 WM_QUEUESYNC 訊息填入 EVENTMSG 結構的 message 欄位中，並設定 paramL 欄位爲「應接受 WM_QUEUESYNC 訊息的視窗的代碼」。由於 WM_QUEUESYNC 很特殊，Windows 允許這個訊息進入系統輸入佇列（system input queue）中。產生「指定於 paramL 欄位中之視窗」的那個執行緒最終會從其訊息佇列中取得 WM_QUEUESYNC 訊息。這時候 GetMessage 或 PeekMessage 函式會呼叫最新被掛上的 WH_CBT filter function，傳入 HCBT_QS hook code。這表示 WH_JOURNALPLAYBACK 的 events 播放動作已完成，CBT hook 可以卸除 playback hook 並繼續它自己的事情了。第８章的 SKDemo 範例程式將示範如何使用 WM_QUEUESYNC 訊息。

13. WH_FOREGROUNDIDLE

當前景（foreground）視窗的產生者（執行緒）即將處於睡眠狀態時，WH_FOREGROUNDIDLE filter function 會被呼叫。假設 WordPad 是目前工作中（active）的程式（也就是擁有前景視窗）。如果 WordPad 執行緒呼叫 GetMessage 函式，而其訊息佇列中沒有任何訊息等待被處理，那麼 WordPad 執行緒將不會再做任何有效益的工作；這時候系統會令這個執行緒進入睡眠狀態，直到其訊息佇列中又有訊息進來爲止。但是在讓執行緒進入睡眠狀態之前，系統會呼叫最新掛上的 WH_FOREGROUNDIDLE filter function。

WH_FOREGROUNDIDLE hook 基本上是爲了系統內部使用，對一般應用程式來說沒有太大用途[8]。列出這個 hook 的目的僅是爲了完整性。下表列出 WH_FOREGROUNDIDLE filter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	NULL	NULL	未使用，傳回 0。

[8] 如果你有足夠時間好好想想這個 hook 的用途，你一定會有這樣一個想法：也許你擁有一個低優先權執行緒，它正等待著這個 hook 的通知；一旦執行緒收到通知便提升它自己的優先權，使它的工作得以獲得較快的處理。這並不是 WH_FOREGROUNDIDLE hook 的好用途，一則因爲沒有任何通告訊息能告訴你前景執行緒何時已不再處於閒置（idle）狀態，以便讓你再次將執行緒的優先權降低；另一則是當前景執行緒呈現閒置（idle）狀態，系統其實已經會自動爲低優先權執行緒安排 CPU 時間。再者，提昇一個執行緒的優先權並不能使其執行速度加快，它只是能夠阻止較低優先權的執行緒取得 CPU 而已。此外，提昇你的這個執行緒的優先權，可能會阻礙前景執行緒的甦醒。

14. WH_DEBUG

當某個程式想要監視已安裝之各種 hooks 的運作，它可以安裝一個 WH_DEBUG hook。WH_DEBUG hook 允許程式掛上一個 filter function，只要任何其他的 hook filter functions 被呼叫，它即被通知。這個 hook 通常用來幫助對一個「使用 hook 的程式」進行除錯。

還記得嗎，當一個新的 hook 被掛上，在呼叫任何已掛上的 hook filter functions 之前，Windows 會先去呼叫這個新的 hook 的 filter function。這個新的 filter function 可以呼叫 hook chain 之中的下一個 filter function，也可以不呼叫 CallNextHookEx 函式，於是便中斷了 hook chain。如果你安裝了一個 hook，卻發現你的 filter function 沒有被呼叫到，可能是因爲在你之後新掛上的 hook，其 filter function 沒有呼叫 CallNextHookEx 函式所致。這種情況下你就可以利用 WH_DEBUG hook 來幫助你，看看到底是哪些 filter functions 在作怪（喔，我猜想微軟並不認爲有這種可能性）。

每當系統要呼叫任何一個 hook filter functions 之前，它會先呼叫在 hook chain 之中最新掛上的 WH_DEBUG filter function。下表列出 WH_DEBUG fiilter function 的參數和傳回值：

nCode	wParam	lParam	lResult
HC_ACTION	指明將被呼叫的 hook 型態（WH_*）	指向一個 DEBUGHOOKINFO 結構。	如果即將被呼叫的 hook 應該要被執行，傳回 0；如果不讓它執行，就傳回 1。

每當一個 hook filter function（WH_DEBUG 除外）即將被呼叫，Windows 就會呼叫最新被掛上的 WH_DEBUG filter function，並傳入 HC_ACTION hook code。其 wParam 參數代表即將被呼叫的 hook 型態，lParam 參數爲一個指向 DEBUGHOOKINFO 結構的指標：

```
typedef struct tagDEBUGHOOKINFO {
   DWORD   idThread;
   DWORD   idThreadInstaller;
   LPARAM  lParam;
```

```
    WPARAM  wParam;
    int     code;
} DEBUGHOOKINFO, *PDEBUGHOOKINFO, NEAR *NPDEBUGHOOKINFO, FAR*
LPDEBUGHOOKINFO;
```

其中 idThread 欄位表示即將執行 filter function 的執行緒，idThreadInstaller 欄位表示安裝「wParam 參數所指出之 hook 型別」的執行緒。

結構中的其餘欄位，像 lParam、wParam 和 code，是即將被呼叫的 filter function 的參數。你可以檢閱這些值，但你對它們所做的任何修改都不會有效果。

和其它所有的 hooks 一樣，如果你想繼續執行 hook chain 中的下一個 filter function，你必須在你的 filter function 中呼叫 CallNextHookEx 函式，並傳入與你的 filter function 所獲得的參數完全相同的參數。千萬不要在呼叫 CallNextHookEx 函式時，把 DEBUGHOOKINFO 結構中的欄位傳入。

KeyCount 程式

KeyCount 程式（KeyCount.EXE）的原始碼顯示於**程式列表 6.1 ~ 6.7** 中，示範如何使用 system-wide keyboard hook 來計數按鍵次數。此程式的執行畫面（主視窗）如**圖 6.2** 所示。

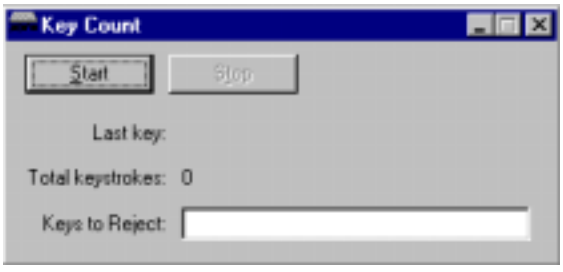


圖 6.2　KeyCount 程式主視窗

KeyCount 程式的主視窗有兩個按鈕：【Start】和【Stop】。壓下【Start】鈕便會掛上一個 system-wide keyboard hook，於是當你按下或放開按鍵時，KeyCount 程式會將每一個按鍵的資訊顯示給你，並顯示從一開始到現在的按鍵 events 總個數。此外，KeyCount 還有一個 edit 視窗，你在上面所鍵入的每一個字元，會使 keyboard hook filter function 傳回 1（表示拒絕此一 event），於是整個系統對於該按鍵皆沒有反應（失效了）。按下【Stop】鈕可以取消 keyboard hook。

表 6.2 和表 6.3 列出建立此程式及其 DLL 所需用到的檔案。由於 KeyCount 程式所安裝的是一個 system-wide hook，所以其 filter function 必須放在 DLL 檔中。

表 6.2 建立 KeyCount 程式所需的檔案

檔案	說明
KeyCount.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
KeyCount.MSG	MsgCrack 的輸入檔，用來產生 KeyCount.H（請參閱附錄 B）。
KeyCount.H	爲 UM_KEYEVENT 通告訊息（notification）所產生的訊息剖析器。
KeyCount.RC	內含主視窗的對話盒面板（template）和其圖示（icon）。
Resource.H	內含 KeyCount.RC 檔中所有資源的 ID。
KeyCount.ICO	主視窗圖示（icon）。
KeyCount.MAK	Visual C++ 的 MAK 檔。

表 6.3 建立 KybdHk.DLL 所需的檔案

檔案	說明
KybdHk.C	內含某個 system-wide keyboard hook 的實作內容（此 hook 用來攔截某些按鍵訊息），以及一個 KeyboardHook API 函式（用來安裝和移除 keyboard hook）。
KybdHk.H	內含 KeyboardHook API 函式的原型宣告。
KybdHk.MAK	Visual C++ 的 MAK 檔，用來產生 Kybdhk.DLL。

這個 KybdHk.DLL 輸出（exports）兩個函式：

```
KYBDHKLIBAPI BOOL WINAPI KeyboardHook_Start (HWND hwndPost,
  UNIT uMsgNotify, HWND hwndEdit);
KYBDHKLIBAPI BOOL WINAPI KeyboardHook_Stop ();
```

KeyboardHook_Start 函式有三個參數：第一個參數 hwndPost，代表「每當有鍵盤 events 發生時皆會被通知」的那個視窗。第二個參數 uMsgNotify，是應該被 “post” 到 hwndPost 視窗上的一個通知訊息。最後一個參數 hwndEdit，代表 edit 控制元件的代碼，在此 edit 控制元件中所出現的字元都將被系統拒絕。

當你按下【Start】鈕，KeyCount 程式的 KeyCount_OnCommand 函式會呼叫 KeyboardHook_Start 函式處理之：

```
void KeyCount_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{

  HWND hwndEdit;

  switch (id) {
    .
    .
    .
    case IDC_START:
      hwndEdit = GetDlgItem(hwnd, IDC_EDIT);
      if (KeyboardHook_Start(hwnd, UM_KEYEVENT, hwndEdit)) {
        SetDlgItemText(hwnd, IDC_LASTKEY, __TEXT(""));
        SetDlgItemInt(hwnd, IDC_KEYCOUNT, 0, FALSE);
        SETKEYCOUNT(hwnd, 0);
        EnableWindow(GetDlgItem(hwnd, IDC_START), FALSE);
        EnableWindow(GetDlgItem(hwnd, IDC_STOP),  TRUE);
        SetFocus(GetDlgItem(hwnd, IDC_STOP));
      } else {
        adgMB(__TEXT("Unable to start keyboard hook"));
      }
      break;

    case IDC_STOP:
      if (!KeyboardHook_Stop())
        adgMB(__TEXT("Unable to stop keyboard hook"));
```

```
        EnableWindow(GetDlgItem(hwnd, IDC_START), TRUE);
        EnableWindow(GetDlgItem(hwnd, IDC_STOP),  FALSE);
        SetFocus(GetDlgItem(hwnd, IDC_START));
        break;
   }
}
```

通告訊息（notification）UM_KEYEVENT 被定義爲 WINAPP+0。我不使用 WM_USER 做爲自定訊息的基底，原因是 KeyCount 程式使用了系統的對話盒類別，而不是我自己註冊的類別。微軟公司保證系統類別不會使用 WM_APP(0x00008000) ~ 0x0000BFFF 之間的值。因此，以 WM_APP 爲基底是我們在定義自己的 UM_KEYEVENT 訊息時最好的選擇。

當你壓下【Stop】鈕，KeyCount 程式的 KeyCount_OnCommand 程式會呼叫 KeyboardHook_Stop 函式。

當我們的 keyboard filter function 收到一個鍵盤通知，它會 "post" 一個 UM_KEYEVENT 通告訊息給 KeyCount 主視窗。KeyCount 的對話盒函式會在 KeyCount_OnKeyEvent 函式中處理這個訊息：

```
#define GETKEYCOUNT(hwnd)    ((UINT) GetWindowLong(hwnd, GWL_USERDATA))
#define SETKEYCOUNT(hwnd, c) SetWindowLong(hwnd, GWL_USERDATA, (UINT) c)
.
.
.
void KeyCount_OnKeyEvent (HWND hwnd, UINT vk, LPARAM lParam) {

   BOOL fDown = ((HIWORD(lParam) & KF_UP) == 0);
   TCHAR sz[128];
   UINT uKeyCount = GETKEYCOUNT(hwnd);
   TCHAR c = (TCHAR) LOWORD(MapVirtualKey(vk, 2));

   if (fDown)
      uKeyCount++;
   SetDlgItemInt(hwnd, IDC_KEYCOUNT, uKeyCount, FALSE);
   wsprintf(sz, __TEXT("char=%c, virtual key=%d <%s>"),
      IsCharAlphaNumeric(c) ? c : __TEXT('?'), vk,
      fDown ? __TEXT("pressed") : __TEXT("released"));
   SetDlgItemText(hwnd, IDC_LASTKEY, sz);
```

```
   SETKEYCOUNT(hwnd, uKeyCount);
}
```

KeyCount_OnKeyEvent 函式利用 GETKEYCOUNT 巨集，從 GWL_USERDATA 中取得目前的按鍵計數。如果 lParam 參數的 KF_UP 位元不爲 on，則表示按鍵被壓下，於是我們把按鍵計數加 1。然後，畫面上的【Total Keystrokes】和【Last Key】欄位會被更新，而且 KeyCount 使用 SETKEYCOUNT 巨集來更新在額外位元組 GWL_USERDATA 中的按鍵計數數值。

現在，將我們的注意力轉移到如何掛上這個 hook。KeyboardHook_Start 函式首先將其三個參數存入全域變數之中，然後呼叫 SetWindowsHookEx 函式：

```
BOOL WINAPI KeyboardHook_Start (HWND hwndPost,
   UINT uMsgNotify, HWND hwndEdit) {

   HHOOK hhook;

   // Return FALSE if hook has already been installed.
   if (g_hhook != NULL)
      return(FALSE);

   adgASSERT(IsWindow(hwndPost));

   g_hwndPost   = hwndPost;
   g_uMsgNotify = uMsgNotify;
   g_hwndEdit   = hwndEdit;

   // Give up the remainder of our thread's timeslice.
   // This gives us a better chance of getting all the way through the call
   // to SetWindowsHookEx and the variable assignment to g_hhook in one shot.
   // If we are preempted after the hook is set, but before the variable is
   // updated, it is possible for another thread to enter our hook filter
   // function before the hook handle is valid.  Under Windows NT this is not
   // a problem.  Under Windows 95, not having a valid hook handle will cause
   // CallNextHookEx to fail.  If there is some reason that it is critical
   // that your application succeed in calling the next filter function in
   // the chain, the only robust way to write this code is to use something
   // like the SWMRG (single-writer, multiple-reader guard) object developed
   // in Advanced Windows (Microsoft Press).
   Sleep(0);
```

```
   // Set our keyboard hook.
   hhook = SetWindowsHookEx(WH_KEYBOARD,
      KeyboardHook_HookProc, g_hinstDll, 0);

   // Ensure that g_hhook is always valid (even if we are preempted whilst
   // in the middle of writing to it) by updating the variable atomically.
   InterlockedExchange((PLONG) &g_hhook, (LONG) hhook);

   return(g_hhook != NULL);
}
```

因爲我們所要安裝的是一個 system-wide hook，所以 dwThreadID 必須爲 0，hmod 參數則應該是傳給 DLLMain 函式的「KybdHk.DLL 執行個體的代碼（instance handle）」。現在，你或許注意到了，在呼叫 SetWindowsHookEx 函式之前，我呼叫了一個 Sleep 函式，並且在之後又呼叫了 InterlockedExchange 函式。這幾行程式碼是爲了解決 Windows 95 中的一個潛在性問題：執行緒同步（synchronization）問題。

在 Windows NT 之中，CallNextHookEx 函式會完全忽略所傳入的 HHOOK。無論你傳入什麼，Windows NT 都會呼叫 hook chain 中的下一個 hook filter function，這是因爲它隨時都在維護這個 hook chain，而且它也知道哪一個 filter function 剛被呼叫。不幸的是，Windows 95 並沒有記錄哪一個 filter functions 剛被呼叫，它也沒有要求 hook handle 必須是合法的。這或許不是什麼大問題，但是請你考量這樣的可能性：執行緒 A 利用 SetWindowsHookEx 函式安裝了一個 system-wide WH_KEYBOARD hook，然後執行緒 A 在 hook 被設定之後，但在 hook handle 被存入全域變數 g_hhook 之前，被作業系統強制岔斷。此時 hook handle 尚未初始化完成。現在，執行緒 B 取得了一個按鍵訊息，於是執行緒 B 會呼叫 WH_KEYBOARD filter function，但此時 HHOOK 變數仍是 NULL，所以對 CallNextHookEx 函式的呼叫動作會失敗，以至於下一個 filter function 不會被喚起。這樣一來 hook chain 就被破壞了，可能會影響到其它已安裝的 hooks。

這個問題只有透過執行緒同步物件的使用（就像 SWMRG，Single-Writer and Multiple-

Reader Guard），才能獲得完全的解決。SWMRG 在 ***Advanced Windows*** 一書有介紹[9]（在這裡，“readers”指的是那些已掛上 hook 的執行緒，而“writer”指的是呼叫 SetWindowsHookEx 函式的執行緒）。然而，我們還是有辦法大大增加執行 SetWindowsHookEx 函式以及指定變數的機會 -- 如果你呼叫 Sleep 函式並傳入參數 0（毫秒），以此方式放棄執行緒的剩餘時間的話。之後，當這個執行緒再次被喚醒，它將會有比較好的機會，得以通過那些引起執行緒同步化問題的程式碼。無疑地，這類「技術」實在不可靠，而且已經超越了你所能控制的範圍（像是執行緒優先權的控制）。但我還是決定呼叫 Sleep 函式，因爲它不會造成任何傷害，而且 SWMRG 物件的使用實在是超出了本書範圍。

在 KeyboardHook_Start 函式中呼叫 InterlockedExchange 函式，是爲了確保在共享記憶體區間（shared section）中的 hook handle（即 g_hhook）被更新期間，不會被其它執行緒強制岔斷。如果在那個節骨眼兒不幸被強制岔斷的話，共享變數（shared variable）中的值就有可能遭到破壞。

KeyboardHook_Start 函式所安裝的 filter function：KeyboardHook_HookProc，看起來大致如下：

```
static LRESULT WINAPI KeyboardHook_HookProc (int nCode,
  WPARAM wParam, LPARAM lParam) {

  LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);

  if (nCode == HC_ACTION) {

     // Notify application's window that this thread received a keystroke
     PostMessage(g_hwndPost, g_uMsgNotify, wParam, lParam);
     .
     .
     .
  }
  return(CallNextHookEx(g_hhook, nCode, wParam, lParam));
}
```

[9] 請參閱 ***Advanced Windows*** 第九章“Thread Synchronization”（Microsoft Press，1995）

如果 hook function 收到的 hook code 爲 HC_ACTION，表示它收到了一個 keyboard event 的通知，於是我們通知 KeyCount 程式（因爲它呼叫 KeyboardHook_Start 函式），方法是"posting" 通告訊息 g_uMsgNotify，將它送往通告視窗 g_hwndPost 之中。

KeyboardHook_Start 函式設定的是一個 system-wide keyboard hook。這表示當另一個行程的執行緒取得了一個 keyboard event，hook filter function 必須被呼叫。但此時內含 filter function 的 DLL 尚未被載入到行程位址空間中，所以系統內部呼叫 LoadLibrary，將 DLL 注射（inject）到行程位址空間中。一旦 DLL 被注入進來，執行緒便可執行其中的 filter function 了。

現在出現了一個有趣的問題：如果其它程式如 Notepad 作用起來，KeyboardHook_HookProc 函式應該在 Notepad 程式的行程內（process context）執行。這很好，但是我們的 DLL 被注射到 Notepad 行程位址空間中，將有其自己的 g_hwndPost 和 g_uMsgNotify 全域變數。而不幸的是，這些變數與我們在 KeyboardHook_Start 函式中所設定的值不同。事實上它們會被設定爲「DLL 被載入到 Notepad 位址空間時」的初值（分別爲 NULL 和 WM_NULL）。只要把變數放到一個共享資料區間（shared section） 中，這個問題很容易就獲得了解決：

```
// Calling SetWindowsHookEx with a thread ID of zero will cause this DLL to be
// injected into other processes. Therefore, we must declare a shared data
// section so that all mappings of our DLL (in all processes) share the same
// set of global variables.

#pragma comment(lib, "kernel32 " "-section:Shared.rws")
#pragma data_seg("Shared")

HHOOK g_hhook     = NULL;      // Hook handle for system-wide keyboard hook
HWND  g_hwndPost  = NULL;      // Window to notify of keyboard events
UINT  g_uMsgNotify = WM_NULL;  // Notification message to post to g_hwndPost
HWND  g_hwndEdit  = NULL;      // Characters to reject with hook

#pragma data_seg()
```

現在，當 Windows 將 DLL 注入到其它的行程位址空間中，這些變數會被映對到相同的實際儲存空間（如果 DLL 被強制載入到不同的基底位址上，它們可能就會有不同的虛擬位址，virtual address）。由於我們的變數被映射到相同的實際儲存空間，所以在所有行程位址空間中，它們的值都是一樣的[10]。

KeyboardHook_HookProc 函式的其餘動作用來處理如何拒絕某些字元的輸入。這些字元是在主視窗的 edit 控制元件中列出的字元（並被儲存到 g_hwndEdit 全域共享變數中）：

```
static LRESULT WINAPI KeyboardHook_HookProc (int nCode,
   WPARAM wParam, LPARAM lParam) {

   LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);

   if (nCode == HC_ACTION) {

      TCHAR ac[2];
      BYTE bKeyState[256];
      UINT uScanCodeAndShiftState = HIWORD(lParam) & (0x00FF | KF_UP);

      // Notify application's window that this thread received a keystroke
      PostMessage(g_hwndPost, g_uMsgNotify, wParam, lParam);

      // Translate keystroke to ANSI or Unicode and reject it if it's one
      // of the characters in g_hwndEdit.
      GetKeyboardState(bKeyState);
#ifdef UNICODE
      if (ToUnicode(wParam, uScanCodeAndShiftState,
         bKeyState, ac, 1, 0) == 1) {
#else
      if (ToAscii(wParam, uScanCodeAndShiftState,
         bKeyState, (PWORD) ac, 0) == 1) {
#endif
      TCHAR sz[256];
      FORWARD_WM_GETTEXT(g_hwndEdit, adgARRAY_SIZE(sz), sz, SendMessage);
      if (_tcschr(sz, ac[0])) {

           // Reject the key event.
```

[10] 如你想多瞭解共享區間（shared section）的使用，請參閱 ***Advanced Windows*** 第 11 章 "Dynamic Link Library"（Microsoft Press，1995）。

```
            lResult = 1;
         }
      }
   }
   return(lResult);
}
```

在我們將我們的 keyboard event 和 edit 控制元件中的內容拿來比較之前，我們必須先呼叫 ToUnicode 和 ToAscii 函式[11]，把 wParam 參數中的虛擬碼（virtual-key code）和 lParam 參數中的掃瞄碼（scan code）及 shift-state 轉換爲一個字元值。然後我們呼叫 SendMessage，送出 WM_GETTEXT 訊息給 edit 控制元件，取得 edit 視窗中的內容。如果 keyboard event 與 edit 控制元件中的任何一個字元相同，我們就傳回 1，要求停止處理這個 event。

似乎我已經把 KeyCount 程式解說完了。然而，在呼叫 SendMessage 函式的背後，還潛藏著許多我們未察覺的事！

爲什麼我們不能夠直接呼叫 GetWindowText 呢？因爲「呼叫 GetWindowText」和「送出 WM_GETTEXT 訊息」兩者並不相同。GetWindowText 函式不會送出 WM_GETTEXT 訊息，它直接呼叫 DefWindowProc 函式來取得視窗上的文字。但 edit 控制元件卻是個特例，它會處理 WM_GETTEXT 訊息並從一個內部緩衝區中取得視窗內容。由於 DefWindowProc 函式並不知道這件事情，所以它還是會去取視窗文字，並且總是一場空。所以我們必須強制送出 WM_GETTEXT 訊息才能獲得 edit 視窗內的文字。

雖然這些行爲上的差異看起來十分奇特，但其背後的原因卻是相當地有趣。當你使用 Alt+Tab 鍵在視窗之間來回切換時，系統必須取得每一個視窗的標題文字，這樣它才能夠在你遵循切換時顯示之。不幸的是，如果產生出那些視窗的某一個執行緒凍住了（hang），系統就沒辦法藉由送出（sending）WM_GETTEXT 訊息來獲得視窗的標題文字，因爲 SendMessage 函式會虛懸，等待該執行緒醒來處理這個訊息；而由於該執行緒永遠不會醒來，所以系統也就得不到其標題文字。使用 GetWindowText 函式可以解決這個問題，因

[11] 如果你想多瞭解 ToAscii 和 ToUnicode 兩個函式，請參閱第８章。

爲 DefWndowProc 函式總是可以得到標題文字 -- 不管產生視窗的那個執行緒的目前狀態爲何。

使用 SendMessage 函式，還有另外一個比較複雜的原因。這必須和執行緒同步化一起配合討論。由於執行緒之間的 SendMessage 動作會將訊息送到執行緒的訊息佇列中，其中已經隱含了執行緒的循序（serialization）性質。不論有多少個執行緒意圖存取我們這個 edit 控制元件的內容（視窗文字），一次只有一個執行緒可以遂其所願。爲什麼這很重要呢？想想看我們應該如何架構 KeyCount 程式，使它能夠傳入一個緩衝區到 KybdHk.DLL 之中的一個名爲 KeyHook_SetRejectCharacters 的函式。每當 edit 控制元件的內容改變，新的內容就會被傳給此函式，而它只是簡單地將這新的資料拷貝到 KybdHk DLL 的共享區間中的一個全域緩衝區內，使所有的行程皆能立即看到這份新的資料。但這其中卻有一個大問題！由於我們無法控制誰能來存取這個緩衝區，所以在資料改變的期間，KeyHook_SetRejectCharacters 函式可能會被其它執行緒強制岔斷，於是共享區間中的全域緩衝區所含資料遭到毀壞，而其它執行緒並不知道！這個結果或許不會對一個像 KeyCount 這樣並沒有什麼大用途的程式帶來大災難，但卻可能對真實世界中的應用軟體造成嚴重的影響。事實上一個有實際作用的軟體應該使用同步化物件（像是 SWMRG，發展於 ***Advanced Windows*** 一書中）來避免這個問題，因爲使用 SendMessage 函式連續存取資料可能會妨礙其它執行緒的正常工作。

KeyCount.ICO

程式列表 6.1　KeyCount.C

```
#0001  /******************************************************************
#0002  Module name: KeyCount.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Keyboard hooks demonstration application
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include <tchar.h>
```

```
#0014  #include "KybdHk.h"
#0015  #include "KeyCount.h"
#0016  #include "Resource.h"
#0017  #pragma warning(disable: 4001)    /* Single line comment */
#0018
#0019
#0020  ////////////////////////////////////////////////////////////////
#0021
#0022
#0023  // Force a link with the import library for KybdHk.dll
#0024  #pragma comment(lib, adgLIBBUILDTYPE adgLIBCPUTYPE "\\" "KybdHk")
#0025
#0026
#0027  ////////////////////////////////////////////////////////////////
#0028
#0029
#0030  #define GETKEYCOUNT(hwnd)    ((UINT) GetWindowLong(hwnd, GWL_USERDATA))
#0031  #define SETKEYCOUNT(hwnd, c) SetWindowLong(hwnd, GWL_USERDATA, (UINT) c)
#0032
#0033
#0034  ////////////////////////////////////////////////////////////////
#0035
#0036
#0037  BOOL KeyCount_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0038
#0039    adgSETDLGICONS(hwnd, IDI_KEYCOUNT, IDI_KEYCOUNT);
#0040
#0041    // Make the keycount dialog topmost so that it is easy to watch keys in
#0042    // applications that take up most or all of the screen.
#0043    SetWindowPos(hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);
#0044
#0045    EnableWindow(GetDlgItem(hwnd, IDC_START), TRUE);
#0046    EnableWindow(GetDlgItem(hwnd, IDC_STOP),  FALSE);
#0047
#0048    // Reset keystroke count.
#0049    SETKEYCOUNT(hwnd, 0);
#0050
#0051    return(TRUE);               // Accept default focus window.
#0052  }
#0053
#0054
#0055  ////////////////////////////////////////////////////////////////
#0056
#0057
#0058  void KeyCount_OnKeyEvent (HWND hwnd, UINT vk, LPARAM lParam) {
#0059
```

```
#0060    BOOL fDown = ((HIWORD(lParam) & KF_UP) == 0);
#0061    TCHAR sz[128];
#0062    UINT uKeyCount = GETKEYCOUNT(hwnd);
#0063    TCHAR c = (TCHAR) LOWORD(MapVirtualKey(vk, 2));
#0064
#0065    if (fDown)
#0066       uKeyCount++;
#0067    SetDlgItemInt(hwnd, IDC_KEYCOUNT, uKeyCount, FALSE);
#0068    wsprintf(sz, __TEXT("char=%c, virtual key=%d <%s>"),
#0069       IsCharAlphaNumeric(c) ? c : __TEXT('?'), vk,
#0070       fDown ? __TEXT("pressed") : __TEXT("released"));
#0071    SetDlgItemText(hwnd, IDC_LASTKEY, sz);
#0072
#0073    SETKEYCOUNT(hwnd, uKeyCount);
#0074 }
#0075
#0076
#0077 ////////////////////////////////////////////////////////////////
#0078
#0079
#0080 void KeyCount_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0081
#0082    HWND hwndEdit;
#0083
#0084    switch (id) {
#0085
#0086       case IDCANCEL:            // Allows dialog box to close.
#0087
#0088          // Unhook keyboard hook before exiting.
#0089          if (!KeyboardHook_Stop())
#0090             adgMB(__TEXT("Unable to stop keyboard hook"));
#0091          EndDialog(hwnd, id);
#0092          break;
#0093
#0094       case IDC_START:
#0095          hwndEdit = GetDlgItem(hwnd, IDC_EDIT);
#0096          if (KeyboardHook_Start(hwnd, UM_KEYEVENT, hwndEdit)) {
#0097             SetDlgItemText(hwnd, IDC_LASTKEY, __TEXT(""));
#0098             SetDlgItemInt(hwnd, IDC_KEYCOUNT, 0, FALSE);
#0099             SETKEYCOUNT(hwnd, 0);
#0100             EnableWindow(GetDlgItem(hwnd, IDC_START), FALSE);
#0101             EnableWindow(GetDlgItem(hwnd, IDC_STOP),  TRUE);
#0102             SetFocus(GetDlgItem(hwnd, IDC_STOP));
#0103          } else {
#0104             adgMB(__TEXT("Unable to start keyboard hook"));
#0105          }
```

```
#0106         break;
#0107
#0108       case IDC_STOP:
#0109         if (!KeyboardHook_Stop())
#0110            adgMB(__TEXT("Unable to stop keyboard hook"));
#0111         EnableWindow(GetDlgItem(hwnd, IDC_START), TRUE);
#0112         EnableWindow(GetDlgItem(hwnd, IDC_STOP),  FALSE);
#0113         SetFocus(GetDlgItem(hwnd, IDC_START));
#0114         break;
#0115    }
#0116  }
#0117
#0118
#0119  ////////////////////////////////////////////////////////////////
#0120
#0121
#0122  BOOL WINAPI KeyCount_DlgProc (HWND hwnd, UINT uMsg,
#0123    WPARAM wParam, LPARAM lParam) {
#0124
#0125     switch (uMsg) {
#0126
#0127       // Standard Windows messages
#0128       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, KeyCount_OnInitDialog);
#0129       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    KeyCount_OnCommand);
#0130
#0131       // Keyboard hook notification message
#0132       adgHANDLE_DLGMSG(hwnd, UM_KEYEVENT,   KeyCount_OnKeyEvent);
#0133    }
#0134
#0135    return(FALSE);              // We didn't process the message.
#0136  }
#0137
#0138
#0139  ////////////////////////////////////////////////////////////////
#0140
#0141
#0142  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0143    LPSTR lpszCmdLine, int nCmdShow) {
#0144
#0145    adgWARNIFUNICODEUNDERWIN95();
#0146    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_KEYCOUNT),
#0147       NULL, KeyCount_DlgProc));
#0148
#0149    return(0);
#0150  }
#0151
```

```
#0152
#0153 ////////////////////////// End of File //////////////////////////
```

程式列表 6.2　KeyCount.MSG

```
#0001 // Module name: KeyCount.msg
#0002 // Written by: Jonathan Locke
#0003 // Notices: Copyright (c) 1995 Jeffrey Richter
#0004 // Purpose: 'MsgCrack' input file for KeyCount application.
#0005
#0006 MessageBase WM_APP
#0007 MessageClass KeyCount
#0008
#0009 Message UM_KEYEVENT KeyEvent \
#0010  - Notification message posted by keyboard hook routine (in KybdHk.dll)\
#0011  when a keyboard event occurs.
#0012 wParam UINT vk - Virtual keycode
#0013 lParam LPARAM lParam - Keystroke flags
#0014 .
```

程式列表 6.3　KeyCount.H

```
#0001 /************************************************************
#0002 Module name: KeyCount.h
#0003 Written by: Jonathan Locke
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Message cracker for UM_KEYEVENT notification message.
#0006 ************************************************************/
#0007
#0008 ////////////////////////////// Messages //////////////////////////
#0009
#0010
#0011 //{{adgMSGCRACK_MESSAGES
#0012
#0013 // Purpose: Notification message posted by keyboard hook routine (in
#0014 //         KybdHk.dll) when a keyboard event occurs.
#0015 // wParam:  UINT vk - Virtual keycode
#0016 // lParam:  LPARAM lParam - Keystroke flags
#0017 // Returns: void
#0018 #define UM_KEYEVENT        (WM_APP + 0)
#0019
#0020 //}}adgMSGCRACK_MESSAGES
#0021
```

```
#0022
#0023 ////////////////////// Message APIs //////////////////////////
#0024
#0025
#0026 //{{adgMSGCRACK_APIS
#0027
#0028 #define KeyCount_KeyEvent(hwnd, vk, lParam) \
#0029    ((void)SendMessage((hwnd), UM_KEYEVENT, (WPARAM)(DWORD)(vk), (LPARAM)(DWORD)(lParam)))
#0030
#0031 //}}adgMSGCRACK_APIS
#0032
#0033
#0034 ////////////////////// Message Crackers //////////////////////////
#0035
#0036
#0037 //{{adgMSGCRACK_CRACKERS
#0038
#0039 // void Cls_OnKeyEvent (HWND hwnd, UINT vk, LPARAM lParam)
#0040 #define HANDLE_UM_KEYEVENT(hwnd, wParam, lParam, fn) \
#0041    ((fn)((hwnd), (UINT)(wParam), (LPARAM)(lParam)), 0)
#0042 #define FORWARD_UM_KEYEVENT(hwnd, vk, lParam, fn) \
#0043    (void)((fn)((hwnd), UM_KEYEVENT, (WPARAM)(DWORD)(vk), (LPARAM)(DWORD)(lParam)))
#0044
#0045 //}}adgMSGCRACK_CRACKERS
#0046
#0047
#0048 ////////////////////////// End of File /////////////////////////////
```

程式列表 6.4　KeyCount.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "resource.h"
#0004
#0005 #define APSTUDIO_READONLY_SYMBOLS
#0006 ////////////////////////////////////////////////////////////////////
#0007 //
#0008 // Generated from the TEXTINCLUDE 2 resource.
#0009 //
#0010 #define APSTUDIO_HIDDEN_SYMBOLS
#0011 #include "windows.h"
#0012 #undef APSTUDIO_HIDDEN_SYMBOLS
#0013
#0014 ////////////////////////////////////////////////////////////////////
#0015 #undef APSTUDIO_READONLY_SYMBOLS
```

```
#0016
#0017
#0018 //////////////////////////////////////////////////////////////
#0019 //
#0020 // Icon
#0021 //
#0022
#0023 IDI_KEYCOUNT           ICON    DISCARDABLE     "KEYCOUNT.ICO"
#0024
#0025 //////////////////////////////////////////////////////////////
#0026 //
#0027 // Dialog
#0028 //
#0029
#0030 IDD_KEYCOUNT DIALOG DISCARDABLE  -32768, 5, 214, 79
#0031 STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0032 CAPTION "Key Count"
#0033 FONT 8, "MS Sans Serif"
#0034 BEGIN
#0035     PUSHBUTTON      "&Start",IDC_START,7,4,50,14
#0036     PUSHBUTTON      "S&top",IDC_STOP,63,4,50,14
#0037     LTEXT          "",IDC_LASTKEY,68,28,138,9
#0038     LTEXT          "0",IDC_KEYCOUNT,68,44,37,9
#0039     RTEXT          "Last key:",IDC_STATIC,4,28,58,9
#0040     RTEXT          "Total keystrokes:",IDC_STATIC,4,44,58,9
#0041     RTEXT          "Keys to Reject:",IDC_STATIC,6,61,56,9
#0042     EDITTEXT       IDC_EDIT,68,59,136,12,ES_AUTOHSCROLL
#0043 END
#0044
#0045
#0046 #ifdef APSTUDIO_INVOKED
#0047 //////////////////////////////////////////////////////////////
#0048 //
#0049 // TEXTINCLUDE
#0050 //
#0051
#0052 1 TEXTINCLUDE DISCARDABLE
#0053 BEGIN
#0054     "resource.h\0"
#0055 END
#0056
#0057 2 TEXTINCLUDE DISCARDABLE
#0058 BEGIN
#0059     "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0060     "#include ""windows.h""\r\n"
#0061     "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
```

```
#0062     "\0"
#0063  END
#0064
#0065  3 TEXTINCLUDE DISCARDABLE
#0066  BEGIN
#0067     "\r\n"
#0068     "\0"
#0069  END
#0070
#0071  /////////////////////////////////////////////////////////////////
#0072  #endif    // APSTUDIO_INVOKED
#0073
#0074
#0075  #ifndef APSTUDIO_INVOKED
#0076  /////////////////////////////////////////////////////////////////
#0077  //
#0078  // Generated from the TEXTINCLUDE 3 resource.
#0079  //
#0080
#0081
#0082  /////////////////////////////////////////////////////////////////
#0083  #endif    // not APSTUDIO_INVOKED
```

程式列表 6.5　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by KeyCount.rc
#0004  //
#0005  #define IDI_KEYCOUNT                101
#0006  #define IDC_START                   1000
#0007  #define IDC_STOP                    1001
#0008  #define IDC_KEYCOUNT                1002
#0009  #define IDC_LASTKEY                 1003
#0010  #define IDD_KEYCOUNT                1004
#0011  #define IDC_EDIT                    1005
#0012  #define IDC_STATIC                  -1
#0013
#0014  // Next default values for new objects
#0015  //
#0016  #ifdef APSTUDIO_INVOKED
#0017  #ifndef APSTUDIO_READONLY_SYMBOLS
#0018  #define _APS_NO_MFC                 1
#0019  #define _APS_NEXT_RESOURCE_VALUE      102
#0020  #define _APS_NEXT_COMMAND_VALUE       40001
```

```
#0021  #define _APS_NEXT_CONTROL_VALUE        1006
#0022  #define _APS_NEXT_SYMED_VALUE          101
#0023  #endif
#0024  #endif
```

程式列表 6.6　KybdHk.C

```
#0001  /************************************************************
#0002  Module name: KybdHk.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Dll which sets a system keyboard hook.
#0006  ************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"         /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #include <tchar.h>
#0013  #pragma warning(disable: 4001)    /* Single line comment */
#0014
#0015
#0016  // We must define KYBDHKLIBAPI as 'dllexport' before including KybdHk.h.
#0017  // KybdHk.h will see that we have already defined KYBDHKLIBAPI and
#0018  // will not (re)define it as 'dllimport'.
#0019  #define KYBDHKLIBAPI  __declspec(dllexport)
#0020  #include "KybdHk.h"
#0021
#0022
#0023  ////////////////////////////////////////////////////////////////
#0024
#0025
#0026  HINSTANCE g_hinstDll = NULL;     // KybdHk.dll's instance handle
#0027
#0028
#0029  ////////////////////////////////////////////////////////////////
#0030
#0031
#0032  // Calling SetWindowsHookEx with a thread Id of zero will cause this DLL to be
#0033  // injected into other processes. Therefore, we must declare a shared data
#0034  // section so that all mappings of our DLL (in all processes) share the same
#0035  // set of global variables.
#0036
#0037  #pragma comment(lib, "kernel32 " "-section:Shared,rws")
#0038  #pragma data_seg("Shared")
```

```
#0039
#0040  HHOOK g_hhook     = NULL;       // Hook handle for system-wide keyboard hook
#0041  HWND  g_hwndPost  = NULL;       // Window to notify of keyboard events
#0042  UINT  g_uMsgNotify = WM_NULL;   // Notification message to post to g_hwndPost
#0043  HWND  g_hwndEdit  = NULL;       // Characters to reject with hook
#0044
#0045  #pragma data_seg()
#0046
#0047
#0048  ////////////////////////////////////////////////////////////////
#0049
#0050
#0051  static LRESULT WINAPI KeyboardHook_HookProc (int nCode,
#0052    WPARAM wParam, LPARAM lParam) {
#0053
#0054    LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);
#0055
#0056    if (nCode == HC_ACTION) {
#0057
#0058       TCHAR ac[2];
#0059       BYTE bKeyState[256];
#0060       UINT uScanCodeAndShiftState = HIWORD(lParam) & (0x00FF | KF_UP);
#0061
#0062       // Notify application's window that this thread received a keystroke
#0063       PostMessage(g_hwndPost, g_uMsgNotify, wParam, lParam);
#0064
#0065       // Translate keystroke to ANSI or Unicode and reject it if it's one
#0066       // of the characters in g_hwndEdit.
#0067       GetKeyboardState(bKeyState);
#0068  #ifdef UNICODE
#0069       if (ToUnicode(wParam, uScanCodeAndShiftState,
#0070          bKeyState, ac, 1, 0) == 1) {
#0071  #else
#0072       if (ToAscii(wParam, uScanCodeAndShiftState,
#0073          bKeyState, (PWORD) ac, 0) == 1) {
#0074  #endif
#0075       TCHAR sz[256];
#0076       FORWARD_WM_GETTEXT(g_hwndEdit, adgARRAY_SIZE(sz), sz, SendMessage);
#0077       if (_tcschr(sz, ac[0])) {
#0078
#0079            // Reject the key event.
#0080            lResult = 1;
#0081          }
#0082       }
#0083    }
#0084    return(lResult);
```

```
#0085 }
#0086
#0087
#0088 ////////////////////////////////////////////////////////////
#0089
#0090
#0091 BOOL WINAPI KeyboardHook_Start (HWND hwndPost,
#0092    UINT uMsgNotify, HWND hwndEdit) {
#0093
#0094    HHOOK hhook;
#0095
#0096    // Return FALSE if hook has already been installed.
#0097    if (g_hhook != NULL)
#0098       return(FALSE);
#0099
#0100    adgASSERT(IsWindow(hwndPost));
#0101
#0102    g_hwndPost   = hwndPost;
#0103    g_uMsgNotify = uMsgNotify;
#0104    g_hwndEdit   = hwndEdit;
#0105
#0106    // Give up the remainder of our thread's timeslice.
#0107    // This gives us a better chance of getting all the way through the call
#0108    // to SetWindowsHookEx and the variable assignment to g_hhook in one shot.
#0109    // If we are preempted after the hook is set, but before the variable is
#0110    // updated, it is possible for another thread to enter our hook filter
#0111    // function before the hook handle is valid.  Under Windows NT this is not
#0112    // a problem.  Under Windows 95, not having a valid hook handle will cause
#0113    // CallNextHookEx to fail.  If there is some reason that it is critical
#0114    // that your application succeed in calling the next filter function in
#0115    // the chain, the only robust way to write this code is to use something
#0116    // like the SWMRG (single-writer, multiple-reader guard) object developed
#0117    // in Advanced Windows (Microsoft Press).
#0118    Sleep(0);
#0119
#0120    // Set our keyboard hook.
#0121    hhook = SetWindowsHookEx(WH_KEYBOARD,
#0122       KeyboardHook_HookProc, g_hinstDll, 0);
#0123
#0124    // Ensure that g_hhook is always valid (even if we are preempted whilst
#0125    // in the middle of writing to it) by updating the variable atomically.
#0126    InterlockedExchange((PLONG) &g_hhook, (LONG) hhook);
#0127
#0128    return(g_hhook != NULL);
#0129 }
#0130
```

```
#0131
#0132 ////////////////////////////////////////////////////////////////
#0133
#0134
#0135 BOOL WINAPI KeyboardHook_Stop () {
#0136
#0137    BOOL fOK = TRUE;
#0138
#0139    // Only uninstall the hook if it was successfully installed.
#0140    if (g_hhook != NULL) {
#0141       fOK = UnhookWindowsHookEx(g_hhook);
#0142       g_hhook = NULL;
#0143    }
#0144    return(fOK);
#0145 }
#0146
#0147
#0148 ////////////////////////////////////////////////////////////////
#0149
#0150
#0151 BOOL WINAPI DllMain (HINSTANCE hinstDll, DWORD fdwReason, LPVOID lpvReserved) {
#0152
#0153    switch (fdwReason) {
#0154
#0155       case DLL_PROCESS_ATTACH:
#0156          g_hinstDll = hinstDll;
#0157          break;
#0158    }
#0159    return(TRUE);
#0160 }
#0161
#0162
#0163 ////////////////////////// End of File ///////////////////////////
```

程式列表 6.7　KybdHk.H

```
#0001 /****************************************************************
#0002 Module name: KybdHk.h
#0003 Written by: Jonathan Locke
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Keyboard hooks dll function declarations
#0006 ****************************************************************/
#0007
#0008
#0009 #ifndef KYBDHKLIBAPI
```

```
#0010  #define KYBDHKLIBAPI  __declspec(dllimport)
#0011  #endif
#0012
#0013
#0014  ////////////////////////////////////////////////////////////////
#0015
#0016
#0017  KYBDHKLIBAPI BOOL WINAPI KeyboardHook_Start (HWND hwndPost,
#0018     UINT uMsgNotify, HWND hwndEdit);
#0019  KYBDHKLIBAPI BOOL WINAPI KeyboardHook_Stop ();
#0020
#0021
#0022  ////////////////////////// End of File ////////////////////////////
```

AppLog 程式

AppLog 程式（AppLog.EXE）的原始碼顯示於程式列表 **6.8 ~ 6.12** 之中，示範如何使用 shell hook 來監視視窗的產生與關閉。當你執行 AppLog 程式，主視窗如圖 **6.3** 所示。

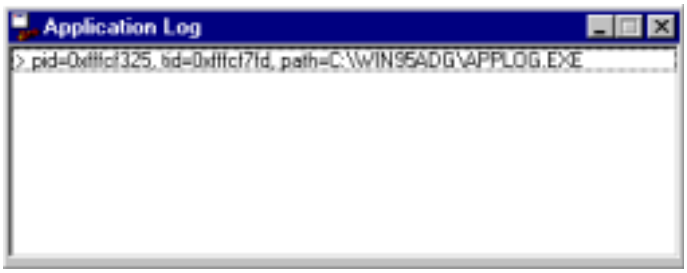


圖 6.3　AppLog 程式

AppLog 是一個非常簡單的程式，只內含一個控制元件：listbox。每當一個程式開始或結束，其行程 ID、執行緒 ID、以及程式路徑都會被加到 AppLog 程式的 listbox 之中。在執行過 Notepad 以及 MSPaint 之後，AppLog 程式畫面如圖 **6.4**。

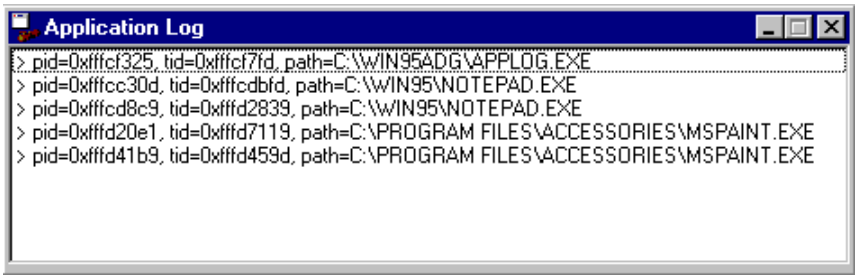


圖 6.4　執行 Notepad 和 MSPaint 程式後的 AppLog 畫面

表 **6.4** 和表 **6.5** 列出此程式及其 DLL 所需用到的檔案。由於 AppLog 程式安裝的是一個 system-wide WH_SHELL hook，所以其 filter function 必須寫在 DLL 之中。

表 6.4　建立 AppLog 程式所需的檔案

檔案	說明
AppLog.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
AppLog.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 AppLog.RC 檔中所有資源的 ID。
AppLog.ICO	主視窗圖示（icon）。
AppLog.MAK	Visual C++ 的 MAK 檔。

表 6.5　建立 ShellHk.DLL 所需的檔案

檔案	說明
ShellHk.C	內含 system-wide shell hook 的實作內容。此 hook 用來攔截視窗的產生和關閉（發生於程式開始之後和結束之前）。
ShellHk.H	內含 ShellHook API 函式的原型宣告，它們被用來安裝或卸除 shell hook。
ShellHk.MAK	Visual C++ 的 MAK 檔，用來產生 Shellhk.DLL。

這個 ShellHk.DLL 輸出（exports）兩個函式：

```
SHELLHKLIBAPI BOOL WINAPI ShellHook_Start (HWND hwndListBox);
SHELLHKLIBAPI BOOL WINAPI ShellHook_Stop ();
```

當 AppLog 程式開始執行，它會呼叫 ShellHook_Start 函式，傳入主視窗中的 listbox 視窗代碼。ShellHook_Start 函式會安裝一個 system-wide hook，其目的僅是將此後執行的程式所產生的視窗，以及所關閉的視窗的相關資訊放置在 listbox 中：

```
BOOL WINAPI ShellHook_Start (HWND hwndListBox) {

   HHOOK hhook = NULL;

   // Return FALSE if hook has already been installed.
   if (g_hhook != NULL)
      return(FALSE);

   adgASSERT(IsWindow(hwndListBox));
   g_hwndListBox = hwndListBox;

   // Give up the remainder of our thread's timeslice.
   // This gives us a better chance of getting all the way through the call
   // to SetWindowsHookEx and the variable assignment to g_hhook in one shot.
   // If we are preempted after the hook is set, but before the variable is
   // updated, it is possible for another thread to enter our hook filter
   // function before the hook handle is valid.  Under Windows NT this is not
   // a problem.  Under Windows 95, not having a valid hook handle will cause
   // CallNextHookEx to fail.  If there is some reason that it is critical
   // that your application succeed in calling the next filter function in
   // the chain, the only robust way to write this code is to use something
   // like the SWMRG (single-writer, multiple-reader guard) object developed
   // in Advanced Windows (Microsoft Press).
   Sleep(0);

   // Set our keyboard hook.
   hhook = SetWindowsHookEx(WH_SHELL, ShellHook_HookProc, g_hinstDll, 0);

   // Ensure that g_hhook is always valid (even if we are preempted whilst
   // in the middle of writing to it) by updating the variable atomically.
   InterlockedExchange((PLONG) &g_hhook, (LONG) hhook);

   return(g_hhook != NULL);
}
```

SetWindowsHookEx 函式的 dwThreadID 參數爲 0，表示所掛上的 hook 爲 system-wide，hmod 參數爲 ShellHk DLL 的執行個體代碼（instance handle），這個值會被 DllMain 存放在全域變數 g_hinstDll 中。由於 ShellHk 和 KeyCount 程式中的 KybdHk 一樣都是 remote hooks，所以同樣也有先前在討論 KeyCount 程式時提過的共享變數問題和執行緒同步問題。

當 ShellHook_HookProc 函式收到了一個 HSHELL_WINDOWCREATE 或 HSHELL_WINDOWDESTROY hook code，它會在 listbox 中加入一筆項目，描述視窗被產生或被摧毀的相關資訊：

```
static LRESULT WINAPI ShellHook_HookProc (int nCode,
  WPARAM wParam, LPARAM lParam) {

  TCHAR sz[128], szPath[128];
  DWORD dwProcessID;
  HWND hwnd = (HWND) wParam;
  DWORD dwThreadID = GetWindowThreadProcessId(hwnd, &dwProcessID);
  LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);

  switch (nCode) {

    case HSHELL_WINDOWCREATED:
    case HSHELL_WINDOWDESTROYED:

      // GetModuleFileName is being called in the context of the process in
      // which this Dll has been injected.
      if (!GetModuleFileName(NULL, szPath, adgARRAY_SIZE(szPath)))
        _tcscpy(szPath, __TEXT("<unknown>"));

      wsprintf(sz, __TEXT("%c pid=0x%08x, tid=0x%08x, path=%s"),
        (nCode == HSHELL_WINDOWCREATED ? __TEXT('>') : __TEXT('<')),
        dwProcessID, dwThreadID, szPath);

      ListBox_AddString(g_hwndListBox, sz);
      break;
  }

  return(lResult);
}
```

ShellHook_HookProc 函式會呼叫 GetWindowThreadID 以獲得與「被產生視窗或被摧毀視窗」相關之執行緒 ID 和行程 ID。接著它又呼叫 GetModuleFileName 函式，設定其 HMODULE 參數爲 NULL，以獲得與此視窗相關之執行程式的完整路徑。通常你可能會以爲 GetModuleFileName(NULL,…) 傳回的是目前行程 AppLog 的路徑名稱，但是不要忘了，ShellHk.DLL 已經被注射到其它行程的位址空間中了。由於 GetModuleFileName 函式是在該行程中被呼叫，所以它所取得的是該程式的路徑名稱，而不是 AppLog.EXE 的路徑名稱。

AppLog.ICO

程式列表 6.8　AppLog.C

```
#0001  /******************************************************************
#0002  Module name: AppLog.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Shell hooks demonstration application
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single line comment */
#0013  #include <tchar.h>
#0014  #include "ShellHk.h"
#0015  #include "Resource.h"
#0016
#0017
#0018  ////////////////////////////////////////////////////////////////////
#0019
#0020
#0021  // Force a link with the import library for ShellHk.dll
#0022  #pragma comment(lib, adgLIBBUILDTYPE adgLIBCPUTYPE "\\" "ShellHk")
#0023
#0024
#0025  ////////////////////////////////////////////////////////////////////
#0026
#0027
#0028  BOOL AppLog_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0029
#0030    adgSETDLGICONS(hwnd, IDI_APPLOG, IDI_APPLOG);
#0031    if (!ShellHook_Start(GetDlgItem(hwnd, IDC_APPLOG))) {
```

```
#0032       adgMB(__TEXT("Unable to start shell hook"));
#0033       EndDialog(hwnd, IDCANCEL);
#0034    }
#0035    return(TRUE);               // Accept default focus window.
#0036 }
#0037
#0038
#0039 ////////////////////////////////////////////////////////////////
#0040
#0041
#0042 void AppLog_OnSize (HWND hwnd, UINT state, int cx, int cy) {
#0043
#0044    // When the user resizes the main window, we must resize the listbox
child.
#0045    SetWindowPos(GetDlgItem(hwnd, IDC_APPLOG), NULL,
#0046       0, 0, cx, cy, SWP_NOZORDER);
#0047 }
#0048
#0049
#0050 ////////////////////////////////////////////////////////////////
#0051
#0052
#0053
#0054 void AppLog_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0055
#0056    switch (id) {
#0057       case IDCANCEL:            // Allows dialog box to close
#0058
#0059          // Unhook shell hook before exiting.
#0060          if (!ShellHook_Stop())
#0061             adgMB(__TEXT("Unable to stop shell hook"));
#0062          EndDialog(hwnd, id);
#0063          break;
#0064    }
#0065 }
#0066
#0067
#0068 ////////////////////////////////////////////////////////////////
#0069
#0070
#0071
#0072 BOOL WINAPI AppLog_DlgProc (HWND hwnd, UINT uMsg,
#0073    WPARAM wParam, LPARAM lParam) {
#0074
#0075    switch (uMsg) {
#0076
```

```
#0077       // Standard Windows messages
#0078       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, AppLog_OnInitDialog);
#0079       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    AppLog_OnCommand);
#0080       adgHANDLE_DLGMSG(hwnd, WM_SIZE,      AppLog_OnSize);
#0081    }
#0082    return(FALSE);              // We didn't process the message.
#0083 }
#0084
#0085
#0086 ////////////////////////////////////////////////////////////////
#0087
#0088
#0089
#0090 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0091    LPSTR lpszCmdLine, int nCmdShow) {
#0092
#0093    adgWARNIFUNICODEUNDERWIN95();
#0094    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_APPLOG),
#0095       NULL, AppLog_DlgProc));
#0096
#0097    return(0);
#0098 }
#0099
#0100
#0101 /////////////////////////// End of File ///////////////////////////
```

程式列表 6.9　AppLog.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "resource.h"
#0004
#0005 #define APSTUDIO_READONLY_SYMBOLS
#0006 ////////////////////////////////////////////////////////////////
#0007 //
#0008 // Generated from the TEXTINCLUDE 2 resource.
#0009 //
#0010 #define APSTUDIO_HIDDEN_SYMBOLS
#0011 #include "windows.h"
#0012 #undef APSTUDIO_HIDDEN_SYMBOLS
#0013
#0014 ////////////////////////////////////////////////////////////////
#0015 #undef APSTUDIO_READONLY_SYMBOLS
#0016
#0017
```

```
#0018 //////////////////////////////////////////////////////////////////
#0019 //
#0020 // Icon
#0021 //
#0022
#0023 IDI_APPLOG           ICON    DISCARDABLE     "APPLOG.ICO"
#0024
#0025 //////////////////////////////////////////////////////////////////
#0026 //
#0027 // Dialog
#0028 //
#0029
#0030 IDD_APPLOG DIALOG DISCARDABLE  0x8000, 5, 252, 76
#0031 STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME
#0032 CAPTION "Application Log"
#0033 FONT 8, "MS Sans Serif"
#0034 BEGIN
#0035     LISTBOX        IDC_APPLOG,4,4,243,68,NOT LBS_NOTIFY | WS_VSCROLL |
#0036                    WS_TABSTOP
#0037 END
#0038
#0039
#0040 #ifdef APSTUDIO_INVOKED
#0041 //////////////////////////////////////////////////////////////////
#0042 //
#0043 // TEXTINCLUDE
#0044 //
#0045
#0046 1 TEXTINCLUDE DISCARDABLE
#0047 BEGIN
#0048     "resource.h\0"
#0049 END
#0050
#0051 2 TEXTINCLUDE DISCARDABLE
#0052 BEGIN
#0053     "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0054     "#include ""windows.h""\r\n"
#0055     "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0056     "\0"
#0057 END
#0058
#0059 3 TEXTINCLUDE DISCARDABLE
#0060 BEGIN
#0061     "\r\n"
#0062     "\0"
#0063 END
```

```
#0064
#0065  //////////////////////////////////////////////////////////////
#0066  #endif    // APSTUDIO_INVOKED
#0067
#0068
#0069  #ifndef APSTUDIO_INVOKED
#0070  ////////////////////////////////////////////////////////////////
#0071  //
#0072  // Generated from the TEXTINCLUDE 3 resource.
#0073  //
#0074
#0075
#0076  //////////////////////////////////////////////////////////////
#0077  #endif    // not APSTUDIO_INVOKED
```

程式列表 6.10　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by AppLog.RC
#0004  //
#0005  #define IDD_APPLOG                  101
#0006  #define IDC_APPLOG                  1000
#0007  #define IDI_APPLOG                  1001
#0008
#0009  // Next default values for new objects
#0010  //
#0011  #ifdef APSTUDIO_INVOKED
#0012  #ifndef APSTUDIO_READONLY_SYMBOLS
#0013  #define _APS_NO_MFC                  1
#0014  #define _APS_NEXT_RESOURCE_VALUE       102
#0015  #define _APS_NEXT_COMMAND_VALUE        40001
#0016  #define _APS_NEXT_CONTROL_VALUE        1002
#0017  #define _APS_NEXT_SYMED_VALUE         101
#0018  #endif
#0019  #endif
```

程式列表 6.11　ShellHk.C

```
#0001  /************************************************************
#0002  Module name: ShellHk.c
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
```

```
#0005  Purpose: Dll which sets a system shell hook.
#0006  ************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include <tchar.h>
#0014  #pragma warning(disable: 4001)   /* Single line comment */
#0015
#0016
#0017  // We must define SHELLHKLIBAPI as 'dllexport' before including ShellHk.h.
#0018  // ShellHk.h will see that we have already defined SHELLHKLIBAPI and
#0019  // will not (re)define it as 'dllimport'.
#0020  #define SHELLHKLIBAPI __declspec(dllexport)
#0021  #include "ShellHk.h"
#0022
#0023
#0024  ////////////////////////////////////////////////////////////////
#0025
#0026
#0027  HINSTANCE g_hinstDll = NULL;     // ShellHk.dll's instance handle
#0028
#0029
#0030  ////////////////////////////////////////////////////////////////
#0031
#0032
#0033  // Calling SetWindowsHookEx with a thread id of zero will cause this DLL to be
#0034  // injected into other processes. Therefore, we must declare a shared data
#0035  // section so that all mappings of our DLL (in all processes) share the same
#0036  // set of global variables.
#0037
#0038  #pragma comment(lib, "kernel32 " "-section:Shared,rws")
#0039  #pragma data_seg("Shared")
#0040
#0041  HHOOK g_hhook      = NULL;    // Hook handle for systemwide shell hook
#0042  HWND  g_hwndListBox = NULL;    // Listbox in which to insert log strings
#0043
#0044  #pragma data_seg()
#0045
#0046
#0047  ////////////////////////////////////////////////////////////////
#0048
#0049
#0050  static LRESULT WINAPI ShellHook_HookProc (int nCode,
```

```
#0051    WPARAM wParam, LPARAM lParam) {
#0052
#0053    TCHAR sz[128], szPath[128];
#0054    DWORD dwProcessID;
#0055    HWND hwnd = (HWND) wParam;
#0056    DWORD dwThreadID = GetWindowThreadProcessId(hwnd, &dwProcessID);
#0057    LRESULT lResult = CallNextHookEx(g_hhook, nCode, wParam, lParam);
#0058
#0059    switch (nCode) {
#0060
#0061       case HSHELL_WINDOWCREATED:
#0062       case HSHELL_WINDOWDESTROYED:
#0063
#0064          // GetModuleFileName is being called in the context of the process in
#0065          // which this Dll has been injected.
#0066          if (!GetModuleFileName(NULL, szPath, adgARRAY_SIZE(szPath)))
#0067             _tcscpy(szPath, __TEXT("<unknown>"));
#0068
#0069          wsprintf(sz, __TEXT("%c pid=0x%08x, tid=0x%08x, path=%s"),
#0070             (nCode == HSHELL_WINDOWCREATED ? __TEXT('>') : __TEXT('<')),
#0071             dwProcessID, dwThreadID, szPath);
#0072
#0073          ListBox_AddString(g_hwndListBox, sz);
#0074          break;
#0075    }
#0076
#0077    return(lResult);
#0078 }
#0079
#0080
#0081 ////////////////////////////////////////////////////////////////
#0082
#0083
#0084 BOOL WINAPI ShellHook_Start (HWND hwndListBox) {
#0085
#0086    HHOOK hhook = NULL;
#0087
#0088    // Return FALSE if hook has already been installed.
#0089    if (g_hhook != NULL)
#0090       return(FALSE);
#0091
#0092    adgASSERT(IsWindow(hwndListBox));
#0093    g_hwndListBox = hwndListBox;
#0094
#0095    // Give up the remainder of our thread's timeslice.
#0096    // This gives us a better chance of getting all the way through the call
```

```
#0097    // to SetWindowsHookEx and the variable assignment to g_hhook in one shot.
#0098    // If we are preempted after the hook is set, but before the variable is
#0099    // updated, it is possible for another thread to enter our hook filter
#0100    // function before the hook handle is valid.  Under Windows NT this is not
#0101    // a problem.  Under Windows 95, not having a valid hook handle will cause
#0102    // CallNextHookEx to fail.  If there is some reason that it is critical
#0103    // that your application succeed in calling the next filter function in
#0104    // the chain, the only robust way to write this code is to use something
#0105    // like the SWMRG (single-writer, multiple-reader guard) object developed
#0106    // in Advanced Windows (Microsoft Press).
#0107    Sleep(0);
#0108
#0109    // Set our keyboard hook.
#0110    hhook = SetWindowsHookEx(WH_SHELL, ShellHook_HookProc, g_hinstDll, 0);
#0111
#0112    // Ensure that g_hhook is always valid (even if we are preempted whilst
#0113    // in the middle of writing to it) by updating the variable atomically.
#0114    InterlockedExchange((PLONG) &g_hhook, (LONG) hhook);
#0115
#0116    return(g_hhook != NULL);
#0117 }
#0118
#0119
#0120 ////////////////////////////////////////////////////////////////
#0121
#0122
#0123 BOOL WINAPI ShellHook_Stop () {
#0124
#0125    BOOL fOK = TRUE;
#0126
#0127    // Only uninstall the hook if it was successfully installed.
#0128    if (g_hhook != NULL) {
#0129       fOK = UnhookWindowsHookEx(g_hhook);
#0130       g_hhook = NULL;
#0131    }
#0132    return(fOK);
#0133 }
#0134
#0135
#0136 ////////////////////////////////////////////////////////////////
#0137
#0138
#0139 BOOL WINAPI DllMain (HINSTANCE hinstDll, DWORD fdwReason, LPVOID lpvReserved) {
#0140
#0141    switch (fdwReason) {
#0142
```

```
#0143       case DLL_PROCESS_ATTACH:
#0144          g_hinstDll = hinstDll;
#0145          break;
#0146    }
#0147    return(TRUE);
#0148 }
#0149
#0150
#0151 ////////////////////// End of File ////////////////////////
```

程式列表 6.12　ShellHk.H

```
#0001 /************************************************************
#0002 Module name: ShellHk.h
#0003 Written by: Jonathan Locke
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Shell hooks dll function declarations
#0006 ************************************************************/
#0007
#0008
#0009 // SHELLHKLIBAPI is defined as dllexport in the implementation file
#0010 // (shellhk.c). Thus, when this is included by shellhk.c, functions will be
#0011 // exported instead of imported.
#0012 #ifndef SHELLHKLIBAPI
#0013 #define SHELLHKLIBAPI  __declspec(dllimport)
#0014 #endif
#0015
#0016
#0017 ////////////////////////////////////////////////////////////////
#0018
#0019
#0020 SHELLHKLIBAPI BOOL WINAPI ShellHook_Start (HWND hwndListBox);
#0021 SHELLHKLIBAPI BOOL WINAPI ShellHook_Stop ();
#0022
#0023
#0024 ////////////////////////// End of File ////////////////////////
```

Echo 程式（巨集記錄器）

Echo 程式(Echo.EXE)的原始碼顯示於**程式列表 6.13 ~ 6.19** 之中，示範如何使用 journal record 和 journal playback 建立一個巨集記錄器，用以記錄和播放硬體的 input events 。

當你執行這個此程式，主視窗如圖 **6.5**。

圖 6.5　Echo 程式

就像大部份的記錄器一樣，Echo 程式有三個按鈕：【記錄（Record）】、【播放（Play）】和【停止（Stop）】。程式一開始，【Play】和【Stop】鈕沒有作用。一旦你按下【Record】鈕，程式就開始記錄 events ，此時【Stop】鈕開始有作用，允許你停止記錄動作。當【Stop】鈕被按下，【Stop】鈕本身又會被除能（disabled），【Play】和【Record】鈕則會被致能（enabled）。此時如果你按下【Play】鈕，Echo 程式將會播放之前所記錄的硬體 input events（包括滑鼠、鍵盤和筆式輸入）。

建立此程式所需用到的檔案，列於表 **6.6** 中。

表 6.6 建立 Echo 程式所需的檔案

檔案	說明
Echo.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Echo.MSG	MsgCrack 的輸入檔，用來產生 Echo.H（請參閱附錄 B）。
Echo.H	定義 UM_STOP 通告訊息與其訊息剖析器。
Record.C	建立巨集記錄器。
Record.H	列舉各種錯誤碼及記錄器模式，以及函式原型宣告。
Echo.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 Echo.RC 檔中所有資源的 ID。
Echo.ICO	主視窗圖示（icon）。
Echo.MAK	Visual C++ 的 MAK 檔。

Echo 程式由兩個部份組成：Echo 程式碼，用來產生主視窗和維護使用者介面；Recorder 程式碼，巨集記錄器的核心，其內函式皆可獨立使用，你可以輕易把它們放到你自己的程式中使用。

Echo 程式碼

當 Echo 程式產生其主視窗時，它呼叫 Recorder_Init 來初始化巨集記錄器。接著使用 Echo_UpdateButtons 函式依目前記錄器的狀態，適當地致能（enabled）或除能（disabled）【Play】、【Record】和【Stop】三個按鈕。你可以利用 Recorder_GetMode 函式得知目前記錄器的狀態，它的傳回值是以下三者之一：

```
typedef enum {
   RECMODE_STOPPED,          // Recorder is stopped
   RECMODE_RECORDING,        // Recorder is recording events
   RECMODE_PLAYING           // Recorder is playing back events
} RECMODE;
```

每當記錄器的模式改變，按鈕也需做適當的改變（致能或除能）。Echo 程式碼幾乎都是在做這些動作，也就是呼叫 Echo_UpdateButtons 函式來改變按鈕狀態。

當使用者按下 Echo 程式中的一個按鈕，對應的記錄器函式會被喚起：

```
// Returns: RECERR_OK, RECERR_CANTHOOK, RECERR_ACTIVE or RECERR_NOMEMORY
RECERR WINAPI Recorder_Record (HWND hendNotify, UINT uMsgNotify);

// Returns: RECERR_OK, RECERR_CANTHOOK, RECERR_ACTIVE or RECERR_NOEVENTS
RECERR WINAPI Recorder_Play (HEVENTLIST h, HWND hwndNotify, UINT uMsgNotify);

// Returns: RECERR_OK or RECERR_INACTIVE
RECERR WINAPI Recorder_Stop (void);
```

當你按下【Record】鈕，它會呼叫 Recorder_Record 函式，傳入自己的視窗代碼和 UM_STOP 通知訊息。這時記錄器便開始記錄硬體的 input events ，直到結束（可能是因爲使用者按下【Stop】鈕，也可能是因爲有錯誤發生）。當記錄器停止，它會送出 UM_STOP 通

告訊息給主視窗，於是主視窗這樣處理這個訊息：

```
void Echo_OnStop (HWND hwnd, HEVENTLIST h, RECERR e) {

   // Echo_OnStop can be called when either recording or playing stops. If the
   // HEVENTLIST parameter is non-NULL, Echo_OnStop was called because
   // recording stopped, which means we need to save the HEVENTLIST for future
   // playback.
   if (h != NULL)
      SETEVENTLIST(hwnd, h);
   Echo_UpdateButtons(hwnd);
   Recorder_DisplayError(e);
}
```

如果 HEVENTLIST[12] 參數 h 是 NULL 的話，表示 Echo_OnStop 函式之所以被呼叫，是因爲播放時【Stop】鈕被按下，否則它之所以被呼叫是因爲記錄時【Stop】鈕被按下，這時候 h 是一個 events 串列（event list）代碼；當使用者按下【Play】鈕，程式會呼叫 Recorder_Play 函式，並利用 h 代碼來重播剛剛記錄的 events 。如果是因爲錯誤發生而收到 UM_STOP 通告訊息，第三個參數將會是下列錯誤代碼之一：

```
typedef enum {
   RECERR_OK,       // Operation was successful
   RECERR_CANTHOOK,  // Hook cannot be installed
   RECERR_ACTIVE,    // Attempt to record/play while already
recording/playing
   RECERR_INACTIVE,  // Attempt to stop recording while NOT recording
   RECERR_NOMEMORY,  // When attempting to start recording or during
recording
   RECERR_NOEVENTS,  // Attempt playback with no events in memory block
   RECERR_SYSMODAL,  // System modal dialog box canceled recording
   RECERR_USERCANCEL // User canceled with Ctrl+Esc
} RECERR;
```

Echo 會使用 Recorder_DisplayError 函式來顯示錯誤訊息。如果沒有錯誤發生，此函式就

[12] HEVENTLIST 是一種代碼（handle），宣告在 Recorder.H 檔中。如果記錄器要在未來改變 event 串列（event list）的結構，你並不需要中止使用任何正在使用該記錄器的程式。

什麼都不做。

這裡只有一個小麻煩。當使用者按下 Ctrl+Esc，系統會自動卸除任何 journal record hooks 和 journal playback hooks，並“post”一個 WM_CANCELJOURNAL 訊息到任何已掛上此類 hooks 的執行緒佇列中。記錄器需要取得這個訊息，它才能夠通知 Echo 程式說記錄動作已停止（如此一來記錄器才會重設其 hook handle 爲 NULL）。

不幸的是，WM_CANCELJOURNAL 訊息並不會被 posted 至 Echo 的主視窗，而是以一個 NULL 視窗代碼直接 posted 至 Echo 程式的訊息佇列中。有兩個方法可以取得這個訊息：第一種方法是在該訊息即將從訊息佇列中被取走時攔截之，作法是掛上一個 WH_GETMESSAGE 或 WH_(SYS)MSGFILTER hook。第二種方法是修改 Echo 程式，使它擁有自己的訊息迴路，這麼一來當它以 GetMessage 函式取得訊息時，我們就可以攔截之。我選擇第二種方法。爲了讓事情更容易些，記錄器提供了一個名爲 Recorder_IsRecorderCanceled 的函式，用來尋找 WM_CANCELJOURNAL 訊息：

```
BOOL WINAPI Recorder_IsRecorderCanceled (PMSG pmsg);
```

Recorder_IsRecorderCanceled 函式檢查訊息是否爲 WM_CANCELJOURNAL 訊息，其方法和 IsDialogMessage 函式檢查按鍵訊息是一樣的。Echo 程式在主訊息迴路中呼叫 Recorder_IsRecorderCanceled 函式。如果 Recorder_IsRecorderCanceled 函式偵測到 WM_CANCELJOURNAL 訊息，它就重設 hook handle 爲 NULL，並將記錄器的錯誤代碼設爲 RECERR_USERCANCEL，用以通知 Echo 程式說，記錄動作或播放動作已經停止了。

當 Echo 程式結束，它會呼叫 Recorder_Free 函式以釋放任何現存的 events 串列：

```
void WINAPI Recorder_Free (HEVENTLIST h);
```

Recorder 程式碼

在記錄器（recorder）程式中，有一個全域變數記錄了記錄器在任何時間的全部狀態。這個全域變數 g_RecorderData 是一個結構，定義如下：

```
typedef struct {
   HHOOK     hhookJournal;        // Journal hook handle
   RECMODE   RecMode;             // Recorder's operating mode
   HWND      hwndNotify;          // Window to notify
   UINT      uMsgNotify;          // Notification message to send
   RECERR    RecErr;              // Error that halted recording
   PRECSTAT  pRecStat;            // Recorded data
} RECORDERDATA;
```

其中 hhookJournal 欄位記錄的是目前的 journal record hook 或 journal playback hook 的 hook handle。如果目前並沒有掛上任何 hook，hhookJournal 應該是 NULL。目前的記錄器模式（RECMODE_*）記錄在 RecMode 欄位中。hwndNotify 和 uMsgNotify 欄位指出一個目標視窗代碼和一個通告訊息，當記錄動作停止，該目標視窗就會獲得那個通告。RecErr 欄位用來表示記錄動作停止原因（以 RECERR_* 表示）。最重要的一個欄位是 pRecStat，它指向一系列被記錄下來的 events。這個串列由一個 RECTSTAT 表頭結構開始，然後是一系列的 EVENTMSG 結構（定義在 WinUser.H 檔中）。RECTSTAT 表頭結構內含被記錄或待播放之 events 的統計資訊：

```
// Statistical information that appears at start of EVENTMSG list.
typedef struct {
   int nNumEvents;               // Number of recorded events
   int nNumEventsPlayed;          // Number of events played back
   DWORD dwStartTime;             // Time when playback started
} RECSTAT, *PRECSTAT;
```

當記錄器被要求開始記錄 events，它會先掛上一個 WH_JOURNALRECORD hook。其 filter function，也就是 Recorder_JournalRecordProc，會等待 HC_ACTION hook code 的到來，並記錄每一個到來的 EVENTMSG 結構，加入到 g_RecorderData.pRecStat 所代表的 events 串列中。如果系統所剩記憶體不足以記錄這些 events，記錄動作會停止下來，並傳回錯

誤代碼 RECERR_NOMEMORY。如果 Recorder_JournalRecordProc 函式收到一個 HC_SYSMODALON hook code[13]，它會將靜態變數 fPause 設定爲 TRUE。只要這個變數爲 TRUE，Recorder_JournalRecordProc 函式便會忽略所有的 HC_ACTION hook code。一旦 system-modal 訊息視窗被移除，filter function 會收到一個 HC_SYSMODALOFF hook code，而記錄器會以 RECERR_SYSMODAL 的原因停止記錄。

只要記錄器停止，程式就會呼叫 UnhookWindowsHookEx 函式，卸除目前已掛上的任何 journal record hook 或 journal playback hook。然後，如果目前記錄器的模式是 RECMODE_RECORDING，那麼程式開始修改所有 EVENTMSG 結構中的 time 欄位，給每一個 time 欄位一個相對的 timestamp（也就是減去記錄動作的開始時刻）。這個相對的 timestamp 能確保 events 播放速度與其原來記錄時的速度相同。最後，Recorder_Stop 函式會送出 g_RecorderData.uMsgNotify 所指定的通告訊息給 g_RecorderData.hwndNotify 所指定的視窗。如果是在記錄過程中被停止，那麼這個通告訊息的 wParam 參數將是目前的 events 串列：g_RecorderData.pRecStat。它會被轉型爲 HEVENTLIST。如果是在播放過程中被停止，那麼 wParam 參數爲 NULL。至於 lParam 參數，不論在何種情況下皆爲原儲存在 g_RecorderData.RecErr 的值。錯誤代碼可以表示記錄或播放動作終止的原因。如果錯誤代碼是 RECERR_OK，表示沒有任何錯誤發生。

當記錄器被要求播放一個 HEVENTLIST（所記錄的 events）時，它會掛上一個 journal playback hook。如果其 filter function，也就是 Recorder_JournalPlaybackProc，收到的 hook code 是 HC_GETNEXT，filter function 會傳回串列中的目前的 EVENTMSG 結構。如果 hook code 是 HC_SKIP，記錄器會跳至串列中的下一個 EVENTMSG 結構；如果後面沒有 event 了，播放動作就會停止。

---

[13] 一個 32 位元的程式永遠不會收到 HC_SYSMODAL*過濾碼，只有 16 位元的程式在 Windows 95 下執行才會收到這過濾碼。

Echo.ICO

程式列表 6.13　Echo.C

```
#0001  /****************************************************************
#0002  Module name: Echo.c
#0003  Written by: Jeffrey Richter and Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: A simple macro recorder application which demonstrates the use of
#0006          journal record and playback hooks.
#0007  ****************************************************************/
#0008
#0009
#0010  #include "..\Win95ADG.h"         /* See Appendix A for details */
#0011  #include <windows.h>
#0012  #include <windowsx.h>
#0013  #pragma warning(disable: 4001)    /* Single line comment */
#0014  #include "Resource.h"
#0015  #include "Echo.h"
#0016  #include "Record.h"
#0017  #pragma warning(disable: 4001)    /* Single line comment */
#0018
#0019
#0020  ////////////////////////////////////////////////////////////////
#0021
#0022
#0023  #define GETEVENTLIST(hwnd)    ((HEVENTLIST) GetWindowLong(hwnd, GWL_USERDATA))
#0024  #define SETEVENTLIST(hwnd, h) SetWindowLong(hwnd, GWL_USERDATA, (LONG) (h))
#0025
#0026
#0027  ////////////////////////////////////////////////////////////////
#0028
#0029
#0030  void Echo_UpdateButtons (HWND hwnd) {
#0031
#0032    RECMODE RecMode = Recorder_GetMode();
#0033
#0034    // The RECORD button should be enabled if the recorder is stopped.
#0035    BOOL fRec = (RecMode == RECMODE_STOPPED);
#0036
#0037    // The PLAY button should be enabled if the recorder is stopped and there
#0038    // are some events to be played back.
#0039    BOOL fPlay = (RecMode == RECMODE_STOPPED) && (GETEVENTLIST(hwnd) != NULL);
#0040
#0041    // The STOP button should be enabled if the recorder is not stopped.
#0042    BOOL fStop = (RecMode != RECMODE_STOPPED);
#0043
#0044    HWND hwndRec  = GetDlgItem(hwnd, IDC_RECORD);
```

```
#0045    HWND hwndPlay = GetDlgItem(hwnd, IDC_PLAY);
#0046    HWND hwndStop = GetDlgItem(hwnd, IDC_STOP);
#0047    HWND hwndFocus;
#0048
#0049    EnableWindow(hwndRec,  fRec);
#0050    EnableWindow(hwndPlay, fPlay);
#0051    EnableWindow(hwndStop, fStop);
#0052
#0053    hwndFocus = (fPlay ? hwndPlay : (fStop ? hwndStop : hwndRec));
#0054
#0055    SendMessage(hwnd, DM_SETDEFID, (fPlay ? IDC_PLAY :
#0056       (fStop ? IDC_STOP : IDC_RECORD)), 0);
#0057
#0058    SetFocus(hwndFocus);
#0059  }
#0060
#0061
#0062  ////////////////////////////////////////////////////////////////
#0063
#0064
#0065  BOOL Echo_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0066
#0067    adgSETDLGICONS(hwnd, IDI_ECHO, IDI_ECHO);
#0068    SETEVENTLIST(hwnd, NULL);
#0069    Recorder_Init();
#0070    Echo_UpdateButtons(hwnd);
#0071    return(TRUE);               // Accept default focus window.
#0072  }
#0073
#0074
#0075  ////////////////////////////////////////////////////////////////
#0076
#0077
#0078  void Echo_OnStop (HWND hwnd, HEVENTLIST h, RECERR e) {
#0079
#0080    // Echo_OnStop can be called when either recording or playing stops.
#0081    // If the HEVENTLIST parameter is non-NULL, Echo_OnStop was called
#0082    // because recording stopped, which means we need to save the HEVENTLIST
#0083    // for future playback.
#0084    if (h != NULL)
#0085       SETEVENTLIST(hwnd, h);
#0086    Echo_UpdateButtons(hwnd);
#0087    Recorder_DisplayError(e);
#0088  }
#0089
#0090
```

```
#0091 ////////////////////////////////////////////////////////////////
#0092
#0093
#0094 void Echo_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0095
#0096    RECERR e = RECERR_OK;
#0097
#0098    switch (id) {
#0099
#0100       case IDCANCEL:            // Allows dialog box to close.
#0101
#0102          Recorder_Free(GETEVENTLIST(hwnd));
#0103
#0104          // Terminate the application when user closes with system menu.
#0105          PostQuitMessage(0);
#0106          break;
#0107
#0108       case IDC_RECORD:
#0109
#0110          // If a macro was already recorded, free it.
#0111          if (GETEVENTLIST(hwnd) != NULL) {
#0112             Recorder_Free(GETEVENTLIST(hwnd));
#0113             SETEVENTLIST(hwnd, NULL);
#0114          }
#0115
#0116          // Recorder sends UM_STOP notification to hwnd when recording stops.
#0117          e = Recorder_Record(hwnd, UM_STOP);
#0118          Echo_UpdateButtons(hwnd);
#0119          Recorder_DisplayError(e);
#0120          break;
#0121
#0122       case IDC_PLAY:
#0123
#0124          // Recorder sends UM_STOP notification to hwnd when playing stops.
#0125          e = Recorder_Play(GETEVENTLIST(hwnd), hwnd, UM_STOP);
#0126          Echo_UpdateButtons(hwnd);
#0127          Recorder_DisplayError(e);
#0128          break;
#0129
#0130       case IDC_STOP:
#0131          e = Recorder_Stop();
#0132          Echo_UpdateButtons(hwnd);
#0133          Recorder_DisplayError(e);
#0134          break;
#0135    }
#0136 }
```

```
#0137
#0138
#0139 ////////////////////////////////////////////////////////////////
#0140
#0141
#0142 BOOL WINAPI Echo_DlgProc (HWND hwnd, UINT uMsg,
#0143    WPARAM wParam, LPARAM lParam) {
#0144
#0145    switch (uMsg) {
#0146
#0147       // Standard Windows messages
#0148       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, Echo_OnInitDialog);
#0149       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    Echo_OnCommand);
#0150
#0151       // Stop notification message sent by Recorder_Stop
#0152       adgHANDLE_DLGMSG(hwnd, UM_STOP,      Echo_OnStop);
#0153    }
#0154
#0155    return(FALSE);              // We didn't process the message.
#0156 }
#0157
#0158
#0159 ////////////////////////////////////////////////////////////////
#0160
#0161
#0162 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0163    LPSTR lpszCmdLine, int nCmdShow) {
#0164
#0165    MSG msg;
#0166    HWND hwnd;
#0167
#0168    adgWARNIFUNICODEUNDERWIN95();
#0169
#0170    // Create a modeless dialog box instead of a modal dialog box because we
#0171    // need to have more control over the message loop processing.
#0172    hwnd = CreateDialog(hinstExe, MAKEINTRESOURCE(IDD_ECHO), NULL,
#0173       Echo_DlgProc);
#0174    adgASSERT(IsWindow(hwnd));
#0175
#0176    // Continue to loop until a WM_QUIT message comes out of the queue.
#0177    while (GetMessage(&msg, NULL, 0, 0)) {
#0178
#0179       // A user can force local input state processing back on by pressing
#0180       // Ctrl+Esc. When this happens, the operating system unhooks the journal
#0181       // record/playback hook we have installed and notifies our thread by
#0182       // posting it a WM_CANCELJOURNAL message. The function
```

```
#0183       // Recorder_IsRecorderCanceled checks to see if the current message is a
#0184       // WM_CANCELJOURNAL message, and if so, resets the recorder's hook
#0185       // handle and stops recording with an error code of RECERR_USERCANCEL.
#0186       Recorder_IsRecorderCanceled (&msg);
#0187
#0188       // Call IsDialogMessage so that the keyboard can be used to control
#0189       // focus in the dialog box.
#0190       if (!IsDialogMessage(hwnd, &msg)) {
#0191          TranslateMessage(&msg);
#0192          DispatchMessage(&msg);
#0193       }
#0194    }
#0195
#0196    // The application is terminating; destroy the modeless dialog box.
#0197    DestroyWindow(hwnd);
#0198    return(0);
#0199 }
#0200
#0201
#0202 ////////////////////////// End of File ///////////////////////////
```

程式列表 6.14　Echo.MSG

```
#0001 // Module name: Echo.msg
#0002 // Written by: Jonathan Locke
#0003 // Notices: Copyright (c) 1995 Jeffrey Richter
#0004 // Purpose: 'MsgCrack' input file for Echo application.
#0005
#0006 MessageBase WM_APP
#0007 MessageClass Echo
#0008
#0009 Message UM_STOP Stop \
#0010  - User-defined notification message sent by Recorder_Stop (see Record.c)\
#0011  when recording or playing stops.
#0012 wParam HEVENTLIST h - Event list handle if it was recording that stopped.\
#0013  If it was playing that stopped, h will be NULL.
#0014 lParam RECERR e - Recorder error code.  In the event of failure, this error\
#0015  code indicates why playing/recording stopped.
#0016 .
```

程式列表 6.15　Echo.H

```
#0001 /*****************************************************************
```

```
#0002  Module name: Echo.h
#0003  Written by: Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Message cracker for UM_STOP notification message.
#0006  ************************************************************/
#0007
#0008  /////////////////////////// Messages ////////////////////////////
#0009
#0010
#0011  //{{adgMSGCRACK_MESSAGES
#0012
#0013  // Purpose: User-defined notification message sent by Recorder_Stop
#0014  //          (see Record.c) when recording or playing stops.
#0015  // wParam:  HEVENTLIST h - Event list handle if it was recording that stopped.
#0016  //          If it was playing that stopped, h will be NULL.
#0017  // lParam:  RECERR e - Recorder error code. In the event of failure, this error
#0018  //          code indicates why playing/recording stopped.
#0019  // Returns: void
#0020  #define UM_STOP            (WM_APP + 0)
#0021
#0022  //}}adgMSGCRACK_MESSAGES
#0023
#0024
#0025  ////////////////////////// Message APIs /////////////////////////////
#0026
#0027
#0028  //{{adgMSGCRACK_APIS
#0029
#0030  #define Echo_Stop(hwnd, h, e) \
#0031     ((void)SendMessage((hwnd), UM_STOP, (WPARAM)(DWORD)(h), (LPARAM)(DWORD)(e)))
#0032
#0033  //}}adgMSGCRACK_APIS
#0034
#0035
#0036  ///////////////////////// Message Crackers //////////////////////////
#0037
#0038
#0039  //{{adgMSGCRACK_CRACKERS
#0040
#0041  // void Cls_OnStop (HWND hwnd, HEVENTLIST h, RECERR e)
#0042  #define HANDLE_UM_STOP(hwnd, wParam, lParam, fn) \
#0043     ((fn)((hwnd), (HEVENTLIST)(wParam), (RECERR)(lParam)), 0)
#0044  #define FORWARD_UM_STOP(hwnd, h, e, fn) \
#0045     (void)((fn)((hwnd), UM_STOP, (WPARAM)(DWORD)(h), (LPARAM)(DWORD)(e)))
#0046
#0047  //}}adgMSGCRACK_CRACKERS
```

```
#0048
#0049
#0050  ////////////////////////// End of File //////////////////////////
```

程式列表 6.16　Record.C

```
#0001  /****************************************************************
#0002  Module name: Record.c
#0003  Written by: Jeffrey Richter and Jonathan Locke
#0004  Notices: Copyright (c) 1993 Jeffrey Richter
#0005  Purpose: Macro recorder implementation.
#0006  ****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"         /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #pragma warning(disable: 4001)    /* Single line comment */
#0012  #include "Record.h"
#0013  #pragma warning(disable: 4001)    /* Single line comment */
#0014
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017
#0018
#0019  // Statistical information that appears at start of EVENTMSG list.
#0020  typedef struct {
#0021     int nNumEvents;              // Number of recorded events
#0022     int nNumEventsPlayed;         // Number of events played back
#0023     DWORD dwStartTime;           // Time when playback started
#0024  } RECSTAT, *PRECSTAT;
#0025
#0026
#0027  ////////////////////////////////////////////////////////////////
#0028
#0029
#0030  typedef struct {
#0031     HHOOK    hhookJournal;       // Journal hook handle
#0032     RECMODE   RecMode;           // Recorder's operating mode
#0033     HWND     hwndNotify;         // Window to notify
#0034     UINT     uMsgNotify;         // Notification message to send
#0035     RECERR    RecErr;            // Error which halted recording
#0036     PRECSTAT  pRecStat;          // Recorded data
#0037  } RECORDERDATA;
#0038
#0039
```

```
#0040 ////////////////////////////////////////////////////////////////
#0041
#0042
#0043 static RECORDERDATA g_RecorderData;
#0044
#0045
#0046 ////////////////////////////////////////////////////////////////
#0047
#0048
#0049 void WINAPI Recorder_Init () {
#0050
#0051    adgINITSTRUCT(g_RecorderData, FALSE);
#0052
#0053    g_RecorderData.uMsgNotify = WM_NULL;
#0054    g_RecorderData.RecMode = RECMODE_STOPPED;
#0055    g_RecorderData.RecErr = RECERR_OK;
#0056 }
#0057
#0058
#0059 ////////////////////////////////////////////////////////////////
#0060
#0061
#0062 static LRESULT WINAPI Recorder_JournalRecordProc (int nCode,
#0063    WPARAM wParam, LPARAM lParam) {
#0064
#0065    static BOOL fPause = FALSE;
#0066    PRECSTAT pRecStat;
#0067    PEVENTMSG pEvent;
#0068    int nNumEvents;
#0069    LRESULT lResult = CallNextHookEx(g_RecorderData.hhookJournal,
#0070       nCode, wParam, lParam);
#0071
#0072    switch (nCode) {
#0073
#0074       case HC_ACTION:
#0075
#0076          // If system-modal dialog box is up, don't record event.
#0077          if (fPause)
#0078             break;
#0079
#0080          // Determine number of events in the memory block now.
#0081          nNumEvents = g_RecorderData.pRecStat->nNumEvents + 1;
#0082
#0083          // Increase size of the memory block to hold new event.
#0084          pRecStat = realloc(g_RecorderData.pRecStat,
#0085             sizeof(RECSTAT) + nNumEvents * sizeof(EVENTMSG));
```

```
#0086          if (pRecStat == NULL) {
#0087
#0088             // Insufficient memory; stop recording.
#0089             g_RecorderData.RecErr = RECERR_NOMEMORY;
#0090             Recorder_Stop();
#0091             break;
#0092          }
#0093          g_RecorderData.pRecStat = pRecStat;
#0094
#0095          // Append the new event to the end of the memory block.
#0096          pEvent = (PEVENTMSG) &g_RecorderData.pRecStat[1];
#0097          pEvent[g_RecorderData.pRecStat->nNumEvents] = *((PEVENTMSG) lParam);
#0098          g_RecorderData.pRecStat->nNumEvents++;
#0099          break;
#0100
#0101       case HC_SYSMODALON:
#0102
#0103          // Stop recording while system-modal dialog box is up.
#0104          fPause = TRUE;
#0105          break;
#0106
#0107       case HC_SYSMODALOFF:
#0108
#0109          // The system-modal dialog box is gone, stop recording and notify the
#0110          // user that recording has stopped.
#0111          fPause = FALSE;
#0112          g_RecorderData.RecErr = RECERR_SYSMODAL;
#0113          Recorder_Stop();
#0114          break;
#0115    }
#0116
#0117    return(lResult);
#0118 }
#0119
#0120
#0121 ////////////////////////////////////////////////////////////////
#0122
#0123
#0124 static LRESULT WINAPI Recorder_JournalPlaybackProc (int nCode,
#0125    WPARAM wParam, LPARAM lParam) {
#0126
#0127    PEVENTMSG pEvent;
#0128    LRESULT lResult = CallNextHookEx(g_RecorderData.hhookJournal,
#0129       nCode, wParam, lParam);
#0130
#0131    switch (nCode) {
```

```
#0132
#0133       case HC_SKIP:
#0134
#0135          // Prepare to return the next event the next time the hook code is
#0136          // HC_GETNEXT. If all events have been played, stop playing.
#0137          if (++g_RecorderData.pRecStat->nNumEventsPlayed ==
#0138               g_RecorderData.pRecStat->nNumEvents)
#0139             Recorder_Stop();
#0140          break;
#0141
#0142       case HC_GETNEXT:
#0143
#0144          // Copy current event to the EVENTMSG structure pointed to by lParam.
#0145          pEvent = (PEVENTMSG) &g_RecorderData.pRecStat[1];
#0146          *((PEVENTMSG) lParam) =
#0147             pEvent[g_RecorderData.pRecStat->nNumEventsPlayed];
#0148
#0149          // Adjust 'time' by adding time that playback started.
#0150          ((PEVENTMSG) lParam)->time += g_RecorderData.pRecStat->dwStartTime;
#0151
#0152          // Return the number of milliseconds Windows should wait before
#0153          // processing the event.
#0154          lResult = ((PEVENTMSG) lParam)->time - GetTickCount();
#0155
#0156          // If the event occurred in the past, have Windows process it now.
#0157          if (lResult < 0)
#0158             lResult = 0;
#0159          break;
#0160
#0161       case HC_SYSMODALOFF:
#0162
#0163          // When the system-modal dialog box is removed, stop playing the
#0164          // events and notify the application.
#0165          g_RecorderData.RecErr = RECERR_SYSMODAL;
#0166          Recorder_Stop();
#0167          break;
#0168    }
#0169
#0170    return(lResult);
#0171 }
#0172
#0173
#0174 ////////////////////////////////////////////////////////////////
#0175
#0176
#0177 void WINAPI Recorder_DisplayError (RECERR e) {
```

```
#0178
#0179    LPCTSTR pMessage = NULL;
#0180
#0181    switch (e) {
#0182
#0183       case RECERR_ACTIVE:
#0184          pMessage = __TEXT("Recorder already recording/playing.");
#0185          break;
#0186
#0187       case RECERR_INACTIVE:
#0188          pMessage = __TEXT("Recorder already stopped.");
#0189          break;
#0190
#0191       case RECERR_NOMEMORY:
#0192          pMessage = __TEXT("Insufficient memory.");
#0193          break;
#0194
#0195       case RECERR_NOEVENTS:
#0196          pMessage = __TEXT("No events to playback.");
#0197          break;
#0198
#0199       case RECERR_USERCANCEL:
#0200          pMessage = __TEXT("Recorder canceled by user.");
#0201          break;
#0202
#0203       case RECERR_CANTHOOK:
#0204          pMessage = __TEXT("Unable to set hook.");
#0205          break;
#0206
#0207       case RECERR_SYSMODAL:
#0208          pMessage = __TEXT("Recorder canceled by system modal dialog.");
#0209          break;
#0210
#0211    }
#0212
#0213    if (pMessage != NULL)
#0214       adgMB(pMessage);
#0215 }
#0216
#0217
#0218 ////////////////////////////////////////////////////////////////
#0219
#0220
#0221 RECMODE WINAPI Recorder_GetMode (void) {
#0222
#0223    return(g_RecorderData.RecMode);
```

```
#0224 }
#0225
#0226
#0227 ////////////////////////////////////////////////////////////////
#0228
#0229
#0230 void WINAPI Recorder_Free (HEVENTLIST h) {
#0231
#0232    free((void*) h);
#0233 }
#0234
#0235
#0236 ////////////////////////////////////////////////////////////////
#0237
#0238
#0239 // Returns: RECERR_OK, RECERR_ACTIVE, RECERR_NOMEMORY or RECERR_USERCANCEL
#0240 RECERR WINAPI Recorder_Record (HWND hwndNotify, UINT uMsgNotify) {
#0241
#0242    if ((g_RecorderData.RecMode == RECMODE_RECORDING) ||
#0243        (g_RecorderData.RecMode == RECMODE_PLAYING))
#0244       return(RECERR_ACTIVE);
#0245
#0246    // Allocate and intialize the memory block to hold the statistical data.
#0247    g_RecorderData.pRecStat = (PRECSTAT) malloc(sizeof(RECSTAT));
#0248    if (g_RecorderData.pRecStat == NULL)
#0249       return(RECERR_NOMEMORY);
#0250    g_RecorderData.pRecStat->nNumEvents =
#0251    g_RecorderData.pRecStat->nNumEventsPlayed = 0;
#0252
#0253    // Save information so it can be used by Recorder_Stop.
#0254    g_RecorderData.hwndNotify = hwndNotify;
#0255    g_RecorderData.uMsgNotify = uMsgNotify;
#0256
#0257    // Start record hook, reset error and signal that we are recording.
#0258    // Turn on the event recording.
#0259    g_RecorderData.hhookJournal = SetWindowsHookEx(WH_JOURNALRECORD,
#0260       Recorder_JournalRecordProc, GetModuleHandle(NULL), 0);
#0261    if (g_RecorderData.hhookJournal == NULL) {
#0262       free(g_RecorderData.pRecStat);
#0263       return(RECERR_CANTHOOK);
#0264    }
#0265
#0266    g_RecorderData.RecErr = RECERR_OK;
#0267    g_RecorderData.RecMode = RECMODE_RECORDING;
#0268    return(RECERR_OK);
#0269 }
```

```
#0270
#0271
#0272  ////////////////////////////////////////////////////////////
#0273
#0274
#0275  // Returns: RECERR_OK or RECERR_INACTIVE
#0276  RECERR WINAPI Recorder_Stop (void)  {
#0277
#0278     HEVENTLIST hEventList;
#0279
#0280     switch (g_RecorderData.RecMode) {
#0281
#0282        case RECMODE_STOPPED:
#0283           return(RECERR_INACTIVE);
#0284
#0285        case RECMODE_RECORDING:
#0286           hEventList = (HEVENTLIST) g_RecorderData.pRecStat;
#0287           break;
#0288
#0289        case RECMODE_PLAYING:
#0290           hEventList = NULL;
#0291           break;
#0292     }
#0293
#0294     // Stop playback or recording of events.
#0295     if (g_RecorderData.hhookJournal != NULL) {
#0296
#0297        // If Recorder_Stop is called from Recorder_IsRecorderCanceled (as a
#0298        // result of a WM_CANCELJOURNAL message), our hook has already been
#0299        // unhooked by Windows and the hook handle will be NULL.
#0300        UnhookWindowsHookEx(g_RecorderData.hhookJournal);
#0301        g_RecorderData.hhookJournal = NULL;
#0302     }
#0303
#0304     if (g_RecorderData.RecMode == RECMODE_RECORDING)  {
#0305
#0306        // Recording stopped - modify all 'time' members in the EVENTMSG
#0307        // structs, making each time relative to when recording started.
#0308        PEVENTMSG pEvent = (PEVENTMSG) &g_RecorderData.pRecStat[1];
#0309        int nNumEvents = g_RecorderData.pRecStat->nNumEvents;
#0310        while (nNumEvents >= 1)
#0311           pEvent[--nNumEvents].time -= pEvent[0].time;
#0312     }
#0313     g_RecorderData.pRecStat = NULL;
#0314
#0315     // Signal ourselves and the notification window that we have stopped.
```

```
#0316    g_RecorderData.RecMode = RECMODE_STOPPED;
#0317    SendMessage(g_RecorderData.hwndNotify, g_RecorderData.uMsgNotify,
#0318       (WPARAM) hEventList, (LPARAM) g_RecorderData.RecErr);
#0319
#0320    return(RECERR_OK);
#0321 }
#0322
#0323
#0324 ////////////////////////////////////////////////////////////////
#0325
#0326
#0327 // Returns: RECERR_OK, RECERR_ACTIVE or RECERR_NOEVENTS
#0328 RECERR WINAPI Recorder_Play (HEVENTLIST h, HWND hwndNotify, UINT uMsgNotify) {
#0329
#0330    if ((g_RecorderData.RecMode == RECMODE_RECORDING) ||
#0331        (g_RecorderData.RecMode == RECMODE_PLAYING))
#0332       return(RECERR_ACTIVE);
#0333
#0334    if (((PRECSTAT) h)->nNumEvents == 0)
#0335       return(RECERR_NOEVENTS);
#0336
#0337    // Set new event list.
#0338    g_RecorderData.pRecStat = (PRECSTAT) h;
#0339
#0340    // Save notification information so it can be used by Recorder_Stop.
#0341    g_RecorderData.hwndNotify = hwndNotify;
#0342    g_RecorderData.uMsgNotify = uMsgNotify;
#0343
#0344    // Initialize statistical data and save playback start time
#0345    g_RecorderData.pRecStat->nNumEventsPlayed = 0;
#0346    g_RecorderData.pRecStat->dwStartTime = GetTickCount();
#0347
#0348    // Start playback hook, reset error and signal that we are playing.
#0349    g_RecorderData.hhookJournal = SetWindowsHookEx(WH_JOURNALPLAYBACK,
#0350       Recorder_JournalPlaybackProc, GetModuleHandle(NULL), 0);
#0351
#0352    if (g_RecorderData.hhookJournal == NULL)
#0353       return(RECERR_CANTHOOK);
#0354
#0355    g_RecorderData.RecErr = RECERR_OK;
#0356    g_RecorderData.RecMode = RECMODE_PLAYING;
#0357    return(RECERR_OK);
#0358 }
#0359
#0360
#0361 ////////////////////////////////////////////////////////////////
```

```
#0362
#0363
#0364  BOOL WINAPI Recorder_IsRecorderCanceled (PMSG pmsg) {
#0365
#0366     BOOL fCanceled = (pmsg->message == WM_CANCELJOURNAL);
#0367     if (fCanceled) {
#0368
#0369        // The system unhooked our hook.
#0370        g_RecorderData.hhookJournal = NULL;
#0371
#0372        // Stop recording.
#0373        g_RecorderData.RecErr = RECERR_USERCANCEL;
#0374        Recorder_Stop();
#0375     }
#0376     return(fCanceled);
#0377  }
#0378
#0379
#0380  /////////////////////////// End of File ///////////////////////////
```

程式列表 6.17　Record.H

```
#0001  /******************************************************************
#0002  Module name: Record.h
#0003  Written by: Jeffrey Richter and Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Macro recorder programmer's interface.
#0006  ******************************************************************/
#0007
#0008
#0009  typedef enum {
#0010     RECERR_OK,       // Operation was successful.
#0011     RECERR_CANTHOOK,  // Hook cannot be installed.
#0012     RECERR_ACTIVE,    // Attempt to record/play while already recording/playing.
#0013     RECERR_INACTIVE,  // Attempt to stop recording while NOT recording.
#0014     RECERR_NOMEMORY,  // When attempting to start recording or during recording.
#0015     RECERR_NOEVENTS,  // Attempt playback with no events in memory block.
#0016     RECERR_SYSMODAL,  // System modal dialog canceled recording.
#0017     RECERR_USERCANCEL // User canceled with Ctrl+Esc.
#0018  } RECERR;
#0019
#0020
#0021  ////////////////////////////////////////////////////////////////////
#0022
#0023
```

```
#0024  typedef enum {
#0025     RECMODE_STOPPED,         // Recorder is stopped
#0026     RECMODE_RECORDING,       // Recorder is recording events
#0027     RECMODE_PLAYING          // Recorder is playing back events
#0028  } RECMODE;
#0029
#0030
#0031  ////////////////////////////////////////////////////////////////
#0032
#0033
#0034  DECLARE_HANDLE(HEVENTLIST);
#0035
#0036
#0037  ////////////////////////////////////////////////////////////////
#0038
#0039
#0040  void WINAPI Recorder_Init ();
#0041  void WINAPI Recorder_Free (HEVENTLIST h);
#0042  void WINAPI Recorder_DisplayError (RECERR e);
#0043  RECMODE WINAPI Recorder_GetMode (void);
#0044  BOOL WINAPI Recorder_IsRecorderCanceled (PMSG pmsg);
#0045
#0046  // Returns: RECERR_OK, RECERR_CANTHOOK, RECERR_ACTIVE or RECERR_NOMEMORY
#0047  RECERR WINAPI Recorder_Record (HWND hwndNotify, UINT uMsgNotify);
#0048
#0049  // Returns: RECERR_OK, RECERR_CANTHOOK, RECERR_ACTIVE or RECERR_NOEVENTS
#0050  RECERR WINAPI Recorder_Play (HEVENTLIST h, HWND hwndNotify, UINT uMsgNotify);
#0051
#0052  // Returns: RECERR_OK or RECERR_INACTIVE
#0053  RECERR WINAPI Recorder_Stop (void);
#0054
#0055
#0056  ////////////////////////// End of File ///////////////////////////
```

程式列表 6.18　Echo.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
```

```
#0010  #include "windows.h"
#0011
#0012  /////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  /////////////////////////////////////////////////////////////////
#0017  //
#0018  // Dialog
#0019  //
#0020
#0021  IDD_ECHO DIALOG DISCARDABLE  -32768, 5, 180, 24
#0022  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0023  CAPTION "Echo"
#0024  FONT 8, "MS Sans Serif"
#0025  BEGIN
#0026      PUSHBUTTON     "&Record",IDC_RECORD,8,5,49,14
#0027      PUSHBUTTON     "&Play",IDC_PLAY,65,5,49,14
#0028      PUSHBUTTON     "&Stop",IDC_STOP,122,5,49,14
#0029  END
#0030
#0031
#0032  /////////////////////////////////////////////////////////////////
#0033  //
#0034  // Icon
#0035  //
#0036
#0037  IDI_ECHO              ICON    DISCARDABLE     "echo.ico"
#0038
#0039  #ifdef APSTUDIO_INVOKED
#0040  /////////////////////////////////////////////////////////////////
#0041  //
#0042  // TEXTINCLUDE
#0043  //
#0044
#0045  1 TEXTINCLUDE DISCARDABLE
#0046  BEGIN
#0047      "resource.h\0"
#0048  END
#0049
#0050  2 TEXTINCLUDE DISCARDABLE
#0051  BEGIN
#0052      "#include ""windows.h""\r\n"
#0053      "\0"
#0054  END
#0055
```

```
#0056  3 TEXTINCLUDE DISCARDABLE
#0057  BEGIN
#0058      "\r\n"
#0059      "\0"
#0060  END
#0061
#0062  ////////////////////////////////////////////////////////////////
#0063  #endif    // APSTUDIO_INVOKED
#0064
#0065
#0066  #ifndef APSTUDIO_INVOKED
#0067  ////////////////////////////////////////////////////////////////
#0068  //
#0069  // Generated from the TEXTINCLUDE 3 resource.
#0070  //
#0071
#0072
#0073  ////////////////////////////////////////////////////////////////
#0074  #endif    // not APSTUDIO_INVOKED
```

程式列表 6.19　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by echo.rc
#0004  //
#0005  #define IDD_ECHO                    101
#0006  #define IDI_ECHO                    102
#0007  #define IDC_RECORD                  1000
#0008  #define IDC_PLAY                    1001
#0009  #define IDC_STOP                    1002
#0010
#0011  // Next default values for new objects
#0012  //
#0013  #ifdef APSTUDIO_INVOKED
#0014  #ifndef APSTUDIO_READONLY_SYMBOLS
#0015  #define _APS_NEXT_RESOURCE_VALUE       103
#0016  #define _APS_NEXT_COMMAND_VALUE        40001
#0017  #define _APS_NEXT_CONTROL_VALUE        1001
#0018  #define _APS_NEXT_SYMED_VALUE          101
#0019  #endif
#0020  #endif
```

Capture 程式

Capture 程式(Capture.EXE)的原始碼顯示於**程式 列表 6.20 ~ 6.22** 中，示範 journal record hook 對於滑鼠捕捉權（mouse capture）的影響。當你執行這個程式，主視窗如**圖 6.6** 所示。

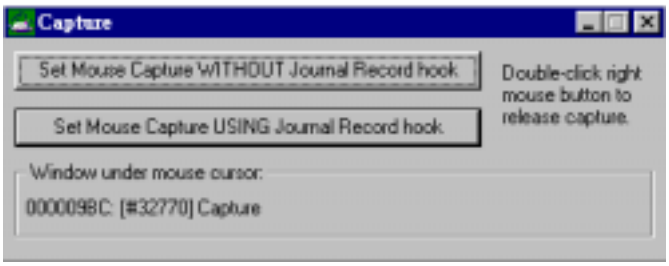


圖 6.6 Capture 程式

Capture 程式的視窗中有兩個按鈕：【Set Mouse Capture WITHOUT Journal Record Hook】和【Set Mouse Capture USING Journal Record Hook】。當你按下其中一個鈕，Capture 程式會獲得滑鼠的專屬捕捉權，而後在每一個 WM_MOUSEMOVE 訊息進來時，它會更新顯示在【Window Under Mouse Cursor】框框內所顯示的視窗代碼、視窗類別和視窗文字。在對話盒的視窗範圍內快按兩下（Double-clicking）滑鼠右鍵，會使得 Capture 程式取消滑鼠捕捉權。

建立此程式所需用到的檔案列於**表 6.7**。

表 6.7 建立 Capture 程式所需的檔案

檔案	說明
Capture.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Capture.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 Capture.RC 檔中所有資源的 ID。

Capture.ICO	主視窗圖示（icon）。
Capture.MAK	Visual C++ 的 MAK 檔。

基本上，Capture 程式主要並不是用來說明如何使用 journal record hook，而是用來討論使用 journal record hooks 所造成的副作用。事實上 Capture 程式的 filter function，也就是 Capture_JournalRecordProc，除了利用 CallNextHookEx 函式呼叫下一個 filter function 之外，並沒有做其他任何事情。

Local Input States

爲了使 32 位元作業系統 Windows 95 和 Windows NT 比 16 位元 Windows 更強固穩定，微軟公司在 32 位元系統上增加了所謂的 local input state processing。簡單地說，local input state processing 意謂著每一個執行緒（不是行程，而是執行緒）認爲它是系統之中唯一能夠取得到鍵盤和滑鼠訊息者。Local input state 模式很理想，因爲執行緒之間各自獨立，這可以避免某個執行緒影響到其他執行緒。

舉個例子，想像一下這樣的故事發生在一個 16 位元 Windows 環境中。一個 task 呼叫 SetCapture 函式，告訴系統說它要捕捉所有的滑鼠訊息，送到此 task 所產生的視窗上。無論使用者將滑鼠移到何處，所有的滑鼠訊息皆會流往該視窗。現在，讓我們假設這個 task 有一隻臭蟲，當使用者按下滑鼠按鈕，程式便進入一個無窮迴路。在 16 位元 Windows 環境中，這個無窮迴路意謂著滑鼠訊息捕捉權不會被取消，而使用者再也不能使用滑鼠了。唔，一個強固的作業系統不應該允許這種情況發生。

Local input state processing 可以避免一個執行緒產生這種情況。對於滑鼠捕捉權而言，Windows 95 和 Windows NT 的工作都是這樣：當使用者按下滑鼠按鈕，執行緒呼叫 SetCapture 函式，於是系統以一種 “system-wide” basis 形式完成滑鼠的捕捉，也就是說和在 16 位元 Windows 環境中的情況沒有兩樣。然而，當滑鼠按鍵放開（即使 ReleaseCapture 函式未被呼叫），系統會改變滑鼠捕捉權，使其範圍侷限爲 thread-local basis。換言之，如果你希望滑鼠訊息流往視窗 A，那麼只有當滑鼠游標落在某個視窗，而此視窗是由產

生視窗 A 之執行緒所產生，才有可能。如果滑鼠游標落在其它執行緒所產生的視窗上，作業系統會如平常一樣地將訊息傳給它 -- 即使該視窗沒有設定滑鼠捕捉權。

對大部份程式而言，Windows 95 和 Windows NT 上的這項改變不會構成問題。更具體地說，任何程式如果不去干擾由其它執行緒所產生出來的視窗，那麼它移植爲 Win32 程式時應該不會有任何問題。至於那些利用 "send"、"post"、 或是攔截鍵盤 events 或滑鼠 events 以操控其它視窗之應用程式，可能會遇到一些移植上的問題。

讓我們看一個例子。微軟的 Spy 工具（自從 Visual C++ 1.5 起便問世），就是這樣一個程式。16 位元的 Spy.EXE 允許使用者在【Window / Window…】命令選項中選取一個目標視窗。這個選項會引起一個對話盒，然後 Spy 呼叫 SetCapture，將所有的滑鼠訊息導引到這個對話盒之中。使用者現在可以在螢幕上任意移動滑鼠至想要窺視的視窗上，Spy 會在對話盒中顯現該視窗的相關資訊。當使用者按下滑鼠左鍵，Spy 對話盒會收到 WM_LBUTTONDOWN 訊息（不管滑鼠目前在哪裡），於是開始監視所選取之視窗的所有視窗訊息。

如果你用的是 Win32 Spy++ 程式（隨著 Visual C++ 2.x 版推出），你會發現它使用了一個與 16 位元 Spy 工具截然不同的技術來讓使用者選取視窗。首先，你必須選擇【Spy / Find Window…】命令選項，於是出現一個如圖 **6.7** 所示的對話盒。當這個對話盒啓動時，Spy++ 並不去呼叫 SetCapture 函式。

圖 6.7 Spy++的 Find Window 對話盒

取而代之的是，這個對話盒上有一個工具叫做【Finder Tool】，使用者必須移動滑鼠至【Finder Tool】的位置上，按下滑鼠左鍵，並拖拉此【Finder Tool】工具至想要監看的視窗上。當使用者在【Finder Tool】位置上點選一下滑鼠，Spy++ 便呼叫 SetCapture。滑鼠按鍵被按下時，滑鼠捕捉權便以 system-wide 的方式執行了。這表示使用者現在可以移動滑鼠至螢幕上的任何地方，所有的滑鼠訊息都會被傳給 SetCapture 函式所指定的視窗。然而當使用者放開滑鼠按鍵，滑鼠捕捉權是以 thread-local basis 方式執行，這表示只有當滑鼠落於同一個執行緒所產生之視窗上時，系統才會將滑鼠訊息傳給 SetCapture 函式所指定的視窗。如果滑鼠移動到由另一個執行緒所產生的視窗上，系統會如一般情況（沒有呼叫 SetCapture 函式）一樣地將滑鼠訊息交給其所坐落的視窗。當滑鼠按鍵放開，Spy++ 會呼叫 ReleaseCapture 函式，完全取消滑鼠捕捉權，而不是以 thread-local basis 的形式存在。

由於這種新的 local input state processing，Spy++ 被強迫使用一種與 16 位元 Spy 不同的技術來完成監看工作。

Journal Record Hooks 和 Local Input State Processing

另有一種方法可以解決 Spy++ 的問題，那就是把作業系統的 local input state processing 能力暫時關閉，作法則是安裝一個 journal record hook。然而一旦關閉了 local input state processing，Win32 環境的行爲就如同 16 位元 Windows 一樣，換句話說此刻執行緒有可能被其它執行緒所影響，也有可能影響所有執行中的行程。

在 Windows 95 和 Windows NT 環境下，local input state processing 會與 journal hooks 相互干擾。因此，爲了維護回溯相容性，不管安裝哪一種 journal hooks，系統都會關閉 local input state processing。一旦卸除 journal record / playback hook，系統會再恢復正常的 local input state processing。然而只要 local input state processing 關閉，所有執行緒皆能取得任何視窗上的任何滑鼠訊息。這表示當掛上 journal hook，滑鼠訊息捕捉權是以 system-wide 的方式來執行。

Capture 程式展示了這項技術。當你按下【Set Mouse Capture WITHOUT Journal hook】鈕，程式會呼叫 SetCapture 函式，並傳入對話盒的視窗代碼。然後在每個 WM_MOUSEMOVE 訊息來臨時，Capture 程式便會去更新【Window Under Mouse Cursor】框框中的資訊。注意，當你將滑鼠移到一個「並非由產生出對話盒的那個執行緒所產生出來的視窗」上時，框框中的資訊並不會改變，這是因爲此時 local input state processing 還有效用。如果你按下的是【Set Mouse Capture USING Journal Record Hook】鈕，請你注意，當你移動滑鼠，框框中的資訊會隨之變化。更重要的是，無論滑鼠游標位於螢幕何處，框框中的資訊皆會隨之改變。這是因爲 Capture 程式掛了一個 journal hook，導至滑鼠捕捉權是以 system-wide 的方式執行。

Capture.ICO

程式列表 6.20　Capture.C

```
#0001  /******************************************************************
#0002  Module name: Capture.c
#0003  Written by: Jeffrey Richter and Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates how to disable the local input state processing.
#0006  ******************************************************************/
#0007
```

```
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)   /* Single line comment */
#0013  #include <tchar.h>
#0014  #include <stdio.h>
#0015  #include "Resource.h"
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  // Global hook handle for journal record hook (NULL when the not installed).
#0021  static HHOOK g_hhookJournalRecord = NULL;
#0022
#0023
#0024  ////////////////////////////////////////////////////////////////
#0025
#0026
#0027  void Capture_CalcWndText (HWND hwnd, LPTSTR szBuf, int nLen) {
#0028
#0029     TCHAR szClass[256], szCaption[256], szBufT[256];
#0030
#0031     if (hwnd == NULL) {
#0032        _tcscpy(szBuf, __TEXT("(no window)"));
#0033        return;
#0034     }
#0035
#0036     if (!IsWindow(hwnd)) {
#0037        _tcscpy(szBuf, __TEXT("(invalid window)"));
#0038        return;
#0039     }
#0040
#0041     GetClassName(hwnd, szClass, adgARRAY_SIZE(szClass));
#0042     GetWindowText(hwnd, szCaption, adgARRAY_SIZE(szCaption));
#0043
#0044     _stprintf(szBufT, __TEXT("%08X: [%s] %s"), hwnd, szClass,
#0045        (*szCaption == 0) ? __TEXT("(no caption)") : szCaption);
#0046     _tcsncpy(szBuf, szBufT, nLen - 1);
#0047     szBuf[nLen - 1] = 0;          // Terminate string
#0048  }
#0049
#0050
#0051  ////////////////////////////////////////////////////////////////
#0052
#0053
```

```
#0054
#0055  static LRESULT WINAPI Capture_JournalRecordProc (int nCode,
#0056    WPARAM wParam, LPARAM lParam) {
#0057
#0058    // This journal record function doesn't need to do anything so we just pass
#0059    // the hook notification on to any other installed journal record hooks.
#0060    return(CallNextHookEx(g_hhookJournalRecord, nCode, wParam, lParam));
#0061  }
#0062
#0063
#0064  ////////////////////////////////////////////////////////////////
#0065
#0066
#0067  BOOL Capture_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0068
#0069    adgSETDLGICONS(hwnd, IDI_CAPTURE, IDI_CAPTURE);
#0070    return(TRUE);               // Accept default focus window.
#0071  }
#0072
#0073
#0074  ////////////////////////////////////////////////////////////////
#0075
#0076
#0077  void Capture_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0078
#0079    switch (id) {
#0080
#0081       case IDCANCEL:
#0082
#0083          // Terminate the application when the user selects
#0084          // Close from the system menu.
#0085          PostQuitMessage(0);
#0086          break;
#0087
#0088       case IDC_CAPTUREWITHOUTJRHOOK:
#0089
#0090          // Set capture and change the mouse cursor shape. Note that the mouse
#0091          // cursor shape is an up arrow ONLY when the cursor is over the
#0092          // dialog box. Mouse cursor shape is also part of the local input
#0093          // state processing.
#0094          SetCapture(hwnd);
#0095          SetCursor(LoadCursor(NULL, IDC_UPARROW));
#0096          break;
#0097
#0098       case IDC_CAPTUREUSINGJRHOOK:
#0099
```

```
#0100         // Install a journal hook to turn off local input state processing,
#0101         // set capture and change the mouse cursor shape. Note that the mouse
#0102         // cursor shape is always an up arrow regardless of where the mouse
#0103         // cursor is on the screen.
#0104         g_hhookJournalRecord = SetWindowsHookEx(WH_JOURNALRECORD,
#0105            Capture_JournalRecordProc, GetModuleHandle(NULL), 0);
#0106         adgASSERT(g_hhookJournalRecord);
#0107         SetCapture(hwnd);
#0108         SetCursor(LoadCursor(NULL, IDC_UPARROW));
#0109         break;
#0110    }
#0111 }
#0112
#0113
#0114 ////////////////////////////////////////////////////////////////
#0115
#0116
#0117 int Capture_OnRButtonDown (HWND hwnd, BOOL fDoubleClick,
#0118    int x, int y, UINT keyFlags) {
#0119
#0120    if (fDoubleClick) {
#0121
#0122       // The user double-clicked the right mouse button.
#0123       ReleaseCapture();
#0124       if (g_hhookJournalRecord != NULL) {
#0125
#0126          // If a journal hook is installed, we must turn local input state
#0127          // processing back on.
#0128          UnhookWindowsHookEx(g_hhookJournalRecord);
#0129          g_hhookJournalRecord = NULL;
#0130       }
#0131    }
#0132    return(0);
#0133 }
#0134
#0135
#0136 ////////////////////////////////////////////////////////////////
#0137
#0138
#0139 void Capture_OnMouseMove (HWND hwnd, int x, int y, UINT keyFlags) {
#0140
#0141    TCHAR szBuf[128];
#0142    POINT pt;
#0143
#0144    // As the mouse moves, update the text in the box.
#0145
```

```
#0146    // Get the position of the mouse cursor in screen coordinates. We can't use
#0147    // the x & y parameters because they are in client-window coordinates.
#0148    GetCursorPos(&pt);
#0149
#0150    Capture_CalcWndText(WindowFromPoint(pt), szBuf, adgARRAY_SIZE(szBuf));
#0151    SetDlgItemText(hwnd, IDC_WNDUNDERMOUSE, szBuf);
#0152  }
#0153
#0154
#0155  ////////////////////////////////////////////////////////////////
#0156
#0157
#0158  BOOL WINAPI Capture_DlgProc (HWND hwnd, UINT uMsg,
#0159    WPARAM wParam, LPARAM lParam) {
#0160
#0161    switch (uMsg) {
#0162
#0163       // Standard Windows messages
#0164       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG,    Capture_OnInitDialog);
#0165       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,      Capture_OnCommand);
#0166       adgHANDLE_DLGMSG(hwnd, WM_MOUSEMOVE,     Capture_OnMouseMove);
#0167       adgHANDLE_DLGMSG(hwnd, WM_RBUTTONDBLCLK, Capture_OnRButtonDown);
#0168    }
#0169    return(FALSE);
#0170  }
#0171
#0172
#0173  ////////////////////////////////////////////////////////////////
#0174
#0175
#0176  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0177    LPSTR lpszCmdLine, int nCmdShow) {
#0178
#0179    MSG msg;
#0180    HWND hwnd;
#0181
#0182    adgWARNIFUNICODEUNDERWIN95();
#0183
#0184    // Create a modeless dialog box instead of a modal dialog box because we
#0185    // need to have more control over the message loop processing.
#0186    hwnd = CreateDialog(hinstExe, MAKEINTRESOURCE(IDD_CAPTURE), NULL,
#0187       Capture_DlgProc);
#0188    adgASSERT(IsWindow(hwnd));
#0189
#0190    // Continue to loop until a WM_QUIT message comes out of the queue.
#0191    while (GetMessage(&msg, NULL, 0, 0)) {
```

```
#0192
#0193       if (msg.message == WM_CANCELJOURNAL) {
#0194
#0195          // A user can force local input state processing back on by pressing
#0196          // Ctrl+Esc. When this happens, the operating system notifies the
#0197          // thread by posting a WM_CANCELJOURNAL message in the thread's
#0198          // message queue. We can look for this here and notify the user that
#0199          // local input state processing is back on.
#0200
#0201          adgMB(__TEXT("System-wide mouse capture has been canceled"));
#0202       } else {
#0203
#0204          // Call IsDialogMessage so that the keyboard can be used to control
#0205          // focus in the dialog box.
#0206          if (!IsDialogMessage(hwnd, &msg)) {
#0207             TranslateMessage(&msg);
#0208             DispatchMessage(&msg);
#0209          }
#0210       }
#0211    }
#0212
#0213    // The application is terminating; destroy the modeless dialog box.
#0214    DestroyWindow(hwnd);
#0215    return(0);
#0216 }
#0217
#0218
#0219 ////////////////////// End of File ///////////////////////////
```

程式列表 6.21　Capture.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "Resource.h"
#0004
#0005 #define APSTUDIO_READONLY_SYMBOLS
#0006 ////////////////////////////////////////////////////////////////
#0007 //
#0008 // Generated from the TEXTINCLUDE 2 resource.
#0009 //
#0010 #include "windows.h"
#0011
#0012 ////////////////////////////////////////////////////////////////
#0013 #undef APSTUDIO_READONLY_SYMBOLS
#0014
```

```
#0015
#0016 //////////////////////////////////////////////////////////////
#0017 //
#0018 // Dialog
#0019 //
#0020
#0021 IDD_CAPTURE DIALOG DISCARDABLE  -32768, 5, 260, 80
#0022 STYLE WS_MINIMIZEBOX | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0023 CAPTION "Capture"
#0024 FONT 8, "MS Sans Serif"
#0025 BEGIN
#0026     DEFPUSHBUTTON   "Set Mouse Capture WITHOUT Journal Record hook",
#0027                     IDC_CAPTUREWITHOUTJRHOOK,4,4,184,14
#0028     DEFPUSHBUTTON   "Set Mouse Capture USING Journal Record hook",
#0029                     IDC_CAPTUREUSINGJRHOOK,4,25,184,14
#0030     LTEXT          "Double-click right mouse button to release capture.",-1,
#0031                     196,8,60,27
#0032     GROUPBOX        "Window under mouse cursor:",-1,4,44,252,28
#0033     LTEXT          "HWND: [class] caption",IDC_WNDUNDERMOUSE,8,57,240,8
#0034 END
#0035
#0036
#0037 #ifdef APSTUDIO_INVOKED
#0038 //////////////////////////////////////////////////////////////
#0039 //
#0040 // TEXTINCLUDE
#0041 //
#0042
#0043 1 TEXTINCLUDE DISCARDABLE
#0044 BEGIN
#0045     "Resource.h\0"
#0046 END
#0047
#0048 2 TEXTINCLUDE DISCARDABLE
#0049 BEGIN
#0050     "#include ""windows.h""\r\n"
#0051     "\0"
#0052 END
#0053
#0054 3 TEXTINCLUDE DISCARDABLE
#0055 BEGIN
#0056     "\r\n"
#0057     "\0"
#0058 END
#0059
#0060 //////////////////////////////////////////////////////////////
```

```
#0061  #endif    // APSTUDIO_INVOKED
#0062
#0063
#0064  /////////////////////////////////////////////////////////////////////
#0065  //
#0066  // Icon
#0067  //
#0068
#0069  IDI_CAPTURE           ICON    DISCARDABLE    "Capture.ico"
#0070
#0071  #ifndef APSTUDIO_INVOKED
#0072  /////////////////////////////////////////////////////////////////////
#0073  //
#0074  // Generated from the TEXTINCLUDE 3 resource.
#0075  //
#0076
#0077
#0078  /////////////////////////////////////////////////////////////////////
#0079  #endif    // not APSTUDIO_INVOKED
```

程式列表 6.22　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by Capture.rc
#0004  //
#0005  #define IDD_CAPTURE                  102
#0006  #define IDC_WNDUNDERMOUSE             104
#0007  #define IDI_CAPTURE                  104
#0008  #define IDC_CAPTUREWITHOUTJRHOOK       1001
#0009  #define IDC_CAPTUREUSINGJRHOOK         1002
#0010
#0011  // Next default values for new objects
#0012  //
#0013  #ifdef APSTUDIO_INVOKED
#0014  #ifndef APSTUDIO_READONLY_SYMBOLS
#0015  #define _APS_NEXT_RESOURCE_VALUE       105
#0016  #define _APS_NEXT_COMMAND_VALUE        40001
#0017  #define _APS_NEXT_CONTROL_VALUE        1003
#0018  #define _APS_NEXT_SYMED_VALUE          101
#0019  #endif
#0020  #endif
```

第７章

檔案的拖放（Drag-and-Drop）技術

在每一個新版本的 Windows 上頭，微軟公司都會加上一些新功能，朝向所謂「資訊彈指可得（Information at your finger）」的目標再邁一步。這個目標讓使用者能更專注於他們的工作上頭，而不必花很多的時間在學習與瞭解他們所使用的工具。OLE（Objecte Linking and Embedding）是可以幫助我們達成這個目標的技術之一。這項技術可以讓使用者選擇一個 “host” 程式，這個程式主要是用來顯現資料（例如文書編輯器、試算表或簡報軟體）。一旦 host 程式決定好了，OLE 就允許使用者將不同形式的資料插入到 host 程式的文件之中。舉個例，在一份由文書處理器所產生的文件中，使用者可以輕易插入一份由試算表軟體所產生出來的圖表。當使用者想要編修圖表，程式會自動呼叫試算表軟體來處理，然後使用者就可以動作了。完成的結果立刻取代原來的內容。這項技術幫助使用者以一個最熟悉的工具，藉著 OLE 的協助，輕鬆完成不同媒體資料的整合工作。

我們希望使用者所關心的是他所要處理的資料，而不是用來處理那些資料的程式。除了 OLE 之外，Windows 還提供了其它功能，幫助使用者朝這個目標邁進，這項技術就是本章的題目：drag-and-drop。或許最容易體會其功能的方法就是，使用一下 Windows 95 的檔案總管（Explorer）。過去，終端用戶很難輕鬆瞭解 MS-DOS 的檔案系統和目錄結構與階層關係，檔案總管的設計讓使用者可以比較輕鬆而有效率地操控檔案。

無疑地，檔案總管之所以能夠如此容易使用，它的 drag-and-drop 能力是一大關鍵。Drag-and-drop 能夠簡化許多一般性的工作。首先我們來看看它是如何在磁碟目錄間做檔

案複製和搬移的工作。先以滑鼠選取一個或多個檔案，然後按住滑鼠按鍵不放，移動滑鼠；當你放開滑鼠按鍵，被選擇的那些檔案就會搬移或複製到新的位置。如果滑鼠所落地點的磁碟機與原本檔案所在的磁碟機不同，就會做複製動作。反之如果屬於同一磁碟機，就會做搬移動作。

你也可以從檔案總管中拖拉一份文件檔，將它放置於欲開啓此檔案的程式圖示上，於是開啓一份文件。舉個例，如果你將 Flock.BMP 檔拖拉至 PBrush.EXE 程式圖示上，檔案總管會建構出如下的命令列，並執行之：

```
C:\WINDOWS\PBRUSH.EXE C:\WINDOWS\FLOCK.BMP
```

到目前爲止，我們都一直在檔案總管內打轉，其實我們也可以將檔案總管內的檔案拖出檔案總管視窗之外，置於其它程式的視窗中。當你將檔案丟進某一個程式時，該程式會企圖將丟進來的檔案打開來。舉個例，如果你將 Setup.INF 檔拖拉至記事本（Notepad.EXE），記事本會開啓這個檔案，並將自己的視窗標題欄從 "Untitled-記事本" 改爲 "SETUP.INF - 記事本"，以此顯示它目前正在編輯的檔案。

你一定可以察覺出這樣的開檔方式對使用者來說是多麼的方便與容易！使用者於是可以致力於他所要處理的資料上，而不需分心於工具的使用。類似的道理，如果桌面上有一個印表機圖示（icon），使用者將檔案拖放到其圖示上，這個檔案就應該被印出，而不是被編輯。如果拖放的目的地是一個拼字檢查軟體，它就應該自動幫你檢查檔案之中是否有拼字錯誤。

使用者總是希望一個程式能同時具備多樣功能，用以處理不同的工作。他們不希望在載入一種文字編輯器（例如筆記本）後，還需再使用其他工具來編輯各種不同的檔案。

成爲一個拖放標的（Drag and Drop Target）

修改你的程式，使它成爲一個拖放標的，是非常簡單的一件事。首先你必須告訴其他程式（例如檔案總管）說，我可以接受丟進來的檔案。只要指定一個擴充的視窗風格

WS_EX_ACCEPTFILES，即可辦到這一點。要加上這個擴充的視窗風格，你必須使用 CreateWindowEx 函式來產生視窗，而不能使用 CreateWindow 函式：

```
hwnd = CreateWindowEx(WS_EX_ACCEPTFILES, "ClassName", "Caption",
   WS_OVERLAPPED, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
   CW_USEDEFAULT, NULL, NULL, hInstance, 0);
```

如果視窗已被產生出來，你也可以利用 DragAcceptFiles 函式來完成這個目標;：

```
void DragAcceptFiles(HWND hwnd, BOOL fAccept);
```

這個函式只是把 WS_EX_ACCEPTFILES 風格位元設爲 on 或 off（視 fAccept 參數而定）。如果 fAccept 參數爲 TRUE，則將該風格位元設爲 on。透過這個函式，我們就可以在程式執行期間，視情況隨時改變這個視窗的風格。當使用者拖拉一個檔案時，檔案總管會檢查滑鼠游標下的視窗，看其 WS_EX_ACCEPTFILES 風格位元爲何？如果該視窗沒有設定 WS_EX_ACCEPTFILES 位元爲 on，滑鼠游標就會如圖 7.1 般地顯示。這個游標式樣告訴使用者說，你不可以把檔案丟在目前的地點。反過來說，如果 WS_EX_ACCEPTFILES 位元是 on，則游標式樣如圖 7.2 所示。

圖 7.1　游標式樣，表示此地不允許將檔案丟入。

圖 7.2　游標式樣，表示此地可接受檔案丟入。

檔案總管也允許使用者一次選取並拖曳數個檔案。舉個例子，這項性質可讓你一次選取數個原始碼檔案，並將它們一起拖拉至 Visual C++ 的視窗中，即可全部打開。Visual C++ 知道如何一次處理數個丟進來的檔案，並將它們打開，放到編輯視窗中。

有些時候，我們也必須將執行中的程式視窗的 WS_EX_ACCEPTFILES 位元設爲 off。譬

如說，當一個程式的執行緒正在列印一個檔案時，它必須呼叫 DragAcceptFiles 函式，設定 fAccept 參數爲 FALSE，使在列印期間不接受任何被拖曳過來的檔案。一旦文件列印完成，你就可以再呼叫 DragAcceptFiles 函式，設定 fAccept 參數爲 TRUE，使它得以繼續接受檔案的丟入。

拖放（Drag and Drop）如何運作

當你使用滑鼠拖曳一個或數個檔案，然後放掉滑鼠按鍵，檔案總管會配置一塊記憶體並填入你所拖曳的所有檔案名稱。事實上，這個名稱也包含了檔案的完整路徑，而不僅僅只是檔案名稱而已。接下來檔案總管會 "post" 一個 WM_DROPFILES 訊息，交給「你放開滑鼠按鍵時，滑鼠坐落的視窗」，其 wParam 參數爲上述那塊記憶體（記錄著所有被選取檔案的名稱）的代碼（handle）[1]，lParam 參數未被使用。

一旦視窗收到這個訊息，程式應該呼叫 DragQueryFile 函式來處理那個記憶體區塊：

```
UNIT DragQueryFile(HDROP hdrop, UNIT uFileNum, LPSTR lpszFile,
   UNIT uMaxFileSize);
```

hdrop 參數就是記憶體區塊代碼，其中記錄著所有丟入此視窗中的檔案的名稱，包含路徑與檔名（這個代碼也就是 WM_DROPFILES 訊息的 wParam 參數）。uFileNum 參數表示要從記憶體區塊中取得哪一個檔案名稱，其範圍可在 0 ~ (丟進來的檔案個數-1) 之間。如果將 uFileNum 參數設爲 -1 的話，DragQueryFile 函式會傳回記憶體區塊中的檔案個數。

[1] 雖然 Win32 API 中仍存在有 GlobalAlloc 和 LocalAlloc 函式，但你應該盡量避免使用它們，你應該使用 HeapAlloc 函式。不幸的是某些 16 位元 Windows 的舊觀念（尤其是記憶體配置）牢牢牽制著許多程式員。所以微軟決定暫時留下一些舊有的 16 位元函式。我們在 drag and drop 這個題目上就看到了這種情況。「檔案總管」內部就是使用 GlobalAlloc 函式來配置記憶體，而不是使用 HeapAlloc。我們從 WM_DROPFILES 訊息的 wParam 參數中得到這塊記憶體的代碼（handle）。

當 DragQueryFile 函式被用來取得單一路徑名稱時，lpszFile 參數表示一個緩衝區位址，路徑名稱會被 DragQueryFile 函式拷貝到這個位址上。uMaxFileSize 參數表示這個緩衝區所能接受的最大字元個數。拷貝動作完成之後，DragQueryFile 會傳回實際拷貝到緩衝區中的字串長度。

如果你將 lpszFile 參數設爲 NULL，DragQueryFile 函式會傳回路徑名稱的字串長度。

下列程式片段取自一個拖放目標視窗的視窗函式中，示範如何將拖放進來的檔案名稱加到 listbox 控制元件中：

```
void SomeCls_OnDropFiles (HWND hwnd, HDROP hdrop) {

   // The HWND of the listbox is in g_hwndLB

   // Get the number of pathnames that have been dropped.
   UNIT nNumFiles = DragQueryFile(hdrop, -1, NULL, 0);
   UNIT uNumChars;
   LPCTSTR pszPathname;

   // Add each pathname to the listbox.
   while (nNumFiles--) {

      // Get the number of characters required by the file's pathname.
      nNumChars = DragQueryFile(hdrop, nNumFiles, NULL, 0);

      // Allocate memory to certain the pathname.
      pszPathname = malloc(sizeof(TCHAR) * (uNumChars + 1));

      // If not enough memory, skip this one.
      if (pszPathname != NULL) {
         // Copy the pathname into the buffer and add to the listbox.
       DragQueryFile(hdrop, uNumFiles, pszPathname, uNumChars);
         ListBox_AddString(g_hwndLB, pszPathname);
      }
   }

   // Free the memory containing the dropped-file information.
   DragFinish(hdrop);
}
```

在處理完 WM_DRAPFILES 訊息之後，我們可以使用 DragFinish 函式釋放記錄著路徑與檔案名稱的記憶體區塊。如果你沒有這麼做，此塊記憶體就必須等到你的行程結束，才會由 Windows 幫你釋放。

記憶體區塊中除了記錄著檔案名稱及路徑，檔案總管同時也爲我們加入了其它一些資訊，給拖放目標程式使用。你可以呼叫 DragQueryPoint 函式得到那些資訊：

```
BOOL DragQueryPoint(HDROP hdrop, LPPOINT lpPoint)
```

和先前一樣，hdrop 參數是一個記憶體代碼，lpPoint 參數則是一個指向 POINT 結構的位址。當檔案被丟入，DragQueryPoint 函式會將「丟入當時」的滑鼠座標拷貝至 lpPoint 所指向的地方。POINT 結構中的 x 和 y 值是相對於「收到 WM_DROPFILES 訊息的視窗」的工作區座標。如果滑鼠游標落在視窗工作區中，DragQueryPoint 函式會傳回 TRUE。如果落在視窗工作區之外，則傳回 FALSE。當然，DragQueryPoint 函式必須在 DragFinish 函式被呼叫之前呼叫之。

一個程式可以利用 DragQueryPoint 函式的傳回值來判斷如何處理這些丟進來的檔案。舉個例子，如果檔案被丟進 WordPad 視窗的非工作區域中（例如丟到標題欄上），WordPad 程式會開啓這個檔案；如果被丟到其工作區中，則會插入一個檔案鏈結（link）到目前的文件中。

有些時候程式一次接收到數個檔案，並沒有什麼意義。再以 WordPad 爲例子，如果有數個檔案被丟進 WordPad 的非工作區中，WordPad 只會開啓其中一個檔案，而忽略其它檔案。如果這些檔案被丟進 WordPad 的工作區中，則 WordPad 會插入所有檔案的捷徑（shortcut）到目前的文件中。無論你的程式是否會用到這些檔案，你都有責任呼叫 DragFinish 函式來釋放記憶體區塊。

Touch 範例程式

Touch 程式（Touch.EXE）的原始碼顯示於**程式列表 7.1 ~ 7.3** 中，示範如何建立一個可

接受物件丟入（所謂的拖放目標）的程式。當你執行這個程式，會出現圖 **7.3** 所示的對話盒：

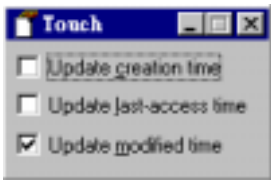


圖 7.3 Touch 程式

你可以在「檔案總管」之中選取一個（以上）的檔案拖放 Touch 程式的視窗上。Touch 程式會將拖放進來的檔案的日期和時間，以目前的系統時間更新之。Touch 程式的對話盒中有三個 checkbox，你可以利用它們來告訴系統說，你想要更新那一個項目。預設情況下，Touch 程式只會更新檔案最近被修改的時間，至於檔案建立時間以及最近被存取的時間不會被改變。如果你想讓 Touch 程式更新其它項目，可以在拖放檔案到 Touch 的視窗之前，先依你的意思選取或取消 checkbox 的選項。

建立此程式所需用到的檔案，列於表 **7.1** 中。

表 7.1 建立 Touch 程式所需的檔案

檔案	說明
Touch.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
Touch.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 Touch.RC 檔中所有資源的 ID。
Touch.ICO	主視窗圖示（icon）。
Touch.MAK	Visual C++ 的 MAK 檔。

當 Touch 的對話盒初始化時，其 Touch_OnInitDialog 函式負責設定 checkbox 的初始狀態。此外它也呼叫 DragAcceptFiles 函式，將 WS_EX_ACCEPTFILES 旗標設爲 on，好讓

「檔案總管」知道 Touch 程式的視窗可接受檔案的拖放動作。

我們最感興趣的是 Touch 程式如何處理這些被丟進來的檔案，也就是說，它究竟如何回應 WM_DROPFILES 訊息。當它收到這個訊息，會呼叫 Touch_OnDropFiles 函式：

```
void Touch_OnDropFiles (HWND hwnd, HDROP hdrop) {

   HANDLE hfile;
   SYSTEMTIME st;
   FILETIME ft;

   // Get the number of files that have been dropped on us.
   int nNumFiles = DragQueryFile(hdrop, 0xFFFFFFFF, NULL, 0);
   PBYTE p = GlobalLock(hdrop);

   TCHAR szBadFiles[10240] =
      __TEXT("The following file(s) could not be opened:\n");
   BOOL fAnyBadFiles = FALSE;

   // Get the current system time and convert it to a file time.
   GetSystemTime(&st);
   SystemTimeToFileTime(&st, &ft);

   while (nNumFiles--) {

      TCHAR szFilename[_MAX_PATH];

      // Get the path of a single file that has been dropped on us.
      DragQueryFile(hdrop, nNumFiles, szFilename, adgARRAY_SIZE(szFilename));

      // Attempt to open the file so that we can alter its timestamp.
      // NOTE: On Windows NT, FILE_FLAG_BACKUP_SEMANTICS allows
      // directories to be opened.  Windows 95 will not open directories.
      hfile = CreateFile(szFilename, GENERIC_WRITE,
         FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
         FILE_FLAG_SEQUENTIAL_SCAN | FILE_FLAG_BACKUP_SEMANTICS, NULL);

      if (hfile != INVALID_HANDLE_VALUE) {

         // If the file opened successfully, change the specified file times.
         adgVERIFY(SetFileTime(hfile,
            IsDlgButtonChecked(hwnd, IDC_UPDATECREATETIME)     ? &ft : NULL,
            IsDlgButtonChecked(hwnd, IDC_UPDATELASTACCESSTIME) ? &ft : NULL,
            IsDlgButtonChecked(hwnd, IDC_UPDATEMODIFIEDTIME)   ? &ft : NULL));
```

```
            CloseHandle(hfile);
         } else {

            if (0 == (GetFileAttributes(szFilename) & FILE_ATTRIBUTE_DIRECTORY)) {

               // We couldn't open the file, so we'll append the filename to the
               // list of bad files.  We will not consider failure to open a
               // directory to be a problem.
               lstrcat(szBadFiles, szFilename);
               lstrcat(szBadFiles, __TEXT("\n"));
               fAnyBadFiles = TRUE;
            }
         }
      }

      if (fAnyBadFiles) {

         // If there were any files appended to the szBadFiles string, display
         // the error message box.
         adgMB(szBadFiles);
      }

      // We must free the memory containing the structure of dropped files.
      DragFinish(hdrop);
   }
```

這個函式首先確定有多少個檔案被丟入，然後取得目前的系統時間。而後，這個函式進入一個迴路，企圖開啓所有檔案，並更新你想要更新的日期與時間。每一次迴路都會呼叫 DragQueryFile 函式以取得檔案的完整路徑與名稱。最後，檔案會被開啓，並呼叫 SetFileTime 函式，依照 checkbox 的設定，更新檔案的日期與時間。

請注意，除了檔案之外，使用者也可能從「檔案總管」中拖拉整個檔案目錄到 Touch 視窗上。當 Touch 程式企圖開啓檔案目錄，可分兩種情況來討論。如果 Touch 程式是在 Windows NT 環境下執行，CreateFile 函式可以成功開啓檔案目錄，因爲此一環境接受 FILE_FLAG_BACKUP_SEMANTICS 旗標（可開啓目錄）。然而，Windows 95 會忽略 FILE_FLAG_BACKUP_SEMANTICS 旗標，導至 CreateFile 函式無法開啓檔案目錄；它會傳回 INVALID_HANDLE_VALUE 表示失敗。

凡是 Touch 程式不能開啓的檔案，其路徑與檔案名稱都會被我加入一個所謂的錯誤檔案

列表（bad-files-list）中。然而我並不認爲檔案目錄的開啓失敗是個錯誤，所以檔案目錄不會被加入到錯誤檔案列表之中。我們可以呼叫 GetFileAttribute 函式，並將其傳回值與 FILE_ATTRIBUTE_DIRECTORY 旗標做“AND”運算，以此來判斷拖放進來的物件是否爲一個檔案目錄。

最後，在 Touch_OnDropFiles 函式返回之前，應該呼叫 DragFinish 函式，釋放記憶體區塊。如果你沒有這麼做，你的行程位址空間中會遺失一塊記憶體，直至行程結束爲止。

Touch.ICO

程式列表 7.1　Touch.C

```
#0001  /****************************************************************
#0002  Module name: Touch.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates using a dialog box for an application's main window
#0006  ****************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single-line comment */
#0013  #include <tchar.h>
#0014  #include "resource.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  BOOL Touch_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0021
#0022    // Default to setting the file's last modified time only.
#0023    CheckDlgButton(hwnd, IDC_UPDATECREATETIME,     BST_UNCHECKED);
#0024    CheckDlgButton(hwnd, IDC_UPDATELASTACCESSTIME, BST_UNCHECKED);
#0025    CheckDlgButton(hwnd, IDC_UPDATEMODIFIEDTIME,   BST_CHECKED);
#0026
#0027    adgSETDLGICONS(hwnd, IDI_TOUCH, IDI_TOUCH);
#0028
#0029    // Notify dropfile source applications that our window accepts dropped files.
#0030    DragAcceptFiles(hwnd, TRUE);
```

```
#0031
#0032    return(TRUE);                          // Accepts default focus window.
#0033 }
#0034
#0035
#0036 ////////////////////////////////////////////////////////////////
#0037
#0038
#0039 void Touch_OnDropFiles (HWND hwnd, HDROP hdrop) {
#0040
#0041    HANDLE hfile;
#0042    SYSTEMTIME st;
#0043    FILETIME ft;
#0044
#0045    // Get the number of files that have been dropped on us.
#0046    int nNumFiles = DragQueryFile(hdrop, 0xFFFFFFFF, NULL, 0);
#0047    PBYTE p = GlobalLock(hdrop);
#0048
#0049    TCHAR szBadFiles[10240] =
#0050       __TEXT("The following file(s) could not be opened:\n");
#0051    BOOL fAnyBadFiles = FALSE;
#0052
#0053    // Get the current system time and convert it to a file time.
#0054    GetSystemTime(&st);
#0055    SystemTimeToFileTime(&st, &ft);
#0056
#0057    while (nNumFiles--) {
#0058
#0059       TCHAR szFilename[_MAX_PATH];
#0060
#0061       // Get the path of a single file that has been dropped on us.
#0062       DragQueryFile(hdrop, nNumFiles, szFilename, adgARRAY_SIZE(szFilename));
#0063
#0064       // Attempt to open the file so that we can alter its timestamp.
#0065       // NOTE: On Windows NT, FILE_FLAG_BACKUP_SEMANTICS allows
#0066       // directories to be opened.  Windows 95 will not open directories.
#0067       hfile = CreateFile(szFilename, GENERIC_WRITE,
#0068          FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
#0069          FILE_FLAG_SEQUENTIAL_SCAN | FILE_FLAG_BACKUP_SEMANTICS, NULL);
#0070
#0071       if (hfile != INVALID_HANDLE_VALUE) {
#0072
#0073          // If the file opened successfully, change the specified file times.
#0074          adgVERIFY(SetFileTime(hfile,
#0075             IsDlgButtonChecked(hwnd, IDC_UPDATECREATETIME)    ? &ft : NULL,
#0076             IsDlgButtonChecked(hwnd, IDC_UPDATELASTACCESSTIME) ? &ft : NULL,
```

```
#0077            IsDlgButtonChecked(hwnd, IDC_UPDATEMODIFIEDTIME)   ? &ft : NULL));
#0078         CloseHandle(hfile);
#0079       } else {
#0080
#0081         if (0 == (GetFileAttributes(szFilename) & FILE_ATTRIBUTE_DIRECTORY)) {
#0082
#0083            // We couldn't open the file, so we'll append the filename to the
#0084            // list of bad files.  We will not consider failure to open a
#0085            // directory to be a problem.
#0086            lstrcat(szBadFiles, szFilename);
#0087            lstrcat(szBadFiles, __TEXT("\n"));
#0088            fAnyBadFiles = TRUE;
#0089         }
#0090       }
#0091    }
#0092
#0093    if (fAnyBadFiles) {
#0094
#0095       // If there were any files appended to the szBadFiles string, display
#0096       // the error message box.
#0097       adgMB(szBadFiles);
#0098    }
#0099
#0100    // We must free the memory containing the structure of dropped files.
#0101    DragFinish(hdrop);
#0102 }
#0103
#0104
#0105 ////////////////////////////////////////////////////////////////
#0106
#0107
#0108 void Touch_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0109
#0110    switch (id) {
#0111       case IDCANCEL:            // Allows dialog box to close
#0112          EndDialog(hwnd, id);
#0113          break;
#0114    }
#0115 }
#0116
#0117
#0118 ////////////////////////////////////////////////////////////////
#0119
#0120
#0121 BOOL WINAPI Touch_DlgProc (HWND hwnd, UINT uMsg,
#0122    WPARAM wParam, LPARAM lParam) {
```

```
#0123
#0124    switch (uMsg) {
#0125
#0126       // Standard Window's messages
#0127       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG,   Touch_OnInitDialog);
#0128       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,     Touch_OnCommand);
#0129       adgHANDLE_DLGMSG(hwnd, WM_DROPFILES,    Touch_OnDropFiles);
#0130    }
#0131    return(FALSE);              // We didn't process the message.
#0132 }
#0133
#0134
#0135 ////////////////////////////////////////////////////////////////
#0136
#0137
#0138 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0139    LPSTR lpszCmdLine, int nCmdShow) {
#0140
#0141    adgWARNIFUNICODEUNDERWIN95();
#0142    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_TOUCH),
#0143       NULL, Touch_DlgProc));
#0144
#0145    return(0);
#0146 }
#0147
#0148
#0149 ////////////////////////// End of File //////////////////////////
```

程式列表 7.2　Touch.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "resource.h"
#0004
#0005 #define APSTUDIO_READONLY_SYMBOLS
#0006 ////////////////////////////////////////////////////////////////
#0007 //
#0008 // Generated from the TEXTINCLUDE 2 resource.
#0009 //
#0010 #include "windows.h"
#0011
#0012 ////////////////////////////////////////////////////////////////
#0013 #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
```

```
#0016  #ifdef APSTUDIO_INVOKED
#0017  //////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  //////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  //////////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_TOUCH DIALOG DISCARDABLE  -32768, 5, 100, 48
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0050  CAPTION "Touch"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      CONTROL        "Update &creation time",IDC_UPDATECREATETIME,"Button",
#0054                     BS_AUTOCHECKBOX | WS_TABSTOP,4,4,79,10
#0055      CONTROL        "Update &last-access time",IDC_UPDATELASTACCESSTIME,
#0056                     "Button",BS_AUTOCHECKBOX | WS_TABSTOP,4,18,89,10
#0057      CONTROL        "Update &modified time",IDC_UPDATEMODIFIEDTIME,"Button",
#0058                     BS_AUTOCHECKBOX | WS_TABSTOP,4,32,80,10
#0059  END
#0060
#0061
```

```
#0062  //////////////////////////////////////////////////////////////////
#0063  //
#0064  // Icon
#0065  //
#0066
#0067  IDI_TOUCH             ICON    DISCARDABLE     "Touch.ico"
#0068
#0069  #ifndef APSTUDIO_INVOKED
#0070  //////////////////////////////////////////////////////////////////
#0071  //
#0072  // Generated from the TEXTINCLUDE 3 resource.
#0073  //
#0074
#0075
#0076  //////////////////////////////////////////////////////////////////
#0077  #endif    // not APSTUDIO_INVOKED
```

程式列表 7.3　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by Touch.rc
#0004  //
#0005  #define IDD_TOUCH                   103
#0006  #define IDI_TOUCH                   104
#0007  #define IDC_UPDATECREATETIME          1000
#0008  #define IDC_UPDATELASTACCESSTIME       1001
#0009  #define IDC_UPDATEMODIFIEDTIME        1002
#0010  #define IDC_SYSTEMTIME               1003
#0011
#0012  // Next default values for new objects
#0013  //
#0014  #ifdef APSTUDIO_INVOKED
#0015  #ifndef APSTUDIO_READONLY_SYMBOLS
#0016  #define _APS_NEXT_RESOURCE_VALUE       106
#0017  #define _APS_NEXT_COMMAND_VALUE        40001
#0018  #define _APS_NEXT_CONTROL_VALUE        1006
#0019  #define _APS_NEXT_SYMED_VALUE         101
#0020  #endif
#0021  #endif
```

技術－個別拖放物件的供應者

可能你已注意到，截至目前，所有的例子都是以檔案總管（Explorer）做爲拖放物件的供應者，原因是 Windows 檔案總管是唯一內附於 Windows 的一個拖放物件供應者。然而，你可以使用定義在 ShlObj.h 表頭檔中的 DROPFILES 結構，建立自己的拖放物件供應程式：

```
typedef struct _DROPFILES {
   DWORD pFiles;  // Offset of file list
   POINT pt;      // Position of mouse cursor (client coordinates)
   BOOL fNC;      // Was mouse in the window's non-client area
   BOOL fWide;    //TRUE if file contains WIDE character, FALSE otherwise
} DROPFILES, FAR * LPDROPFILES;
```

當使用者將檔案拖放到某個視窗時，檔案總管便會產生一個這樣的結構。在配置一塊足以容納此結構的記憶體區塊之後，pFiles 欄位將表示這個結構佔有多少個位元組，因爲從那裡開始就是放置路徑與檔案名稱的地方。

pt 欄位記錄著滑鼠按鍵被放開時，滑鼠的 x 和 y 座標（此座標是滑鼠游標之下的視窗工作區座標）。當使用者放開滑鼠按鍵，如果滑鼠落在視窗的非工作區中，Fnc 欄位會被設爲 TRUE。緊接著這個結構之後記錄的是「檔案總管」中所選取的每一個檔案的路徑與名稱。這些路徑與檔案名稱可能是 ANSI 字元或 Unicode 字元；DROPFILES 結構中的 fWide 欄位用以表示哪一種字元。每一個完整的檔案名稱皆以 0 作爲結束字元。最後一筆完整的檔案名稱之後，會再多加一個 0，表示這是最後一筆資料。

要成爲一個拖放物件的供應者，首先你必須建立一塊如上所述的記憶體區塊。我寫了兩個函式來幫助我們完成這項工作：第一個函式是 DFSrc_Create，用來幫你配置一塊足以容納 DROPFILES 結構的記憶體區塊，並設定其中的四個欄位。這個函式會傳回此記憶體區塊的代碼；如有錯誤發生則傳回 NULL：

```
HDROP WINAPI DFSrc_Create (PPOINT ppt, BOOL fNC, BOOL fWide) {

   HDROP hdrop;
   LPDROPFILES pDropFiles;

   // Allocate dynamic memory for the DFSRC data structure and for the extra
   // zero character identifying that there are no path names in the block yet.
   hdrop = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT,
      sizeof(DROPFILES) + (fWide ? sizeof(WCHAR) : sizeof(char)));

   // If successful, lock block and initialize the data members.
   if (hdrop != NULL) {
      pDropFiles = (LPDROPFILES) GlobalLock(hdrop);
      pDropFiles->pFiles = sizeof(*pDropFiles);
      pDropFiles->pt = *ppt;
      pDropFiles->fNC = fNC;
      pDropFiles->fWide = fWide;
      GlobalUnlock(hdrop);
   }

   return(hdrop);
}
```

第二個函式是 DFSrc_AppendPathname，它會被重複呼叫，將新的路徑與檔案名稱記錄到記憶體區塊中：

```
HDROP WINAPI DFSrc_AppendPathname (HDROP hdrop, PVOID pvPathname) {

   LPDROPFILES pDropFiles = (LPDROPFILES) GlobalLock(hdrop);

   // Point to first path name in list.
   PSTR  szPathA = (PSTR) pDropFiles + pDropFiles->pFiles;
   PWSTR szPathW = (PWSTR) szPathA;
   int nOffsetOfNewPathname, nPathSize;

   if (pDropFiles->fWide) {

      // Search for a path name where first character is a zero character.
      while (*szPathW) {         // While the first character is nonzero
         while (*szPathW)        // Find end of current path.
            szPathW++;
         szPathW++;             // Skip over the zero character.
      }

      // Get the offset from the beginning of the block
```

```
      // where the new path name should go.
      nOffsetOfNewPathname = ((PBYTE) szPathW - (PBYTE) pDropFiles);

      // Get the number of bytes needed for the new path name,
      // it's terminating zero character, and the zero-length
      // path name that marks the end of the list of path names.
      nPathSize = sizeof(WCHAR) * (wcslen(pvPathname) + 2);
   } else {

      // Search for a pathname where first character is a zero character.
      while (*szPathA) {         // While the first character is nonzero
         while (*szPathA)        // Find end of current path.
            szPathA++;
         szPathA++;              // Skip over the zero character.
      }

      // Get the offset from the beginning of the block
      // where the new path name should go.
      nOffsetOfNewPathname = ((PBYTE) szPathA - (PBYTE) pDropFiles);

      // Get the number of bytes needed for the new path name,
      // it's terminating zero character, and the zero-length
      // path name that marks the end of the list of path names.
      nPathSize = sizeof(char) * (strlen(pvPathname) + 2);
   }

   GlobalUnlock(hdrop);

   // Increase block size to accommodate new path name.
   hdrop = GlobalReAlloc(hdrop, nPathSize + nOffsetOfNewPathname,
      GMEM_MOVEABLE | GMEM_ZEROINIT);

   // If successful, append the path name to the end of the block.
   if (hdrop != NULL) {
      pDropFiles = (LPDROPFILES) GlobalLock(hdrop);

   if (pDropFiles->fWide)
      wcscpy((PWSTR) ((PSTR) pDropFiles + nOffsetOfNewPathname), pvPathname);
   else
      strcpy((PSTR)  ((PSTR) pDropFiles + nOffsetOfNewPathname), pvPathname);

      GlobalUnlock(hdrop);
   }

   return(hdrop);              // Returns the new handle to the block.
}
```

此函式的第一個參數是記憶體區塊代碼，那是由 DFSrc_Create 函式所產生，或是由先前呼叫的 DFSrc_AppendPathname 函式所產生。接著 DFSrc_AppendPathname 函式先鎖定該記憶體區塊，並計算目前已經用掉的位元組大小，然後呼叫 GlobalReAlloc 函式來加大記憶體區塊空間，使它可以容納目前的資料和新加入的路徑與檔案名稱。如果一切順利，新的路徑與檔案名稱會被拷貝到記憶體區塊尾端，函式並傳回新的記憶體區塊代碼。

在利用 DFSrc_AppendPathname 函式將所有的路徑與檔案名稱加入記憶體區塊的尾端之後，滑鼠游標下的視窗會處理這些被丟入的檔案。拖放物件供應程式應該“post”一個 WM_DROPFILES 訊息給拖放目標程式，其 wParam 參數應該放置 DFSrc_AppendPathname 函式傳回的記憶體區塊代碼。這裡我們必須使用 PostMessage 函式來“post”訊息，而不要使用 SendMessage 函式來“send”訊息，這樣子拖放物件供應程式的執行緒才不會「癡癡等待拖放目標程式處理完拖放過去的檔案後，才繼續它自己的工作」（侯俊傑註：因爲 PostMessage 是非同步動作，SendMessage 是同步動作）。

如果你是一位精明的 Win32 程式設計者，你應該可以立刻看出這項技術的問題所在：每一個行程有自己的私有位址空間。雖然一個行程（譬如說「檔案總管」）可以配置一塊記憶體並設定其內容，然後將此記憶體區塊代碼經由一個視窗訊息傳遞給另一個行程，然而收到此訊息的行程卻無法順利使用這個代碼。原因是某個行程的位址空間中的記憶體不能夠被另一個行程取用（侯俊傑註：因爲 Win32 實現出「分離位址空間」，而非如 16 位元 Windows 般的「共享位址空間」）。那麼微軟如何能夠允許你將「檔案總管」中的物件（檔案）拖放到其他程式的視窗去呢？

答案在於，PostMessage 函式以特殊方式對待 WM_DROPFILES 訊息。如果你“post”一個 WM_DROPFILES 訊息，PostMessage 函式一看到它，便在訊息接受端（一個行程）的位址空間中配置一塊記憶體，並將原記憶體區塊中的內容拷貝到新配置的記憶體區塊中。PostMessage 函式然後會呼叫 GlobalFree 函式，釋放原來的記憶體區塊；並將 wParam 參數設定爲新記憶體區塊的代碼。當拖放目標程式的視窗函式收到了 WM_DROPFILES 訊息，其 wParam 參數所指的是自己這個行程位址空間中的記憶體區塊，DragQueryPoint、DragQueryFile 和 DragFinish 都所以毫無問題地使用這塊記憶體。

DFSrcDem 範例程式

DFSrc 範例程式（DFSrcDem.EXE）的原始碼顯示在**程式列表 7.4 ~ 7.8** 中，示範如何建立一個拖放物件供應程式。當你執行這個程式，出現**圖 7.4** 的對話盒。

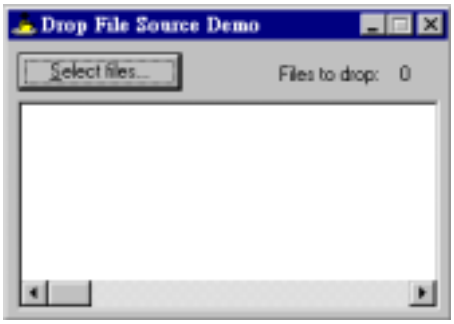


圖 7.4　DFSrcDem 程式

一開始，對話盒中並沒有任何可被拖放的檔案。然而如果你選按【Select files】鈕，就會顯示一個【開啓舊檔】對話盒，你可以從中選取一個或多個檔案。關閉【開啓舊檔】對話盒後，在 DFSrcDem 主對話盒的 listbox 中就會出現你剛剛所選的檔案，每一個檔案皆會顯現其完整的路徑名稱。如欲將這些檔案拖放至其它程式中，你只需選擇任何一個檔案，按住滑鼠的按鍵不放，移動至其它程式的視窗上（此時滑鼠游標的形狀會改變）。當你將滑鼠移動到某一個視窗的領空，DFSrcDem 程式會檢查該視窗，看看是否 WS_EX_ACCEPTFILES 視窗風格爲 on。如果是 on，DFSrcDem 就將滑鼠游標形狀改變爲一個箭頭；如果是 off，就顯示「禁止進入」的游標形狀。

當你在設立了 WS_EX_ACCEPTFILES 旗標位元的視窗上放開滑鼠按鍵，DFSrcDem 程式會產生一塊記錄著 DROPFILES 資訊的記憶體，並 "post" 一個 WM_DROPFILES 訊息到目標視窗（也就是滑鼠游標下的視窗）。拖放目標程式接受了檔案之後，就開始執行前面我們討論過的動作。

建立此程式所需用到的檔案，列於**表 7.2**。

表 7.2　建立 DFSrcDem 程式所需的檔案

檔案	說明
DFSrcDem.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
DFSrcDem.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
DFSrc.C	此檔案建立一個函式庫，提供許多函式，使你能夠比較容易地建立一個拖放物件供應程式。
DFSrc.H	DFSrc.C 檔中的函式原型宣告。任何程式只要用到 DFSrc.C 檔中的函式，皆需含入此檔。
Resource.H	內含 DFSrcDem.RC 檔中所有資源的 ID。
DFSrcDem.ICO	主視窗圖示（icon）。
DFSrcDem.MAK	Visual C++ 的 MAK 檔。

當你按下【Select files】按鈕，程式會去呼叫 DFSrcDem_OnCommand 函式，這個函式設定一個 OPENFILENAME 結構，然後呼叫 GetOpenFileName 函式。如果使用者選取檔案後按下【開啓舊檔】的按鈕，GetOpenFileName 函式會傳回 TRUE，並將被選取的檔案顯示在 listbox 中。如果使用者僅選取了一個檔案，緩衝區中會記錄著這個檔案的完整路徑名稱與檔名。舉個例，如果僅選取 Notepad，則緩衝區內容如下：

```
"C:\\Windows\\NOTEPAD.EXE\0"
```
[2]

然而，如果使用者選取了數個檔案，其 OPENFILENAME 結構的 lpstrFile 緩衝區會記錄著以 0 結尾的路徑名稱，後面再緊跟著一個以‘\0’結尾的檔案名稱。整個緩衝區的尾端會再有一個額外的‘\0’字元，表示所有檔案的結尾。舉個例子，如果你從 C:\Windows 目錄下選取了筆記本（Notepad）、小畫家（PBrush）和小算盤（Calc），緩衝區的內容如下：

[2] 注意，緩衝區尾端並沒有另一個“\0”。

```
"C:\\Windows\0NOTEPAD.EXE\0PBRUSH.EXE\0CALC.EXE"
```
[3]

由於 lpstrFile 緩衝區會出現兩種不同的格式，所以我建立一些函式來剖析這項資訊。這些函式都撰寫在 DFSrc.C 檔中，你可以自行應用到你自己的程式中。我會概略介紹這些函式，但我衷心希望你能夠仔細揣摩它們。

MultiStrUtil_FindStr 函式，它會傳回緩衝區中的一個字串的位址，該字串以 0 結尾：

```
LPCTSTR WINAPI MultiStrUtil_FindStr (LPCTSTR szStrAll, int nIndex, PINT pnMax);
```

其中的 szStrAll 參數指向一個記錄著多個字串的緩衝區，nIndex 參數則表示你想取出第幾個字串。MultiStrUtil_FindStr 函式的傳回值是一個字串位址。如果 nIndex 參數（索引值）太大的話，則傳回 NULL。如果你傳入一個整數變數的位址做爲最後一個參數，並且把 nIndex 參數設爲 -1，那麼這個函式就會傳回緩衝區中的字串個數。

FileOpenUtil_AreMultipleFilesSelected 函式，接受一個早先由 GetOpenFileName 函式所設定的 OPENFILENAME 結構的指標。如果 lpstrFile 緩衝區中記錄著多個檔案，則 FileOpenUtil_AreMultipleFilesSelected 函式會傳回 TRUE；如果緩衝區中僅含一個檔案，也就是說只有一個檔案被選取，則此函式傳回 FALSE：

```
BOOL WINAPI FileOpenUtil_AreMultipleFilesSelected (OPENFILENAME *pofn);
```

FileOpenUtil_GetNumFiles 函式，也是接受一個早先由 GetOpenFileName 函式所設定的 OPENFILENAME 結構的指標。它會傳回 lpstrFile 緩衝區中所記錄的檔案個數：

```
int WINAPI FileOpenUtil_GetNumFiles (OPENFILENAME *pofn);
```

---

[3] 這裡所描述的格式是當 OFN_EXPLORER 旗標被設立時才成立。Windows NT 3.51 不理會 OFN_EXPLORER 旗標，它使用較舊的檔案格式 -- 以空白來區分檔案名稱。爲了讓 DFSrcDem 程式也能在 Windows NT 3.51 中正常運作，我寫了一些程式碼來轉換緩衝區中原本「以空白字元區分檔案」的格式，改爲以 0 字元來區分。

FileOpenUtil_GetFile 函式，有三個參數：一個是指向 OPENFILENAME 結構的指標，一個是索引值，還有一個是指向另一緩衝區的指標。這個函式會將索引值所指向的檔案名稱（包含路徑）填入到 szPathname 緩衝區中：

```
int WINAPI FileOpenUtil_GetFile (OPENFILENAME *pofn, int nIndex,
   LPTSTR szPathname);
```

DFSrcDem 藉著處理三種訊息來完成 drag and drop 的功能：WM_LBUTTONDOWN、WM_MOUSEMOVE 和 WM_LBUTTONUP。讓我們先看 DFSrcDem_OnLButtonDown 函式：

```
void DFSrcDemo_OnLButtonDown (HWND hwnd, BOOL fDoubleClick,
   int nIndex, int y, UINT keyFlags) {

   // Make sure that there are some files to be dropped.
   if (ListBox_GetCount(GetDlgItem(hwnd, IDC_PATHNAMELIST)) == 0) {
      adgMB(__TEXT("No files to drop."));
      return;
   }

   // Initiate the drag-and-drop sequence.
   SetCapture(hwnd);
}
```

這個函式首先檢查看看 listbox 中是否有檔案存在；如果有的話，就設定滑鼠捕捉權（mouse capture），準備開始拖放（drag and drop）程序。WM_MOUSEMOVE 和 WM_LBUTTONDOWN 的訊息處理常式會完成拖放程序的後續動作。

當使用者移動滑鼠，程式會呼叫 DFSrcDemo_OnMouseMove 函式。此函式的唯一任務就是負責改變滑鼠游標的形狀，以便使用者能很清楚地看出檔案是否能被丟入：

```
void DFSrcDemo_OnMouseMove (HWND hwnd, int x, int y, UINT keyFlags) {

   // If we don't have capture, the user didn't initiate the drag-and-drop
   // sequence -- we have nothing to do here.
   if (GetCapture() != hwnd)
      return;
```

```
   // Get cursor position and window under the cursor.
   SetCursor(IsWindow(DFSrc_OkToDrop(NULL)) ?
      LoadCursor(GetWindowInstance(hwnd), MAKEINTRESOURCE(IDC_DROPOK)) :
      LoadCursor(NULL, IDC_NO));
}
```

真正負責判斷檔案是否能被丟入的是 DFSrc.C 檔中的 DFSrc_OkToDrop 函式：

```
// Macro to ease source code readability
#define IsAcceptingFiles(hwnd) \
   ((GetWindowExStyle((hwnd)) & WS_EX_ACCEPTFILES) != 0)

////////////////////////////////////////////////////////////////////

HWND WINAPI DFSrc_OkToDrop (PPOINT ppt) {

   POINT ptMousePos;
   HWND hwndTarget;

   if (ppt == NULL) {

      // The caller passes NULL. Assume that they want the mouse position
      // at the time of the last retrieved window message.
      ptMousePos.x = LOWORD(GetMessagePos());
      ptMousePos.y = HIWORD(GetMessagePos());
   }

   hwndTarget = WindowFromPoint(ptMousePos);

   // See if the target window or any of its parent windows are prepared to
   // accept dropped files.
   while (IsWindow(hwndTarget) && !IsAcceptingFiles(hwndTarget))
      hwndTarget = GetParent(hwndTarget);

   // If it's OK to drop files, return the HWND of the target window or NULL
   // if we do not have a target window accepting files.
   // the WS_EX_ACCEPTFILES style.
   return((IsWindow(hwndTarget) && IsAcceptingFiles(hwndTarget))
      ? hwndTarget : NULL);
}
```

每當 DFSrc_OkToDrop 函式被呼叫，它都會要求滑鼠位置。然後它就會檢查滑鼠位置下

的視窗是否有將 WS_EX_ACCEPTFILES 風格位元設立爲 on。如果答案是否定的，則再檢查這視窗的父視窗，一直找下去。如果都沒有找到一個視窗曾經設立 WS_EX_ACCEPTFILES 爲 on，就傳回 NULL。然而如果找到其中有一個視窗設立了 WS_EX_ACCEPTFILES 爲 on，此函式就傳回該視窗的視窗代碼。DFSrcDem_OnMouseMove 函式會利用這個傳回值來決定應該使用哪一種滑鼠游標。

當滑鼠按鍵被放開，程式會呼叫 DFSrcDem_OnLButtonUp 函式：

```
void DFSrcDemo_OnLButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {

   HWND hwndTarget, hwndLB = GetDlgItem(hwnd, IDC_PATHNAMELIST);
   RECT rc;
   POINT ptMousePos;
   int nIndex, nNumFiles = ListBox_GetCount(hwndLB);
   HDROP hdrop, hdropT;
   TCHAR szPathName[_MAX_PATH];

   // If we don't have capture, the user didn't initiate the drag-and-drop
   // sequence -- we have nothing to do here.
   if (GetCapture() != hwnd)
      return;

   // End the drag-and-drop sequence.
   ReleaseCapture();

   // Get the HWND of the target window.
   hwndTarget = DFSrc_OkToDrop(NULL);

   if (!IsWindow(hwndTarget)) {

      // If the target window is invalid, we have nothing else to do.
      return;
   }

   // Get the client rectangle of the target window and convert the mouse
   // position to the target's client coordinates so that we can tell if
   // the mouse is in the target's client area when we call DFSrc_Create.
   GetClientRect(hwndTarget, &rc);
   ScreenToClient(hwndTarget, &ptMousePos);

   // Create dropfile memory block and initialize it.
#ifdef UNICODE
```

```
   hdrop = DFSrc_Create(&ptMousePos, !PtInRect(&rc, ptMousePos), TRUE);
#else
   hdrop = DFSrc_Create(&ptMousePos, !PtInRect(&rc, ptMousePos), FALSE);
#endif

   if (hdrop == NULL) {
      adgMB(__TEXT("Insufficient memory to drop file(s)."));
      return;
   }

   // Append each path name to the dropfile memory block.
   for (nIndex = 0; nIndex < nNumFiles; nIndex++) {

      ListBox_GetText(hwndLB, nIndex, szPathName);
      hdropT = DFSrc_AppendPathname(hdrop, szPathName);

      if (hdropT == NULL) {
         adgMB(__TEXT("Insufficient memory to drop file(s)."));
         hdrop = GlobalFree(hdrop);
         break;                // Terminates the 'for' loop.
      } else {
         hdrop = hdropT;
      }
   }

   if (hdrop != NULL) {

      // All path names appended successfully, post the message to the
      // dropfile target window.
      FORWARD_WM_DROPFILES(hwndTarget, hdrop, PostMessage);

      // Update the window's caption to reflect the number of selected files
      // available for dropping.
      SetDlgItemInt(hwnd, IDC_NUMFILES, 0, FALSE);
      ListBox_ResetContent(hwndLB);

      // NOTE: We should not call GlobalFree passing in hdrop.  The system
      // will free it for us after it creates a copy of it in the target
      // process's address space.  The target process is responsible for
      // freeing it's copy of the hdrop by calling DragFinish.
   }
}
```

這個函式首先呼叫 ReleaseCapture，取消滑鼠的捕捉權。接著再呼叫 DFSrc_OkToDrop，

判斷滑鼠按鍵被放開時，其下的視窗是否爲一個拖放目標視窗。如果是的話，就呼叫 DFSrc_Create 函式，傳入滑鼠的位置（轉換成目的視窗工作區的座標），以及一個布林值（boolean），表示滑鼠是否落在視窗工作區中，還有另一個布氏值，表示我們究竟想以 Unicode 字串還是 ANSI 字串將路徑和檔名加到資料緩衝區中。這個參數的設定是根據「程式被編譯時，UNICODE 是否有被定義」而定。

如果 DFSrc_Create 函式傳回一個不是 NULL 的代碼，那麼 DFSrcDem_OnLButtonUp 函式接下來會從 listbox 中取得每一個檔案名稱（包含路徑），並將它加入資料緩衝區中。所有檔案都被順利地取出並加入緩衝區之後，就 “post” 一個 WM_DROPFILES 訊息（其 wParam 參數爲 DROPFILES 記憶體區塊代碼）到拖放目標視窗：

```
FORWARD_WM_DROPFILES(hwndTarget, hdrop, PostMessage);
```

這時，DFSrcDem 程式會清除 listbox 中的所有內容。注意，這個程式並不釋放記憶體區塊，因爲 PostMessage 會幫我們釋放。拖放目標程式則需呼叫 DragFinish 函式來釋放這份被複製的記憶體。

DFSrcDem.ICO

程式列表 7.4　DFSrcDem.C

```
#0001  /******************************************************************
#0002  Module name: DFSrcDem.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates how to create a dropfile source application
#0006  ******************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details. */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single-line comment */
#0013  #include "DFSrc.h"
#0014  #include "Resource.h"
#0015
#0016
```

```
#0017 ////////////////////////////////////////////////////////////////
#0018
#0019
#0020 BOOL DFSrcDemo_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0021
#0022    adgSETDLGICONS(hwnd, IDI_DFSRCDEMO, IDI_DFSRCDEMO);
#0023    ListBox_SetHorizontalExtent(GetDlgItem(hwnd, IDC_PATHNAMELIST),
#0024       _MAX_PATH * LOWORD(GetDialogBaseUnits()));
#0025
#0026    return(TRUE);               // Accepts default focus window.
#0027 }
#0028
#0029
#0030 ////////////////////////////////////////////////////////////////
#0031
#0032
#0033 void DFSrcDemo_OnCommand (HWND hwnd, int id,
#0034    HWND hwndCtl, UINT codeNotify) {
#0035
#0036    TCHAR szAllFileNames[1024]  = { 0 }, szPathname[_MAX_PATH];
#0037    OPENFILENAME ofn;
#0038    int nNumFiles, nIndex;
#0039    HWND hwndLB = GetDlgItem(hwnd, IDC_PATHNAMELIST);
#0040    LPTSTR szT;
#0041
#0042    switch (id) {
#0043       case IDCANCEL:            // Allows dialog box to close.
#0044          EndDialog(hwnd, id);
#0045          break;
#0046
#0047       case IDC_SELECTFILES:      // "Select Files" button
#0048
#0049          // Initialize structure for calling the "Open File" common dialog box.
#0050          adgINITSTRUCT(ofn, TRUE);
#0051          ofn.hwndOwner = hwnd;
#0052          ofn.lpstrFilter = __TEXT("All files\0*.*\0");
#0053          ofn.Flags = OFN_ALLOWMULTISELECT |
#0054             OFN_FILEMUSTEXIST | OFN_HIDEREADONLY | OFN_EXPLORER;
#0055
#0056          // Set up the buffer to receive the selected file(s).
#0057          szAllFileNames[0] = 0;
#0058          ofn.lpstrFile = szAllFileNames;
#0059          ofn.nMaxFile = adgARRAY_SIZE(szAllFileNames);
#0060
#0061          ListBox_ResetContent(hwndLB);
#0062          if (GetOpenFileName(&ofn)) {
```

```
#0063
#0064  #ifndef WINDOWSNT_COMPATIBILITY
#0065
#0066            // Windows NT 3.51 ignores the OFN_EXPLORER flag. This means that
#0067            // the files in the returned buffer are space delimited instead of
#0068            // zero-character delimited (like Windows 95). So, the following
#0069            // 'if' block converts the space delimiters to zero-character
#0070            // delimiters. When Windows NT supports OFN_EXPLORER, this 'if'
#0071            // block should be removed.
#0072
#0073            if (ofn.lpstrFile[ofn.nFileOffset - 1] == __TEXT(' ')) {
#0074
#0075               // If the buffer is space delmited, convert all space
#0076               // characters to zero characters.
#0077               for (szT = ofn.lpstrFile; *szT != 0; szT++)
#0078                  if (*szT == __TEXT(' '))
#0079                     *szT = 0;
#0080               // Extra terminate zero character to mark end of all strings
#0081               *szT = 0;
#0082            }
#0083  #endif
#0084
#0085            nNumFiles = FileOpenUtil_GetNumFiles(&ofn);
#0086            for (nIndex = 0; nIndex < nNumFiles; nIndex++) {
#0087
#0088               // Get the full path name to append.
#0089               FileOpenUtil_GetFile(&ofn, nIndex, szPathname);
#0090               ListBox_AddString(hwndLB, szPathname);
#0091            }
#0092         } else {
#0093            nNumFiles = 0;
#0094         }
#0095
#0096         // Update the "Files to Drop" static window to reflect the number
#0097         // of selected files available for dropping.
#0098         SetDlgItemInt(hwnd, IDC_NUMFILES, nNumFiles, FALSE);
#0099         break;
#0100    }
#0101  }
#0102
#0103
#0104  ////////////////////////////////////////////////////////////////
#0105
#0106
#0107  void DFSrcDemo_OnLButtonDown (HWND hwnd, BOOL fDoubleClick,
#0108    int nIndex, int y, UINT keyFlags) {
```

```
#0109
#0110    // Make sure that there are some files to be dropped.
#0111    if (ListBox_GetCount(GetDlgItem(hwnd, IDC_PATHNAMELIST)) == 0) {
#0112       adgMB(__TEXT("No files to drop."));
#0113       return;
#0114    }
#0115
#0116    // Initiate the drag-and-drop sequence.
#0117    SetCapture(hwnd);
#0118 }
#0119
#0120
#0121 ////////////////////////////////////////////////////////////////
#0122
#0123
#0124 void DFSrcDemo_OnMouseMove (HWND hwnd, int x, int y, UINT keyFlags) {
#0125
#0126    // If we don't have capture, the user didn't initiate the drag-and-drop
#0127    // sequence -- we have nothing to do here.
#0128    if (GetCapture() != hwnd)
#0129       return;
#0130
#0131    // Get cursor position and window under the cursor.
#0132    SetCursor(IsWindow(DFSrc_OkToDrop(NULL)) ?
#0133       LoadCursor(GetWindowInstance(hwnd), MAKEINTRESOURCE(IDC_DROPOK)) :
#0134       LoadCursor(NULL, IDC_NO));
#0135 }
#0136
#0137
#0138 ////////////////////////////////////////////////////////////////
#0139
#0140
#0141 void DFSrcDemo_OnLButtonUp (HWND hwnd, int x, int y, UINT keyFlags) {
#0142
#0143    HWND hwndTarget, hwndLB = GetDlgItem(hwnd, IDC_PATHNAMELIST);
#0144    RECT rc;
#0145    POINT ptMousePos;
#0146    int nIndex, nNumFiles = ListBox_GetCount(hwndLB);
#0147    HDROP hdrop, hdropT;
#0148    TCHAR szPathName[_MAX_PATH];
#0149
#0150    // If we don't have capture, the user didn't initiate the drag-and-drop
#0151    // sequence -- we have nothing to do here.
#0152    if (GetCapture() != hwnd)
#0153       return;
#0154
```

```
#0155    // End the drag-and-drop sequence.
#0156    ReleaseCapture();
#0157
#0158    // Get the HWND of the target window.
#0159    hwndTarget = DFSrc_OkToDrop(NULL);
#0160
#0161    if (!IsWindow(hwndTarget)) {
#0162
#0163       // If the target window is invalid, we have nothing else to do.
#0164       return;
#0165    }
#0166
#0167    // Get the client rectangle of the target window and convert the mouse
#0168    // position to the target's client coordinates so that we can tell if
#0169    // the mouse is in the target's client area when we call DFSrc_Create.
#0170    GetClientRect(hwndTarget, &rc);
#0171    ScreenToClient(hwndTarget, &ptMousePos);
#0172
#0173    // Create dropfile memory block and initialize it.
#0174  #ifdef UNICODE
#0175    hdrop = DFSrc_Create(&ptMousePos, !PtInRect(&rc, ptMousePos), TRUE);
#0176  #else
#0177    hdrop = DFSrc_Create(&ptMousePos, !PtInRect(&rc, ptMousePos), FALSE);
#0178  #endif
#0179
#0180    if (hdrop == NULL) {
#0181       adgMB(__TEXT("Insufficient memory to drop file(s)."));
#0182       return;
#0183    }
#0184
#0185    // Append each path name to the dropfile memory block.
#0186    for (nIndex = 0; nIndex < nNumFiles; nIndex++) {
#0187
#0188       ListBox_GetText(hwndLB, nIndex, szPathName);
#0189       hdropT = DFSrc_AppendPathname(hdrop, szPathName);
#0190
#0191       if (hdropT == NULL) {
#0192          adgMB(__TEXT("Insufficient memory to drop file(s)."));
#0193          hdrop = GlobalFree(hdrop);
#0194          break;                // Terminates the 'for' loop.
#0195       } else {
#0196          hdrop = hdropT;
#0197       }
#0198    }
#0199
#0200    if (hdrop != NULL) {
```

```
#0201
#0202       // All path names appended successfully, post the message to the
#0203       // dropfile target window.
#0204       FORWARD_WM_DROPFILES(hwndTarget, hdrop, PostMessage);
#0205
#0206       // Update the window's caption to reflect the number of selected files
#0207       // available for dropping.
#0208       SetDlgItemInt(hwnd, IDC_NUMFILES, 0, FALSE);
#0209       ListBox_ResetContent(hwndLB);
#0210
#0211       // NOTE: We should not call GlobalFree passing in hdrop.  The system
#0212       // will free it for us after it creates a copy of it in the target
#0213       // process's address space.  The target process is responsible for
#0214       // freeing it's copy of the hdrop by calling DragFinish.
#0215    }
#0216 }
#0217
#0218
#0219 ////////////////////////////////////////////////////////////////
#0220
#0221
#0222 BOOL WINAPI DFSrcDemo_Proc (HWND hwnd, UINT uMsg,
#0223    WPARAM wParam, LPARAM lParam) {
#0224
#0225    switch (uMsg) {
#0226       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG,  DFSrcDemo_OnInitDialog);
#0227       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,     DFSrcDemo_OnCommand);
#0228       adgHANDLE_DLGMSG(hwnd, WM_LBUTTONDOWN, DFSrcDemo_OnLButtonDown);
#0229       adgHANDLE_DLGMSG(hwnd, WM_MOUSEMOVE,   DFSrcDemo_OnMouseMove);
#0230       adgHANDLE_DLGMSG(hwnd, WM_LBUTTONUP,   DFSrcDemo_OnLButtonUp);
#0231    }
#0232
#0233    return(FALSE);              // We didn't process the message.
#0234 }
#0235
#0236
#0237 ////////////////////////////////////////////////////////////////
#0238
#0239
#0240 int WINAPI WinMain (HINSTANCE hinstExe,
#0241    HINSTANCE hinstPrev, LPSTR lpszCmdLine, int nCmdShow) {
#0242
#0243    adgWARNIFUNICODEUNDERWIN95();
#0244    adgVERIFY(-1 != DialogBox(hinstExe, MAKEINTRESOURCE(IDD_DFSRCDEMO),
#0245       NULL, DFSrcDemo_Proc));
#0246
```

```
#0247    return(0);
#0248 }
#0249
#0250
#0251 //////////////////////////// End of File ////////////////////////////
```

程式列表 7.5　DFSrc.C

```
#0001 /******************************************************************
#0002 Module name: DFSrc.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Drop file source library functions.
#0006 ******************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"         /* See Appendix A for details */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)    /* Single-line comment */
#0013 #include <ShlObj.h>             /* For DROPFILES structure */
#0014 #include <tchar.h>
#0015 #include <limits.h>
#0016 #include "DFSrc.h"
#0017
#0018
#0019 ////////////////////////////////////////////////////////////////////
#0020
#0021
#0022 // Macro to ease source code readability
#0023 #define IsAcceptingFiles(hwnd) \
#0024    ((GetWindowExStyle((hwnd)) & WS_EX_ACCEPTFILES) != 0)
#0025
#0026
#0027 ////////////////////////////////////////////////////////////////////
#0028
#0029
#0030 HWND WINAPI DFSrc_OkToDrop (PPOINT ppt) {
#0031
#0032    POINT ptMousePos;
#0033    HWND hwndTarget;
#0034
#0035    if (ppt == NULL) {
#0036
#0037       // The caller passes NULL. Assume that they want the mouse position
```

```
#0038       // at the time of the last retrieved window message.
#0039       ptMousePos.x = LOWORD(GetMessagePos());
#0040       ptMousePos.y = HIWORD(GetMessagePos());
#0041    }
#0042
#0043    hwndTarget = WindowFromPoint(ptMousePos);
#0044
#0045    // See if the target window or any of its parent windows are prepared to
#0046    // accept dropped files.
#0047    while (IsWindow(hwndTarget) && !IsAcceptingFiles(hwndTarget))
#0048       hwndTarget = GetParent(hwndTarget);
#0049
#0050    // If it's OK to drop files, return the HWND of the target window or NULL
#0051    // if we do not have a target window accepting files.
#0052    // the WS_EX_ACCEPTFILES style.
#0053    return((IsWindow(hwndTarget) && IsAcceptingFiles(hwndTarget))
#0054       ? hwndTarget : NULL);
#0055 }
#0056
#0057
#0058 ////////////////////////////////////////////////////////////////
#0059
#0060
#0061 HDROP WINAPI DFSrc_Create (PPOINT ppt, BOOL fNC, BOOL fWide) {
#0062
#0063    HDROP hdrop;
#0064    LPDROPFILES pDropFiles;
#0065
#0066    // Allocate dynamic memory for the DFSRC data structure and for the extra
#0067    // zero character identifying that there are no path names in the block yet.
#0068    hdrop = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT,
#0069       sizeof(DROPFILES) + (fWide ? sizeof(WCHAR) : sizeof(char)));
#0070
#0071    // If successful, lock block and initialize the data members.
#0072    if (hdrop != NULL) {
#0073       pDropFiles = (LPDROPFILES) GlobalLock(hdrop);
#0074       pDropFiles->pFiles = sizeof(*pDropFiles);
#0075       pDropFiles->pt = *ppt;
#0076       pDropFiles->fNC = fNC;
#0077       pDropFiles->fWide = fWide;
#0078       GlobalUnlock(hdrop);
#0079    }
#0080
#0081    return(hdrop);
#0082 }
#0083
```

```
#0084
#0085 ////////////////////////////////////////////////////////////////
#0086
#0087
#0088 HDROP WINAPI DFSrc_AppendPathname (HDROP hdrop, PVOID pvPathname) {
#0089
#0090    LPDROPFILES pDropFiles = (LPDROPFILES) GlobalLock(hdrop);
#0091
#0092    // Point to first path name in list.
#0093    PSTR  szPathA = (PSTR) pDropFiles + pDropFiles->pFiles;
#0094    PWSTR szPathW = (PWSTR) szPathA;
#0095    int nOffsetOfNewPathname, nPathSize;
#0096
#0097    if (pDropFiles->fWide) {
#0098
#0099       // Search for a path name where first character is a zero character.
#0100       while (*szPathW) {         // While the first character is nonzero
#0101          while (*szPathW)        // Find end of current path.
#0102             szPathW++;
#0103          szPathW++;             // Skip over the zero character.
#0104       }
#0105
#0106       // Get the offset from the beginning of the block
#0107       // where the new path name should go.
#0108       nOffsetOfNewPathname = ((PBYTE) szPathW - (PBYTE) pDropFiles);
#0109
#0110       // Get the number of bytes needed for the new path name,
#0111       // it's terminating zero character, and the zero-length
#0112       // path name that marks the end of the list of path names.
#0113       nPathSize = sizeof(WCHAR) * (wcslen(pvPathname) + 2);
#0114    } else {
#0115
#0116       // Search for a pathname where first character is a zero character.
#0117       while (*szPathA) {         // While the first character is nonzero
#0118          while (*szPathA)        // Find end of current path.
#0119             szPathA++;
#0120          szPathA++;             // Skip over the zero character.
#0121       }
#0122
#0123       // Get the offset from the beginning of the block
#0124       // where the new path name should go.
#0125       nOffsetOfNewPathname = ((PBYTE) szPathA - (PBYTE) pDropFiles);
#0126
#0127       // Get the number of bytes needed for the new path name,
#0128       // it's terminating zero character, and the zero-length
#0129       // path name that marks the end of the list of path names.
```

```
#0130       nPathSize = sizeof(char) * (strlen(pvPathname) + 2);
#0131    }
#0132
#0133    GlobalUnlock(hdrop);
#0134
#0135    // Increase block size to accommodate new path name.
#0136    hdrop = GlobalReAlloc(hdrop, nPathSize + nOffsetOfNewPathname,
#0137       GMEM_MOVEABLE | GMEM_ZEROINIT);
#0138
#0139    // If successful, append the path name to the end of the block.
#0140    if (hdrop != NULL) {
#0141       pDropFiles = (LPDROPFILES) GlobalLock(hdrop);
#0142
#0143    if (pDropFiles->fWide)
#0144       wcscpy((PWSTR) ((PSTR) pDropFiles + nOffsetOfNewPathname), pvPathname);
#0145    else
#0146       strcpy((PSTR)  ((PSTR) pDropFiles + nOffsetOfNewPathname), pvPathname);
#0147
#0148       GlobalUnlock(hdrop);
#0149    }
#0150
#0151    return(hdrop);              // Returns the new handle to the block.
#0152 }
#0153
#0154
#0155 ////////////////////////////////////////////////////////////////
#0156
#0157
#0158 LPCTSTR WINAPI MultiStrUtil_FindStr (LPCTSTR szStrAll, int nIndex,
#0159    PINT pnMax) {
#0160
#0161    int nNumStrs = 0;
#0162    LPCTSTR szStrSingle = szStrAll;
#0163
#0164    if (nIndex == -1)
#0165       nIndex = INT_MAX;
#0166
#0167    // Find the nIndex'th string.
#0168    // If the next string contains no characters, we have reached the end.
#0169    while ((*szStrSingle != 0) && (nNumStrs < nIndex)) {
#0170
#0171       // Find the end of a string.
#0172       szStrSingle = _tcschr(szStrSingle, 0) + 1;
#0173         nNumStrs++;
#0174    }
#0175
```

```
#0176    if (pnMax != NULL)
#0177       *pnMax = nNumStrs;
#0178
#0179    // If there are fewer strings than requested, return NULL.
#0180    return((nNumStrs < nIndex) ? NULL : szStrSingle);
#0181 }
#0182
#0183
#0184 ////////////////////////////////////////////////////////////////
#0185
#0186
#0187 BOOL WINAPI FileOpenUtil_AreMultipleFilesSelected (OPENFILENAME *pofn) {
#0188
#0189    // The offset of the file name is beyond the length of the
#0190    // first string. The first string contains only a path instead
#0191    // of a path and a file name.
#0192    return(_tcslen(pofn->lpstrFile) < pofn->nFileOffset);
#0193 }
#0194
#0195
#0196 ////////////////////////////////////////////////////////////////
#0197
#0198
#0199 int WINAPI FileOpenUtil_GetNumFiles (OPENFILENAME *pofn) {
#0200
#0201    int nNumFiles = 1;
#0202    if (FileOpenUtil_AreMultipleFilesSelected(pofn)) {
#0203       (void) MultiStrUtil_FindStr(pofn->lpstrFile, -1, &nNumFiles);
#0204
#0205       // Because the first string is a path, there is one less file than strings.
#0206       nNumFiles--;
#0207    }
#0208    return(nNumFiles);
#0209 }
#0210
#0211
#0212 ////////////////////////////////////////////////////////////////
#0213
#0214
#0215 int WINAPI FileOpenUtil_GetFile (OPENFILENAME *pofn,
#0216    int nIndex, LPTSTR szPathname) {
#0217
#0218    _tcscpy(szPathname, MultiStrUtil_FindStr(pofn->lpstrFile, 0, NULL));
#0219    if (FileOpenUtil_AreMultipleFilesSelected(pofn)) {
#0220       _tcscat(szPathname, __TEXT("\\"));
#0221       _tcscat(szPathname, MultiStrUtil_FindStr(pofn->lpstrFile,
```

```
#0222         nIndex + 1, NULL));
#0223    }
#0224    return(_tcslen(szPathname));   // Returns length of string.
#0225 }
#0226
#0227
#0228 ////////////////////////// End of File /////////////////////////////
```

程式列表 7.6　DFSrc.H

```
#0001 /*****************************************************************
#0002 Module name: DFSrc.h
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Dropfile source library functions
#0006 *****************************************************************/
#0007
#0008
#0009 HWND  WINAPI DFSrc_OkToDrop (PPOINT ppt);
#0010 HDROP WINAPI DFSrc_Create (PPOINT ppt, BOOL fNC, BOOL fWide);
#0011 HDROP WINAPI DFSrc_AppendPathname (HDROP hdrop, PVOID pvPathname);
#0012
#0013
#0014 ////////////////////////////////////////////////////////////////
#0015
#0016
#0017 LPCTSTR WINAPI MultiStrUtil_FindStr (LPCTSTR szStrAll, int nIndex,
#0018    PINT pnMax);
#0019
#0020 BOOL    WINAPI FileOpenUtil_AreMultipleFilesSelected (OPENFILENAME *pofn);
#0021
#0022 int     WINAPI FileOpenUtil_GetNumFiles (OPENFILENAME *pofn);
#0023
#0024 int     WINAPI FileOpenUtil_GetFile (OPENFILENAME *pofn, int nIndex,
#0025    LPTSTR szPathname);
#0026
#0027
#0028 ////////////////////////// End of File /////////////////////////////
```

程式列表 7.7　DFSrcDem.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
```

```
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #define APSTUDIO_HIDDEN_SYMBOLS
#0011  #include "windows.h"
#0012  #undef APSTUDIO_HIDDEN_SYMBOLS
#0013
#0014  ////////////////////////////////////////////////////////////////
#0015  #undef APSTUDIO_READONLY_SYMBOLS
#0016
#0017
#0018  ////////////////////////////////////////////////////////////////
#0019  //
#0020  // Icon
#0021  //
#0022
#0023  IDI_DFSRCDEMO          ICON    DISCARDABLE    "DFSrcDem.ico"
#0024
#0025  ////////////////////////////////////////////////////////////////
#0026  //
#0027  // Cursor
#0028  //
#0029
#0030  IDC_DROPOK            CURSOR  DISCARDABLE    "DropOK.cur"
#0031
#0032  ////////////////////////////////////////////////////////////////
#0033  //
#0034  // Dialog
#0035  //
#0036
#0037  IDD_DFSRCDEMO DIALOG DISCARDABLE  -32768, 5, 175, 103
#0038  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0039  CAPTION "Drop File Source Demo"
#0040  FONT 8, "MS Sans Serif"
#0041  BEGIN
#0042      DEFPUSHBUTTON   "&Select files...",IDC_SELECTFILES,4,4,65,14
#0043      LTEXT          "Files to drop:",IDC_STATIC,108,8,44,8
#0044      LTEXT          "0",IDC_NUMFILES,156,8,16,8
#0045      LISTBOX         IDC_PATHNAMELIST,4,22,168,78,NOT LBS_NOTIFY |
#0046                     LBS_NOINTEGRALHEIGHT | WS_VSCROLL | WS_HSCROLL |
#0047                     WS_TABSTOP
#0048  END
```

```
#0049
#0050
#0051  #ifdef APSTUDIO_INVOKED
#0052  ////////////////////////////////////////////////////////////////
#0053  //
#0054  // TEXTINCLUDE
#0055  //
#0056
#0057  1 TEXTINCLUDE DISCARDABLE
#0058  BEGIN
#0059      "resource.h\0"
#0060  END
#0061
#0062  2 TEXTINCLUDE DISCARDABLE
#0063  BEGIN
#0064      "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0065      "#include ""windows.h""\r\n"
#0066      "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0067      "\0"
#0068  END
#0069
#0070  3 TEXTINCLUDE DISCARDABLE
#0071  BEGIN
#0072      "\r\n"
#0073      "\0"
#0074  END
#0075
#0076  ////////////////////////////////////////////////////////////////
#0077  #endif    // APSTUDIO_INVOKED
#0078
#0079
#0080  #ifndef APSTUDIO_INVOKED
#0081  ////////////////////////////////////////////////////////////////
#0082  //
#0083  // Generated from the TEXTINCLUDE 3 resource.
#0084  //
#0085
#0086
#0087  ////////////////////////////////////////////////////////////////
#0088  #endif    // not APSTUDIO_INVOKED
```

程式列表 7.8　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
```

```
#0003  // Used by DFSrcDem.rc
#0004  //
#0005  #define IDC_DROPMANY                 102
#0006  #define IDC_DROPONE                  103
#0007  #define IDC_DROPOK                   103
#0008  #define IDI_DFSRCDEMO                104
#0009  #define IDD_DFSRCDEMO                105
#0010  #define IDC_NUMFILES                 1001
#0011  #define IDC_SELECTFILES               1002
#0012  #define IDC_PATHNAMELIST              1003
#0013  #define IDC_STATIC                   -1
#0014
#0015  // Next default values for new objects
#0016  //
#0017  #ifdef APSTUDIO_INVOKED
#0018  #ifndef APSTUDIO_READONLY_SYMBOLS
#0019  #define _APS_NO_MFC                  1
#0020  #define _APS_NEXT_RESOURCE_VALUE       106
#0021  #define _APS_NEXT_COMMAND_VALUE        40001
#0022  #define _APS_NEXT_CONTROL_VALUE        1004
#0023  #define _APS_NEXT_SYMED_VALUE          101
#0024  #endif
#0025  #endif
```

拖放（Drag and Drop）的其它用途

Drag and Drop（拖放）可以讓使用者以比較高的工作效率來使用我們的程式。舉個例，當我們使用 Word for Windows 和 Visual C++ 時，它們兩個程式都允許使用者標示一塊文字區段，然後拖放到文件中的另一個新位置上。這大大改變了我們對於文字處理的傳統觀念。同樣地，Excel 也允許使用者在試算表中選取某個範圍的儲存格（cells），然後利用拖放法將其內容搬移或複製到其它位置上。在你的程式中加入這項功能，其實非常容易，你只需提供先前討論過的資料結構與函式即可。如果你只是想在你自己的程式中使用拖放功能，那就根本毋需擔心程式間溝通的協定問題，也就是說你不需擔心它會與未來的 Windows 版本發生衝突。以下是你必須採取的行動的大略綱要：

- 自行定義一些訊息，像是 WM_DROPTEXT 等等，取代 WM_DROPFILES。
- 自行設計一個類似 DFSrc_Create 的函式，擁有像我的 DFSrc_Create 函式同樣的參數，再加上任何你希望的額外參數。舉個例，如果你要搬移一段文字，你

可能需要知道檔案中的文字起始位址，以及需要搬動多少個字元。對於試算表來說，你可能需要知道被指定的儲存格的範圍。

- 依你的特定情況來處理 WM_LBUTTONDOWN、WM_MOUSEMOVE 和 WM_LBUTTONUP 訊息。舉個例，當使用者將滑鼠移動經過你的視窗的不同區域時，你可以依自己的意思改變滑鼠游標的形狀。
- 一旦你收到了自己定義的訊息，表示物件已被丟入，此時可做一些你想做的動作，但最後別忘了釋放記憶體區塊。

第 8 章

按鍵的處理

在這一章中，我們進一步看看 Windows 如何處理按鍵（keystroke）動作。乍看之下，這個處理程序似乎相當簡單，但是 Windows 最初的設計是以滑鼠來操控，鍵盤支援能力是事後添加上去的。不幸的是，Microsoft Windows 的開發小組沒有在應用軟體開發人員（例如我們）之前將許多醜陋的鍵盤痕跡隱藏起來。

鍵盤與掃瞄碼

讓我們從 PC 的鍵盤開始討論起。實際上鍵盤是一個很複雜的設備，它有自己的微處理器（microprocessor）。它有許多特性，像是自動重複（auto-repeat）、Caps Lock 和 Num Lock 和 Scroll Lock 按鍵指示燈、功能鍵（function key）、用以改變下一個按鍵動作之意義的 Shift 鍵和 Alt 鍵和 Ctrl 鍵、以及一些執行相同工作的（不同的）鍵。此外，鍵盤的配置（layout）每一個國家皆不相同，單單在美國，就有 81 鍵、101 鍵、QWERTY 鍵盤和 Dvorak 鍵盤等等。

考慮到這麼多不同型式的鍵盤，並且想到 Windows 是所謂的「設備獨立」，很明顯地，必須付出很大的代價才能達成對鍵盤的支援能力。

當你壓下鍵盤上的一個按鍵，鍵盤的微處理器會測得一個被壓下的實際鍵（physical key），並產生一個硬體中斷，由已安裝妥當的鍵盤驅動程式處理。當產生硬體中斷的時

候，鍵盤會傳遞一個所謂的掃瞄碼（scan code）給系統的中斷服務程式（interrupt service routine，ISR）。掃瞄碼是一個 8 位元的二進位數字，用來確認鍵盤上的實際按鍵。如果按鍵被壓下，掃瞄碼的最高位元（位元 7）將被設爲 0；如果按鍵被放開，則該位元會被設立爲 1。

鍵盤製造業者設計鍵盤時，會制訂鍵盤上的哪一個實際按鍵產生出什麼樣的掃描碼。舉個例子，如果業者指定 0x04 值給【3】這個鍵，就表示每當使用者按下【3】鍵，鍵盤就會產生一個硬體中斷並將 0x04 掃瞄碼傳給鍵盤的驅動程式。由於一個掃瞄碼代表一個實際按鍵，所以它沒辦法告訴驅動程式說使用者按的是【3】鍵或是【#】符號。

掃瞄碼與鍵盤硬體密不可分，而不同的鍵盤需要不同的驅動程式。幸運的是，大部份鍵盤製造廠商多少做了點標準化動作，使鍵盤驅動程式傾向於語文導向（language-specific），而不是廠商導向（manufacturer-specific）。換句話說，對於一個葡萄牙語系鍵盤，你需要一個驅動程式，對於一個英語系鍵盤，你需要另一個驅動程式；然而面對不同的英語系鍵盤，你通常不需要不同的驅動程式。

鍵盤驅動程式與虛擬碼（virtual key codes）

Windows 試圖將所有的硬體設備抽象化，也就是說 Win32 程式員毋需擔心各種鍵盤上有不同的按鍵掃瞄碼，也不需爲了特定的鍵盤，撰寫特定的程式碼。爲了抽象化所有鍵盤設備，微軟公司必須提出一套標準的編碼方法，用以辨識各種不同鍵盤的按鍵。這種編碼方法稱爲虛擬鍵碼（virtual-key code），已經被正式記載並公開，所有的 Win32 程式員都應該使用它。你可以在 ***Win32 Programmer's reference*** 和 WinUser.H 表頭檔中找到虛擬鍵碼表。Windows 的鍵盤驅動程式會將鍵盤的掃瞄碼轉換爲虛擬鍵碼。

不幸的是，許多定義在 WinUser.H 表頭檔中的虛擬鍵碼的命名並不很明確。舉個例，Alt 鍵的識別字爲 VK_MENU（這是因爲在 Windows 程式中壓下並放開 Alt 鍵，會使視窗的選單起作用），Page Up 和 Page Down 鍵則分別被定義爲 VK_PRIOR 和 VK_NEXT。此外，有一些虛擬鍵碼並不代表鍵盤上的任何實際按鍵，像 VK_LBUTTON 和

VK_RBUTTON 虛擬鍵碼所表示的是滑鼠左鍵被按下或右鍵被按下。（無論使用者是否已告訴 Windows 要交換滑鼠的左右按鍵，爲了避免困擾，我總是說上述兩個虛擬鍵碼的實際意義分別代表滑鼠的左鍵（LBUTTON）被按下和右鍵（RBUTTON）被按下）

從鍵盤驅動程式到系統輸入佇列（System Input Queue）

鍵盤驅動程式將一個鍵盤的掃瞄碼轉換成一個虛擬鍵碼之後，驅動程式還必須將按鍵事件（keystroke event）附加到系統輸入佇列中。Windows 95 和 Windows NT 都只有一個系統輸入佇列，用來存放鍵盤、滑鼠和筆式輸入訊息。鍵盤驅動程式會呼叫 keybd_event 函式（內含在 User32.DLL 中）將按鍵事件加到系統輸入佇列之中：

```
VOID keybd_event(BYTE bVirtualKey, BYTE bScanCode,
   DWORD dwFlags, DWORD dwExtraInfo);
```

其中的 bVirtualKey 參數爲一個虛擬鍵碼（由鍵盤驅動程式決定其值），bScanCode 參數是硬體掃瞄碼（由鍵盤傳遞過來），dwFlags 參數爲一個旗標組合，詳列於表 **8.1**：

表 8.1　keybd_event 函式的 dwFlags 參數可能用到的旗標值。

旗標識別字	意義
KEYEVENTF_EXTENDEDKEY	此旗標如果設立，表示掃瞄碼之前會先有一個前置位元組，其值爲 0xE0 (224)。
KEYEVENTF_KEYUP	此旗標如果設立，表示按鍵正被放開。此旗標如果未被設立，表示按鍵正被壓下。

dwExtraInfo 參數用來指定一些附加的與設備有關的資料。這些資料不常被使用，所以此參數通常被設爲 0。GetMessage 和 PeekMessage 函式取得硬體訊息之後，執行緒可以利用 GetMessageExtraInfo 函式取得這個附加資料的內容：

```
LONG GetMessageExtraInfo(VOID);
```

同樣地，滑鼠驅動程式也會呼叫 mouse_event 函式，將訊息加入系統輸入佇列之中：

```
void mouse_event(
   DWORD dwFlags,     // Flags specifying various motion/click variants
   DWORD dx,          // Horizontal mouse position or position change
   DWORD dy,          // Vertical mouse position or position change
   DWORD cButtons,    // Unused, reserved for future use, set to zero
   DWORD dwExtraInfo  // 32 bits of application-defined information
   );
```

如果你想強行加入一組按鍵訊息到系統輸入佇列之中，使用 keybd_event 函式將是你最好的決定。Win32 技術文件中也曾指出，你可以模擬「使用者按下 Print Screen 鍵」，將螢幕或「作用中（active）視窗」的畫面拷貝到剪貼簿（Clipboard）中：

```
// Capture the full screen to the Clipboard.
Keybd_event(VK_SNAPSHOT, 0, 0, 0);   // bScanCode = 0

// Capture the active window to the Clipboard.
Keybd_event(VK_SNAPSHOT, 1, 0, 0);   // bScanCode = 1
```

按鍵狀態陣列（Key State Arrays）

系統內部有一個執行緒，稱爲 Raw Input Thread（RIT），它大部份時間都處於睡眠狀態，等待著訊息在系統輸入佇列中出現。一旦 keybd_event 函式將訊息加入系統輸入佇列之中，RIT 會醒來並分析訊息，然後更新系統的非同步按鍵狀態陣列（asynchronous key state array），並將適當的按鍵訊息放到「焦點視窗所隸屬之執行緒」的訊息佇列中。

系統的「非同步按鍵狀態陣列」是一個內部陣列，由 256 筆記錄（entry）所組成，每一筆記錄代表一個虛擬鍵碼。當按鍵訊息從系統輸入佇列中被取走，RIT 會設定或清除陣列中的位元，使陣列總是能夠反應鍵盤的目前狀態 -- 也就是哪一個鍵被按下，哪一個鍵被放開。

執行緒可以呼叫 GetAsyncKeyState 函式來得知鍵盤上目前的、即時的（real-time）按鍵狀態：

```
SHORT GetAsyncKeyState(int nVirtKey);
```

這個函式只接受一個虛擬鍵碼做爲參數，傳回一個 short 整數。如果整數的最高位元（high-bit）被設爲 1 的話，則虛擬鍵碼所定義的那個實際鍵目前是被壓下。如果最高位元被清除爲 0，則該實際鍵目前是被放開的。傳回值的最低位元（low bit）表示自從先前呼叫 GetAsyncKeyState 函式以後，是否按鍵狀態有所改變：1 表示已有改變，0 則表示未曾改變。關於 GetAsyncKeyState 函式，有一件重要的事必須注意：如果呼叫此函式的執行緒並未擁有焦點視窗視窗的話，GetAsyncKeyState 會傳回 0。微軟之所以會這樣設計，是爲了防止按鍵訊息被另一個執行緒的視窗處理。

RIT 將系統的非同步按鍵狀態陣列更新完畢後，它會將按鍵訊息派送到合適的視窗。RIT 首先確認焦點視窗隸屬於哪一個執行緒，然後把按鍵訊息附加到這個執行緒的虛擬化輸入佇列（virtualized input queue）中。每一個執行緒都有它自己的一個虛擬化輸入佇列，這個佇列就像系統輸入佇列一樣，只不過其中僅包含「預定要給此執行緒所產生之視窗」的硬體訊息。

除了虛擬化輸入佇列外，每一個執行緒還有一個自己的同步按鍵狀態陣列（synchronous key state array）。當執行緒呼叫 GetMessage 或 PeekMessage，這些函式會先檢查看看是否有一個按鍵訊息正從執行緒的虛擬化輸入佇列中被取走。如果真是這樣，那麼在傳回這個訊息之前，這兩個函式會先更新執行緒的同步按鍵狀態陣列。

執行緒的「同步按鍵狀態陣列」所表示的，是當「最近一個訊息從執行緒的虛擬化輸入佇列中被取走」時，鍵盤上所有按鍵的狀態。執行緒可以呼叫 GetKeyState 來存取其「同步按鍵狀態陣列」：

```
SHORT GetKeyState(int nVirtKey);
```

和 GetAsyncKeyState 函式一樣，它也是以一個虛擬鍵碼作爲唯一參數，並傳回一個 short 整數。事實上，這個整數的最高位元和 GetAsyncKeyState 函式傳回值的最高位元一樣，

都是用來表示按鍵是否被壓下。然而，GetKeyState 函式傳回值的最低位元則是用來表示這個鍵是否被切換。這個位元只有在檢驗 Caps Lock（VK_CAPITAL）、Scroll Lock（VK_SCROLL）或 Num Lock（VK_NUMLOCK）時才有意義。

回想一下，執行緒的「同步按鍵狀態陣列」是表示當鍵盤訊息從執行緒的虛擬化輸入佇列中被取走時的按鍵狀態。這意味著，當一個訊息被處理時，你可以告知鍵盤上所有按鍵的狀態。換句話說，當你正在處理一個訊息的時候，如果使用者敲擊鍵盤上的按鍵，你的執行緒的「同步按鍵狀態陣列」不受影響。

這裡有一個例子，說明爲什麼你會想使用 GetKeyState 函式。當使用者選取了一個選單項目，你的視窗函式便會收到一個 WM_COMMAND 訊息。然而在你收到 WM_COMMAND 訊息的同時，如果 Shift 鍵處於被按下狀態，你或許應該去執行另一個不同的動作。爲實現這個願望，程式寫法如下：

```
void Cls_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {

   switch (id) {
      case IDC_MENUITEM:
         if (GetKeyState(VK_SHIFT) & 0x8000) {
            // The Shift Key down.
         } else {
            // The Shift Key Up.
         }
         break;
      .
      .
      .
}
```

如果我使用 GetAsyncKeyState 函式取代 GetKeyState 函式，程式可能不會正常運作。就這個例子來說，當執行緒開始處理 WM_COMMAND 訊息，就在執行緒呼叫 GetAsyncKeyState 函式之前，使用者可能放開了 Shift 按鍵。而 GetAsyncKeyState 函式傳回的是當下（目前）鍵盤上的按鍵狀態，不是最近一個訊息被取得時的按鍵狀態。所以在先前的程式片段中，如果使用 GetAsyncKeyState 函式，結果將不是你所預期的。

現在你知道何時使用 GetKeyState 函式了！我要另舉例子來說明何時使用 GetAsyncKeyState 函式。讓我們假設，當使用者壓下滑鼠按鍵，你希望進入一個迴路做些事情，直到滑鼠按鍵被放開爲止：

```
void Cls_OnLButtonDown (HWND hwnd, BOOL fDoubleClick,
   int x, int y, UNIT keyFlags) {

   // Determine which mouse button to really check
   int vkMouseBtn = GetSystemMetrics(SM_SWAPBUTTON) ?
      VK_RBUTTON : VK_LBUTTON;

   while (GetAsyncKeyState(vkMouseBtn) & 0x8000) {
      // Do loop processing.
   }
}
```

每當滑鼠的主鍵（primary button）被按下，便產生 WM_LBUTTONDOWN 訊息。通常主鍵是指左鍵，然而如果使用者曾經利用【Mouse Property】對話盒交換了滑鼠左右按鍵，那麼當右鍵按下時會產生 WM_LBUTTONDOWN 訊息。我曾說過，無論使用者是否經由【Mouse Property】對話盒交換過滑鼠的左右按鍵，如果你想要檢查滑鼠左鍵的實際狀態，都應該將 VK_LBUTTON 傳給 GetAsyncKeyState 函式或 GetKeyState 函式。當先前的程式片段呼叫 GetAsyncKeyState 函式之前，必須先知道到底真正要檢查的滑鼠按鍵是哪一個，這可以利用 GetSystemMetrics 函式（並傳入 SM_SWAPBUTTON 識別字）來達成。如果使用者已交換過滑鼠的左右按鍵，GetSystemMetrics 函式會傳回 TRUE，否則傳回 FALSE。這將使 vkMouseBtn 變數會得到一個正確的值，不是 VK_LBUTTON 就是 VK_RBUTTON。

在 vkMouseBtn 變數設定過後，程式反覆呼叫 GetAsyncKeyState 函式來檢查是否滑鼠的主鍵已被放開。如果我們在這裡以 GetKeyState 函式取代 GetAsyncKeyState 函式的話，我的執行緒會被滯留在一個無限迴路中！因爲當 WM_LBUTTONDOWN 訊息被派送（dispatched）出去時，無論滑鼠按鍵是否被按下，GetKeyState 函式的傳回值都不會改變。由於 WM_LBUTTONDOWN 訊息被處理時滑鼠主鍵總是處於被按下狀態，所以 GetKeyState 函式傳回值的最高位元總是會被設立（爲 1）。

現在，我們再來看看兩個可以讓執行緒自行操控其「同步按鍵狀態陣列」的函式。首先我們要看的是 GetKeyboardState 函式：

```
BOOL GetKeyboardState(PBYTE pbKeyState);
```

這個函式的唯一參數是一個指向由 256 個位元組所組成的陣列的指標。呼叫後，這個陣列將內含目前執行緒的「同步按鍵狀態陣列」的一份拷貝複本。就像 GetKeyState 函式一樣，陣列中每一個位元組的最高位元表示「最近一次從執行緒的佇列中取得訊息時，按鍵是否被按下」，而最低位元表示「如果按鍵是個切換（toggle）鍵，是否它被切換了」。執行緒可以利用 GetKeyboardState 函式來取得數個按鍵的狀態。想要檢查一個虛擬鍵的狀態，你只需將虛擬鍵碼當做索引值，查閱這個陣列即可。舉個例子，如果你要檢查看看空白鍵（Spacebar）是否被按下，就這麼做：

```
BYTE pbKeyState[256];
BOOL fSpaceIsDown;
GetKeyboardState(pbKeyState);
fSpaceIsDown = (pbKeyState[VK_SPACE] & 0x80) != 0;
```

當一個訊息正被處理時，執行緒可以呼叫 GetKeyboardState 函式，取得目前的「同步按鍵狀態陣列」的一份「快照」，將它儲存到一個全域陣列或靜態陣列之中，然後在未來處理訊息時，使用這個陣列。這種鍵盤「快照」並不常被使用，但常和 SetKeyboardState 函式一起合作：

```
BOOL SetKeyboardState(PBYTE pbKeyState);
```

SetKeyboardState 函式允許執行緒改變其「同步鍵盤狀態陣列」。它的唯一參數是一個由 256 個位元組所構成的陣列的位址，陣列中記錄著每一個你所希望的虛擬鍵狀態。下面的程式片段示範如何讓執行緒認爲空白鍵（Spacebar）已被按下：

```
BYTE pbKeyState[256];
GetKeyboardState(&pbKeyState);
pbKeyState[VK_SPACE] |= 0x80;
SetKeyboardState(&pbKeyState);
```

爲了改變單一按鍵狀態，你必須先呼叫 GetKeyboardState 函式來取得目前的同步按鍵狀態陣列，此函式會將目前的同步按鍵狀態陣列中的內容填到你所指定的一個 256 位元組所組成的陣列中。這是必要的，因爲 SetKeyboardState 函式會一次改變所有的虛擬鍵，它沒有辦法改變單獨一個虛擬鍵。

SetKeyboardState 函式可改變所有虛擬鍵的狀態；然而，如果你以此改變 Caps Lock、Num Lock、或 Scroll Lock 等等 on-off 切換鍵的狀態，使用者鍵盤上的 LED 燈並不會轉亮或轉滅。如果你希望更改這些鍵的狀態並讓鍵盤上的 LED 燈有適當反應的話，你就必須搭配使用 GetKeyState、keybd_event 和 MapVirtualKey 函式（稍後討論），像這樣：

```
void ToggleLockKey (BYTE bVirtKey ,BOOL fLocked) {

   // NOTE: bVirtKey must be VK_CAPITAL, VK_NUMLOCK, or VK_SCROLL.

   // Get the current state of the toggle key.
   BOOL fKeyLocked = GetKeyState(bVirtKey) & 1;
   if (fKeyLocked == fLocked) {

      // The key is in the requested state; do nothing.
   } else {

      // The Key is not in the requested state.
      // toggle it by pressing and releasing the key.
      Keybd_event(bVirtKey, MapVirtualKey(bVirtKey, 0), 0, 0);
      Keybd_event(bVirtKey, MapVirtualKey(bVirtKey, 0), KEYEVENTF_KEYUP, 0);
   }
}
```

上述函式中的 bVirtKey 參數表示你想要切換哪一個 on-off 切換鍵（Caps Lock 的虛擬鍵碼爲 VK_CAPITAL，Num Lock 的虛擬鍵碼爲 VK_NUMLOCK，Scroll Lock 的虛擬鍵碼爲 VK_SCROLL）；fLock 參數表示你希望將按鍵設爲何種狀態。這個函式首先呼叫 GetKeyState，取得我們想要處理的按鍵的目前狀態。如果位元 0 是 on，表示按鍵目前已被切換。如果按鍵的目前狀態和我們所要求的 fLock 參數一致，那就什麼事情都不做。如果不一致的話，就呼叫 keybd_event 兩次：第一次模擬按鍵的壓下動作，第二次模擬按鍵的放開動作。當然，這個動作會改變（切換）按鍵的狀態。

按鍵訊息（Keystroke Messages）

就像每一個 Win32 程式員所知道的，執行緒可以使用 GetMessage 函式或 PeekMessage 函式，從它的虛擬輸入佇列中取得 WM_(SYS)KEYDOWN 和 WM_(SYS)KEYUP 訊息。一旦訊息被取得，程式通常會呼叫 DispatchMessage 函式將訊息分送給視窗。當視窗函式收到了 WM_(SYS)KEYDOWN 或 WM_(SYS)KEYUP 訊息，其 wParam 參數爲虛擬鍵碼，而 lParam 參數爲一連串的旗標位元，如表 **8.2** 所示。

表 8.2 WM_(SYS)KEY* 訊息的 lParam 參數意義

位元	意義
0-15	這個值表示按鍵被重複按下（由於使用者一直按著不放）的次數，。
16-23	OEM 掃瞄碼。
24	是否爲擴充鍵，例如功能鍵或鍵盤右側的數字鍵。如果是擴充鍵，則爲 1；否則爲 0。
25-26	未使用。
27-28	由 Windows 內部使用。
29	所謂的 context code。如果按鍵被按下時 Alt 鍵也被按下，則此位元爲 1；否則爲 0。
30	先前的按鍵狀態。如果在訊息被送出之前，按鍵是被壓下的，則此位元爲 1；否則爲 0。
31	表示 "key-transmition" 狀態。如果按鍵正被放開，則此位元爲 1；如果按鍵正被壓下，則此位元爲 0。

要取得 lParam 參數的較高字組（high word）比較簡單，因爲 WinUser.H 表頭檔中定義有一些識別字，如表 **8.3** 所示。

表 8.3　可取用鍵盤訊息旗標值的識別字

識別字	數值
KF_EXTENDED	0x0100
KF_DLGMODE	0x0800
KF_MENUMODE	0x1000
KF_ALTDOWN	0x2000
KF_REPEAT	0x4000
KF_UP	0x8000

當你使用這些識別字時，請確定你總是把它拿來和 lParam 參數的 HIWORD 做“AND”運算，像這樣：

```
void Cls_OnKey (HWND hwnd, UNIT vk, BOOL fDown, int cRepeat, UNIT flags) {

   BOOL fIsExtendedKey;

   // The following line of code is correct.
   fIsExtendedKey = HIWORD(lParam) & KF_EXTENDED;

   // The following line of code is incorrect.
   fIsExtendedKey = lParam & KF_EXTENDED;
   .
   .
   .
}
```

由於上述識別字被定義在 WinUser.H 表頭檔中，,微軟一不小心公開了內部使用的兩個位元，位元 27 和 28，識別字分別爲 KF_DLGMODE 和 KF_MENUMODE。我的經驗證顯示，這些位元是用來告訴 Windows 說，當按鍵正被處理時，Windows 處於何種模式之下。舉個例，如果當時選單（menu）被拉下，KF_MENUMODE 位元就爲 1。

由於所有關於按鍵被壓下或被放開的訊息，只在 wParam 參數中記錄著虛擬鍵碼，所以一個程式很難處理按鍵。假定你爲一個視窗寫了個視窗函式，其行爲如同 edit 控制元件

一樣。每當你收到一個 WM_KEYDOWN 訊息，便將視窗中的字元加到緩衝區中；你必須檢查 Shift、Ctrl、Alt、Caps Lock 和 Num Lock 鍵的狀態，才能將虛擬鍵碼轉換爲一個實際字元。如果使用者按下 A 鍵，WM_KEYDOWN 訊息的 wParam 參數將是 VK_A，此時程式並不知道使用者按的是大寫 A，還是小寫的 a，還是 Ctrl+A。由於這些處理對於一個視窗函式來說太過困難，所以你應該使用 TranslateMessage 函式。

TranslateMessage 函式通常是在一個執行緒的訊息迴路中被呼叫 -- 在呼叫 GetMessage 或 PeekMessage 函式之後，並且在呼叫 DispatchMessage 函式之前。TranslateMessage 函式首先檢查所取得的訊息類型；如果訊息與鍵盤無關，它傳回 FALSE。如果訊息與鍵盤有關，則 TranslateMessage 函式會檢查執行緒的「同步按鍵狀態陣列」，並將虛擬鍵碼轉換爲一個字元（character）。然後，它會放置一個 WM_CHAR 訊息（其 wParam 參數內含轉換後所得的字元）到執行緒的訊息佇列中（而不是執行緒的「虛擬化輸入佇列」中）。這樣就能保證下次執行緒呼叫 GetMessage 或 PeekMessage 函式時，會取得 WM_CHAR 訊息。

所以，視窗函式處理的對象應該是 WM_CHAR 訊息，而不是 WM_KEYDOWN 訊息。對一個類似 edit 控制元件的視窗類別來說，其視窗函式應該是將 WM_CHAR 訊息的 wParam 參數所代表的字元安插到其字元緩衝區中。然而，還有許多其它按鍵，例如 "scroll" 鍵（上下左右方向鍵、Page Down 和 Page Up 鍵等等），並不會被轉換爲任何字元，因爲 TranslateMessage 函式並不保證對每一個虛擬鍵碼都產生出對應的 WM_CHAR 訊息。爲了這個原因，那些特別關心使用者鍵盤動作的視窗函式，必須同時處理 WM_KEYDOWN 和 WM_CHAR 訊息。

TranslateMessage 函式的內部是利用 ToAscii 函式[1] 將一個虛擬鍵碼轉換爲一個字元[2]：

---

[1] ToAscii 函式會將一個虛擬鍵碼轉換爲對應的字元（如果存在的話）。其結果可能是（也可能不是）一個 ASCII 字元。

[2] 另有一個 ToUnicode 函式，和 ToAscii 函式的工作沒什麼兩樣，只不過它是將一個虛擬鍵碼轉換爲一個 Unicode 字元。Windows 95 並未支援此函式，Windows NT 有支援。

```
int ToAscii(UNIT uVirtKey, UNIT uScanCode, PBYTE pbKeyState,
   PWORD pwTransKey, UNIT fuState);
```

前兩個參數是虛擬鍵碼（wParam）和掃瞄碼（lParam 參數的位元 16~23）。第三個參數 pbKeyState，是一個 256 位元組陣列的位址，其中記錄著當此函式提取最近一個按鍵訊息時，所有鍵盤的狀態。TranslateMessage 函式會以「執行緒同步按鍵狀態陣列的位址」當作第三個參數傳入。第四個參數 pwTransKey，是一個 WORD 指標，用來接收一個（或多個）被轉換字元。如果超過一個以上的字元被產生出來，第一個字元通常會是一個重音符號（accent）或母音變化記號（umlaut）以爲識別，促使 TranslateMessage 函式將 WM_(SYS)DEADCHAR 訊息放到執行緒的訊息佇列中[3]。任何其它字元則是以 WM_(SYS)CHAR 訊息被放置到訊息佇列中。最後一個參數 fuState，如果 1 表示選單（menu）目前作用中（active）。表 **8.4** 列出 ToAscii 函式的傳回值。

表 8.3　ToAscii 函式的傳回值

傳回值	意義
負數	無作用鍵（dead key）。
0	虛擬鍵碼不需轉換（因目前鍵盤狀態的緣故）。
1	複製一個字元到緩衝區中。
2	複製兩個字元到緩衝區中。當 dead key 不能被轉換時，這兩個字元通常指的是一個重音符號（accent）和一個 dead key 字元。

[3] 有些字元需由使用者壓下鍵盤上的兩個按鍵才可形成。舉個例，如果使用者想產生 umlant-O 字元(Ö)，首先必須壓下一個 umlaut（double-dot）鍵，然後再按下 O 鍵。當使用者壓下第一個鍵，系統會產生 WM_DEADCHAR 訊息，接著再按下 O 鍵時，系統會產生 WM_CHAR 訊息，其 wParam 參數爲完整的 umlaut-O 字元。所以大部份視窗函式皆可以忽略 WM_DEADCHAR 訊息，因爲 WM_CHAR 訊息已包含了完整的字元（可立即輸出或直接使用）。文字編輯程式則是藉由處理 WM_DEADCHAR 訊息來將母音變化記號顯示在與字元同一個位置的螢幕上。

如你所見，Windows 系統之中發生許多轉換動作。當你的視窗函式正等待接收適當訊息的時候，大部份轉換動作發生在各個 Win32 函式中。如果你想要自行轉換，有一些函式可以幫你完成心願，例如 VkKeyScan。

VkKeyScan 函式接受一個字元做爲其唯一參數，並傳回對應的虛擬鍵碼和 shift 狀態：

```
short VkKeyScan(TCHAR uChar);
```

如果這個函式不能轉換參數所指定的字元，就傳回 -1；否則，其低位元組記錄著虛擬鍵碼，高位元組記錄著 shift 狀態旗標。表 **8.5** 列出各種可能的 shift 狀態旗標。

表 8.5 VkKetScan 函式所傳回的 shift 狀態旗標。

數值	意義
1	The Shift key is required。
2	The Ctrl key is required 。
4	The Alt key is required 。

另一個幫助你做轉換的函式是 MapVirtualKey，它能使用於數種不同的轉換型態：

```
UNIT MapVirtualKey(UNIT uKeyCode, UNIT fuMapType);
```

這個函式會依照 fuMapType 參數值的不同來做不同類型的轉換。表 **8.6** 列出 fuMapType 參數可能會出現的值。

表 8.6　MapVirtualKey 函式的使用方法。

fuMapType	uKeyCode	傳回值
0	虛擬鍵碼	掃瞄碼
1	掃瞄碼	虛擬鍵碼
2	虛擬鍵碼	Unshifted 字元
3	掃瞄碼	虛擬鍵碼 *

* 將 fuMapType 參數設爲 3 和設爲 1 幾乎是一樣的。差別在於，當你將它設爲 3，MapVirtualKey 函式會區分左邊的 shift 鍵和右邊的 shift 鍵；如果設爲 1，則左右 shift 鍵沒有差別。

System Key 的區別

到目前爲止我未曾提過 system key 和一般按鍵之間有什麼區別。事實上，截至目前所討論的主題爲止，它們之間並沒有什麼區別。就 Windows 而言，所謂 system key 和 system character 是「當 Alt 鍵被按下時，所產生的 key 和 character」。很簡單吧！如果 Ctrl 鍵和 Alt 鍵相繼被按下，則這個按鍵和字元不能算是一個 system key。表 **8.7** 顯示了一些按鍵組合，以及它們所構成的 system key。

表 8.7　一些按鍵組合，及它們所產生的 WM_SYSKEY* 訊息

次序	壓下 / 放開	按鍵	產生的訊息
1	壓下	Alt	WM_SYSKEYDOWN
2	放開	Alt	WM_SYSKEYUP
3	壓下	Alt	WM_SYSKEYDOWN
4	壓下	Ctrl	WM_KEYDOWN
5	壓下	A	WM_KEYDOWN
6	放開	A	WM_KEYUP
7	放開	Ctrl	WM_KEYUP
8	壓下	A	WM_SYSKEYDOWN

次序	壓下 / 放開	按鍵	產生的訊息
9	放開	A	WM_SYSKEYUP
10	壓下	A	WM_SYSKEYDOWN
11	壓下	Ctrl	WM_KEYDOWN
12	放開	A	WM_KEYUP
13	放開	Ctrl	WM_KEYUP
14	放開	Alt	WM_KEYUP

（侯俊傑註：這個表格的意思是，如果你按照表中的 1 2 3 4…次序操作，系統會產生出最右側所列的訊息，其中有的是 system key，有的是一般的鍵盤訊息，視按鍵組合的不同而不同）

表 **8.7** 的項目 1~2 顯示，當 Alt 鍵被壓下和放開時會發生什麼事情。這種情況下，Windows 會產生 system key 訊息，其 wParam 參數爲 VK_MENU。

項目 3~5 之中，請注意，當 Alt 鍵被壓下，會產生 WM_SYSKEYDOWN 訊息，之後再壓下 Ctrl 鍵，Windows 會產生 WM_KEYDOWN 訊息（雖然現在 Alt 鍵並未被放開）。

項目 6~7 顯示 A 鍵和 Ctrl 鍵被放開的結果，並沒有什麼特別的。項目 8~9 顯示 A 鍵被壓下和放開，此時 Alt 鍵仍未被放開而 Ctrl 鍵已被放開，所以會產生 system key 訊息。

現在讓我們看看項目 10~13。請注意，其中所得到的按鍵訊息不需要對稱。這意思是說，當 Windows 送出 WM_SYSKEYDOWN 之後，並不一定要再送出一個與其相配的 WM_SYSKEYUP。

項目 14 相當古怪。此時除了 Alt 鍵之外沒有其它按鍵被壓下。當 Alt 鍵被放開，Windows 產生的竟然是一個 WM_KEYUP，而不是一個 WM_SYSKEYUP。這與表 **8.7** 的第個項目相互違背。Windows 一定是在這裡偷偷做了某件事情！啊，Windows 的秘密是，如果在放開 Alt 鍵之前，有任何其它的鍵盤事件發生，那麼當放開 Alt 鍵，Windows 會產生

WM_KEYUP 訊息，而不是 WM_SYSKEYUP 訊息。

對其他程式模擬按鍵動作

有許多程式希望能夠強制送出某些按鍵訊息到其它程式的視窗之中。舉個例，多年前我購得一套 Windows 版的辭典程式。這個軟體可巧妙地與任何文字編輯軟體合作。當它執行起來，首先掛上一個 keyboard hook（請參閱第 6 章）。如果想知道某個字的同義字，我只需在我的文書編輯軟體的文件內標示出一個字，然後壓下 Ctrl+Alt+T 鍵。被辭典軟體掛上去的 keyboard hook 會等待這個按鍵組合，並會在擁有焦點之視窗中（目前是你所使用的文書軟體）模擬【Alt+E】和【C】兩個鍵的按鍵動作。這樣的按鍵組合會使文書編輯軟體拉下【編輯(E)】選單並選取【複製(C)】命令。

當這個按鍵組合執行完畢，辭典軟體會切換到前景來，並搜尋位於剪貼簿中的這個字，將結果呈現出來。這時，我可以在螢幕上選取一個同義字並壓下一個代表【取代】動作的按鈕。於是辭典軟體把這個新字複製到剪貼簿中，並喚起文書編輯軟體，再次模擬按鍵【Alt+E】和【P】鍵。這個按鍵組合會選取文書編輯軟體的【編輯(E)】選單中的【貼上(P)】命令，於是原先的字（即剛剛被你標記起來的字）會被你所選取的（從辭典中查出來的）同義字取代。

另一個按鍵模擬應用是撰寫應用程式的測試描述檔（script for testing），你可以寫一個小程式（script）來做一些重複性高的測試工作。傳統的測試方法需要投入大量人力，整天不斷敲打鍵盤，進行測試，直到程式出現問題爲止。找出問題所在並修復之後，又需要再進行回歸測試以確保程式的正確性（保證原本正確的功能並沒有被程式員失手又改掉）。通常，這些測試工作需要不斷重複地進行，既費時也費力。如果能夠利用按鍵模擬的方法來替代重複性的鍵盤敲打動作，可以減輕不少時間上的負擔並可提高其精確度。

按鍵模擬的另一個應用是作爲「片語庫（phrase library）」，這是一種巨集重播工具。你可以寫一個程式，讓使用者播放一連串按鍵動作。舉個例子，也許你想重新定義 “~” 鍵，

使它一被按下就產生一連串的按鍵：【Ctrl+Esc】、【R】、【Calc】、【Enter】；這串按鍵組合會開啓工作列上的【開始】選單（Ctrl+Esc），然後選取其中的【執行】選項（R），然後執行 Cale 程式（Calc 和 Enter）。

為什麼 "Send" 或 "Post" 鍵盤訊息不是個好主意

當我開始寫上述這個程式的時候，我直覺的反應是 "post" WM_CHAR 訊息，達到鍵盤模擬的效果。首先，你必須取得你的按鍵模擬動作的對象（一個視窗）的視窗代碼。你可能已經知道了這個視窗代碼，或著你可以利用 FindWindow 函式或 WindowFromPoint 函式來取得。然後你使用 PostMessage 函式，將一個 WM_CHAR 訊息 "post" 給那個視窗。當然，wParam 和 lParam 參數都需事先準備好。

問題產生了。如果你想要送出 Alt+F 鍵來開啓一個程式的【檔案】選單，怎麼辦？靠 WM_CHAR 訊息是沒有辦法送出 Alt 鍵的，因爲 Alt 並不是一個字元。解決辦法是以 WM_SYSKEYDOWN 取代 WM_CHAR。然而，如果你 "post" WM_KEYDOWN 訊息的話，你同時也必須 "post" WM_KEYUP 訊息。這就要求你的程式必須更聰明些，可以處理更多事情。好，"post" 一個 Alt+F 按鍵組合的程式動作如下：

```
PostMessage(hwnd, WM_SYSKEYDOWN, VK_MENU, ...);
PostMessage(hwnd, WM_SYSKEYDOWN, VK_F, ...);
PostMessage(hwnd, WM_SYSKEYUP,VK_F, ...);
PostMessage(hwnd, WM_KEYUP,  VK_MENU, ...);
```

請注意 WM_SYSKEYDOWN 和 WM_SYSKEYUP 的使用。你可能會奇怪爲什麼我不使用 SendMessage 函式而使用 PostMessage 函式？原因是被 "posted" 的訊息會被執行緒以 GetMessage 函式從執行緒訊息佇列中取走。如果執行緒只對字元感興趣，而不想處理虛擬鍵碼的話，它會在 GetMessage 函式回返後呼叫 TranslateMessage 函式。如果我們用的是 SendMessage 函式，訊息會被直接送到視窗函式中，也就沒有機會經過 TranslateMessage 函式了。

這裡有幾個問題。第一個問題和 PostMessage 函式有關：同步化控制沒有辦法被正確掌

握。假設你模擬一個 Tab 字元，要交給對話盒中的一個控制元件，促使輸入焦點轉移到對話盒中的下一個子控制元件。這種情況下，下一次呼叫 PostMessage 函式時你就應該使用一個不同的視窗代碼做爲第一參數。你可能會想說，在每次呼叫 PostMessage 函式之前先呼叫 GetFocus 取得目前擁有輸入焦點的視窗代碼，問題就可以解決了。

但是這個方法沒有用，因爲 PostMessage 函式並不能夠讓按鍵訊息立刻被送出以及被處理。只有當控制元件釋放控制權，訊息才會被處理。所以，你必須在每次 “post” 訊息的動作之間，釋放控制權給其他程式。事實上如果不考慮 TranslateMessage 函式派不上用場的話，使用 SendMessage 函式會比較好。此外，PostMessage 技術其實也沒辦法有效運作，因爲 GetFocus 函式只能取得其呼叫者（一個執行緒）所產生的視窗的視窗代碼；如果目前擁有輸入焦點的視窗是被其他執行緒所產生，GetFocus 會傳回 NULL。你可以利用 AttachThreadInput 函式將兩個執行緒的 local input states 繫結在一起，以解決這個問題：

```
BOOL AttachThreadInput(DWORD idAttach, DWORD idAttachTo, BOOL fAttach);
```

另一個問題和 PostMessage 以及 SendMessage 都有關係。不管哪一個函式都不會將按鍵訊息放置在虛擬輸入佇列（virtualized input queue）中。之前我曾說過，當執行緒從執行緒的虛擬輸入佇列中取走訊息，Windows 會更新執行緒的「同步按鍵狀態陣列」。此時，如果一個視窗函式正在處理一個被 “posted” 或被 “sent” 的按鍵訊息，並且呼叫 GetKeyState 函式，它取回的將是一份錯誤的資訊。如果視窗函式利用這一份錯誤資訊來決定它應該採取的動作，那麼它的反應動作當然也是錯誤的。

誠如你所看到的，不論使用 PostMessage 函式或是 SendMessage 函式，都不是好方法。它們唯一可以有效運作的機會是，你非常瞭解並熟悉你要模擬按鍵動作的對象（一個視窗）。我決定尋求其它方法。

使用 keybd_event 函式如何？

放棄使用 PostMessage 和 SendMessage 函式，我現在想利用 keybd_event 函式來模擬按

鍵。的確，我用了 keybd_event 函式來模擬按鍵，它運作十分良好！使用 keybd_event 函式，不需判斷哪一個鍵是系統鍵，哪一個鍵又是一般鍵。系統會正確地設定按鍵訊息的 lParam 參數的位元旗標、系統的「非同步按鍵狀態陣列」、以及每一個執行緒的「同步按鍵狀態陣列」。

唉呀，但是使用 keybd_event 函式卻有兩件事情一直困擾著我。第一件事情是，如果有一長串按鍵訊息被餵給系統的輸入佇列，此時使用者有可能在鍵盤上敲一些按鍵，這些按鍵會摻雜到我們想要模擬的按鍵組合之中，於是模擬出來的視窗行爲將無可預料。

第二件我不喜歡的事情是，如果使用 keybd_event 函式，當所有模擬按鍵被處理完畢，keybd_event 函式並不會做任何通知。舉個例，本節一開始我介紹過的辭典軟體，它強制送出一串按鍵【Alt+E】和【C】，告訴文書處理軟體將被標示起來的字複製到剪貼簿中。當這些按鍵執行完畢，辭典軟體知道它所需要的資料已經被放在剪貼簿上了。如果辭典軟體用的是 keybd_event 函式，它就不知道那些模擬按鍵何時處理完畢，也就不知何時才能到剪貼簿中提取資料。

因爲上述缺點，我們還是再尋求其它方法吧！

使用 Journal Playback Hook

最後，我還是決定使用 journal playback hook 來做按鍵模擬。我曾在第 6 章仔細討論過 journal playback hook，所以這裡不再多做介紹。掛上一個 journal playback hook 會促使系統將所有的鍵盤和滑鼠硬體輸入除能（disable）。事實上滑鼠游標會被凍結在螢幕上，只有當 playback hook 播放滑鼠移動訊息，或是當 journal playback hook 被取消，滑鼠才能再移動。

當 journal playback hook 被掛上之後，系統會呼叫其 filter function 以便能夠得到下一個硬體事件，並派送（dispatch）給系統中的各個視窗。爲了模擬按鍵動作，每當我的 journal playback filter function 被呼叫，它就製造出 WM_(SYS)KEYDOWN 訊息和 WM_(SYS)KEYUP 訊息。所有模擬按鍵的組合都播放完畢後，hook 即被卸除。

使用 journal playback hook，前一節困擾我們的兩件事情都會解決。第一，由於 journal playback hook 會將所有的硬體輸入除能（disable），使用者不可能敲下按鍵，更不可能造成「我們想要模擬的按鍵」與「使用者敲入的按鍵」龍蛇雜處的情況，所以第一個問題跟本不可能發生。至於第二個問題，由於掛上 journal playback hook 會促使系統將系統中所有的執行緒的虛擬輸入佇列（virtualized input queue）繫結在一起，所以按鍵訊息被完全處理完畢後，我的 filter function 才會被呼叫。也就是說我們可以輕鬆判斷出整個模擬按鍵的組合是否已被處理完畢。

很明顯我們可以看出，使用 journal playback hook 來模擬按鍵是一個非常不錯的方法，它保證同步與非同步的「按鍵狀態陣列」會被正確地更新，而且同時存在的多個視窗焦點也可以被正確地維護好。然而，使用 journal playback hook，你需要撰寫一大堆程式碼，這是它最大的敗筆，好消息是，我幫你做掉了一切。

SKDemo 範例程式

SKDemo 程式（SKDemo.EXE）的原始碼顯示於**程式列表 8.1 ~ 8.5** 中，示範如何強制送出按鍵訊息到其它程式之中。當你執行這個程式，對話盒如**圖 8.1** 所示。

在此對話盒的最頂端有一個【Windows】combobox，記錄著系統中所有的最上層（top level）視窗。你隨時可以按下【Refresh】鈕來重新顯示 combobox 的內容。【Key String】combobox 內含一組預先定義好的測試字串，藉著 SKDemo 程式，你可以將這些字串送給各個視窗。你也可以在 combobox 中建立你自己的測試字串。

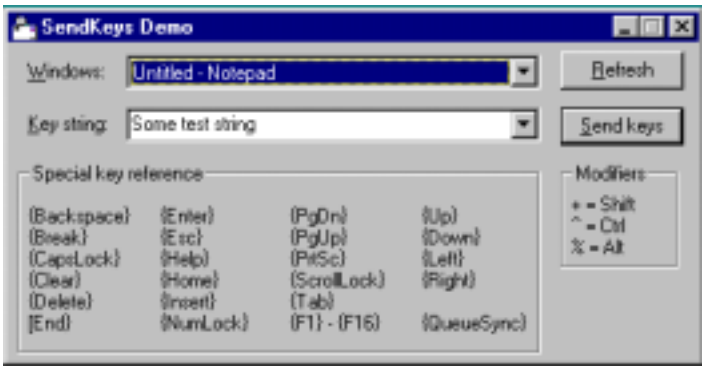


圖 8.1　Send Keys demo 程式（SKDemo.EXE）

當你按下【Send keys】鈕，程式會執行以下的碼（在 SKDemo.C 檔中）：

```
void SKDemo_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {

   TCHAR szBuf[MAXKEYSTRBUFFERSIZE];
   SKERR SKErr;
   HWND hwndCB, hwndT;
   int nIndex;

   switch (id) {
      .
      .
      .
      case IDC_SENDKEYS:
         hwndCB = GetDlgItem(hwnd, IDC_WINDOWS);
         nIndex = ComboBox_GetCurSel(hwndCB);
         hwndT = (HWND) ComboBox_GetItemData(hwndCB, nIndex);
         if (!IsWindow(hwndT)) {
            adgMB(__TEXT("Window no longer exists."));
         } else {
            Edit_GetText(GetDlgItem(hwnd, IDC_KEYSTRING),
               szBuf, adgARRAY_SIZE(szBuf));
            if ((SKErr = SendKeys(hwnd, szBuf)) != SKERR_NOERROR) {
               adgMB(g_szSKErrors[SKErr]);
            } else
               SetForegroundWindow(hwndT);
         }
         break;
```

```
        .
        .
        .
    }
}
```

首先，取得使用者在【Windows】combobox 中選擇的視窗的代碼，並看看此視窗是否還存在。接著，取得【Key String】combobox 中的字串，呼叫 SendKeys 函式，傳入 SKDemo 視窗代碼以及上述的按鍵字串。如果 SendKeys 函式的傳回值不是 SKERR_NOERROR，表示有錯誤發生，此時會出現一個訊息盒，將錯誤類型告訴使用者。如果 SendKeys 函式順利地剖析了它收到的按鍵字串，此時 playback hook 也已掛上，當我們的執行緒流程回到其訊息迴路時，hook 便開始播放那一串按鍵字串。

就在完成 WM_COMMAND 訊息的處理之前，上一段程式碼會呼叫 SetForegroundWindow 函式，令【Windows】combobox 中所指定的視窗作用起來（activated），注意，SetForegroundWindow 函式必須在 hook 被掛上之後才呼叫，這是因爲欲使「另一個執行緒所產生之視窗」作用起來（activated），是一個非同步事件；如果我們在掛上 hook 之前呼叫 SetForegroundWindow 函式，就不能保證當 playback hook 要播放模擬按鍵時，【Windows】combobox 中所指定的視窗已準備好要接受按鍵訊息了。反過來說，在 hook 掛上之後，系統中所有執行緒的虛擬輸入佇列（virtualized input queue）都已繫結在一起，而且 local input state 變數也被共享。如果我們在這個時候呼叫 SetForegroundWindow 函式，【Windows】combobox 中所指定的視窗會在函式回返之時，被置於前景。

當模擬按鍵正被播放時，使用者可以按下 Ctrl+Esc 鍵取消 journal playback hook。如果這樣的事情真的發生，系統會“post”一個 WM_CANCELJOURNAL 訊息到執行緒的訊息佇列中，通知 SKDemo 程式。當 SKDemo 程式的訊息迴路取得這個訊息，就會顯示一個對話盒，如圖 **8.2** 所示。表 **8.8** 列出此程式所需的檔案。

圖 8.2　取消 journal playback hook。

表 8.8　建立 SKDemo 程式所需用到的檔案

檔案	說明
SKDemo.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
SendKeys.C	此爲一個函式庫，提供一些函式讓你可以輕易地播放按鍵訊息。你可以應用於自己的程式中。
SendKeys.H	SendKeys.C 檔中函式原型的宣告。任何程式只要使用到 SendKey.C 檔中的函式，皆需含入此檔。
SKDemo.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 SKDemo.RC 檔中所有資源的 ID。
SKDemo.ICO	主視窗圖示（icon）。
SKDemo.MAK	Visual C++ 的 MAK 檔。

SKDemo.ICO

程式列表 8.1　SKDemo.C

```
#0001  /****************************************************
#0002  Module name: SKDemo.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Demonstrates using a dialog box for an application's main window
#0006  ****************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details. */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single-line comment */
#0013  #include "SendKeys.h"
#0014  #include "resource.h"
```

```
#0015
#0016
#0017 ////////////////////////////////////////////////////////////////
#0018
#0019
#0020 #define MAXKEYSTRBUFFERSIZE   1024
#0021
#0022
#0023 ////////////////////////////////////////////////////////////////
#0024
#0025
#0026 BOOL SKDemo_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0027
#0028    int x;
#0029    HWND hwndCB;
#0030    TCHAR *szKeyStrings[] = {
#0031       __TEXT("Some test string"),
#0032       __TEXT("%fonsetup.txt{Enter}"),
#0033       __TEXT("^{ESC}"),
#0034       __TEXT("Notify when playback complete{QueueSync}"),
#0035       __TEXT("{F1}"),
#0036       __TEXT("!@#$&*<>?:"),
#0037       __TEXT("z{Left}{Del}ABCD"),
#0038       __TEXT("xyz{Home}+{End}{Del}ABCD"),
#0039       __TEXT("{CAPSLOCK}abc{CAPSLOCK}abc"),
#0040       __TEXT("%"),
#0041       __TEXT("%+{TAB}"),
#0042       __TEXT("{{}Braces{}}"),
#0043       __TEXT("%F"),
#0044       __TEXT("{x 10}"),
#0045       __TEXT("{H 20}"),
#0046       __TEXT("{LeFt 5}"),
#0047       __TEXT("{%}{+}+{^}{D}"),
#0048       __TEXT("Error test: Missing close brace{ENTER"),
#0049       __TEXT("Error test: Invalid key {BOBO}"),
#0050       __TEXT("Error test: Missing close parenthesis+(jeff"),
#0051       __TEXT("Error test: Invalid count {z z}"),
#0052       __TEXT("Error test: String too long{J 3000}")
#0053    };
#0054
#0055    // Load the key strings combobox with a set of default string.
#0056    hwndCB = GetDlgItem(hwnd, IDC_KEYSTRING);
#0057    ComboBox_LimitText(hwndCB, MAXKEYSTRBUFFERSIZE);
#0058    for (x = 0; x < adgARRAY_SIZE(szKeyStrings); x++)
#0059       ComboBox_AddString(hwndCB, szKeyStrings[x]);
#0060    ComboBox_SetCurSel(hwndCB, 0);
```

```
#0061
#0062    // Force the initial loading of the window caption combobox.
#0063    FORWARD_WM_COMMAND(hwnd, IDC_WNDREFRESH,
#0064       GetDlgItem(hwnd, IDC_WNDREFRESH), BN_CLICKED, PostMessage);
#0065
#0066    adgSETDLGICONS(hwnd, IDI_SKDEMO, IDI_SKDEMO);
#0067    return(TRUE);               // Accepts default focus window.
#0068 }
#0069
#0070
#0071 ////////////////////////////////////////////////////////////////
#0072
#0073
#0074 static const TCHAR *g_szSKErrors[] = {
#0075    __TEXT("No error"),
#0076    __TEXT("Missing close brace"),
#0077    __TEXT("Invalid key"),
#0078    __TEXT("Missing close parenthesis"),
#0079    __TEXT("Invalid count"),
#0080    __TEXT("String too long"),
#0081    __TEXT("Can't install hook")
#0082 };
#0083
#0084
#0085 ////////////////////////////////////////////////////////////////
#0086
#0087
#0088 void SKDemo_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0089
#0090    TCHAR szBuf[MAXKEYSTRBUFFERSIZE];
#0091    SKERR SKErr;
#0092    HWND hwndCB, hwndT;
#0093    int nIndex;
#0094
#0095    switch (id) {
#0096       case IDC_WNDREFRESH:
#0097          hwndCB = GetDlgItem(hwnd, IDC_WINDOWS);
#0098          ComboBox_ResetContent(hwndCB);
#0099
#0100          for (hwndT = GetFirstChild(GetDesktopWindow());
#0101               IsWindow(hwndT); hwndT = GetNextSibling(hwndT)) {
#0102
#0103             if (IsWindowVisible(hwndT)) {
#0104                GetWindowText(hwndT, szBuf, adgARRAY_SIZE(szBuf));
#0105                if (szBuf[0] != 0) {
#0106                   nIndex = ComboBox_AddString(hwndCB, szBuf);
```

```
#0107                ComboBox_SetItemData(hwndCB, nIndex, hwndT);
#0108             }
#0109          }
#0110       }
#0111       ComboBox_SetCurSel(hwndCB, 0);
#0112       break;
#0113
#0114    case IDC_SENDKEYS:
#0115       hwndCB = GetDlgItem(hwnd, IDC_WINDOWS);
#0116       nIndex = ComboBox_GetCurSel(hwndCB);
#0117       hwndT = (HWND) ComboBox_GetItemData(hwndCB, nIndex);
#0118       if (!IsWindow(hwndT)) {
#0119          adgMB(__TEXT("Window no longer exists."));
#0120       } else {
#0121          Edit_GetText(GetDlgItem(hwnd, IDC_KEYSTRING),
#0122             szBuf, adgARRAY_SIZE(szBuf));
#0123          if ((SKErr = SendKeys(hwnd, szBuf)) != SKERR_NOERROR) {
#0124             adgMB(g_szSKErrors[SKErr]);
#0125          } else
#0126             SetForegroundWindow(hwndT);
#0127       }
#0128       break;
#0129
#0130    case IDCANCEL:            // Allows dialog box to close.
#0131       PostQuitMessage(0);
#0132       break;
#0133 }
#0134 }
#0135
#0136
#0137 ////////////////////////////////////////////////////////////////////
#0138
#0139
#0140 void SKDemo_OnQueueSync (HWND hwnd) {
#0141
#0142    adgMB(__TEXT("Got WM_QUEUESYNC message!"));
#0143 }
#0144
#0145
#0146 ////////////////////////////////////////////////////////////////////
#0147
#0148
#0149 BOOL WINAPI SKDemo_DlgProc (HWND hwnd, UINT uMsg,
#0150    WPARAM wParam, LPARAM lParam) {
#0151
#0152    switch (uMsg) {
```

```
#0153
#0154       // Standard Window's messages
#0155       adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, SKDemo_OnInitDialog);
#0156       adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    SKDemo_OnCommand);
#0157       adgHANDLE_DLGMSG(hwnd, WM_QUEUESYNC,  SKDemo_OnQueueSync);
#0158    }
#0159    return(FALSE);              // We didn't process the message.
#0160 }
#0161
#0162
#0163 ////////////////////////////////////////////////////////////////
#0164
#0165
#0166 int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0167    LPSTR lpszCmdLine, int nCmdShow) {
#0168
#0169    MSG msg;
#0170    HWND hwnd;
#0171
#0172    adgWARNIFUNICODEUNDERWIN95();
#0173
#0174    // Create a modeless dialog box instead of a modal dialog box because we
#0175    // need to have more control over the message loop processing.
#0176    hwnd = CreateDialog(hinstExe, MAKEINTRESOURCE(IDD_SKDEMO), NULL,
#0177       SKDemo_DlgProc);
#0178    adgASSERT(IsWindow(hwnd));
#0179
#0180    // Continue to loop until a WM_QUIT message comes out of the queue.
#0181    while (GetMessage(&msg, NULL, 0, 0)) {
#0182
#0183       // A user can cancel a journal playback hook by pressing Ctrl+Esc. When
#0184       // this happens, the operating system unhooks the journal playback hook
#0185       // we have installed and notifies our thread by posting it a
#0186       // WM_CANCELJOURNAL message.
#0187       if (msg.message == WM_CANCELJOURNAL)
#0188          adgMB(__TEXT("Sendkeys canceled by user"));
#0189
#0190       // Call IsDialogMessage so that the keyboard can be used to control
#0191       // focus in the dialog box.
#0192       if (!IsDialogMessage(hwnd, &msg)) {
#0193          TranslateMessage(&msg);
#0194          DispatchMessage(&msg);
#0195       }
#0196    }
#0197
#0198    // The application is terminating; destroy the modeless dialog box.
```

```
#0199    DestroyWindow(hwnd);
#0200    return(0);
#0201 }
#0202
#0203
#0204 ////////////////////////// End of File //////////////////////////
```

程式列表 8.2　SendKeys.C

```
#0001 /****************************************************************
#0002 Module name: SendKeys.c
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Functions to simulate keystrokes to Windows applications
#0006 ****************************************************************/
#0007
#0008
#0009 #include "..\Win95ADG.h"         /* See Appendix A for details. */
#0010 #include <windows.h>
#0011 #include <windowsx.h>
#0012 #pragma warning(disable: 4001)    /* Single-line comment */
#0013 #include <tchar.h>
#0014 #include <stdio.h>
#0015 #include "SendKeys.h"
#0016
#0017
#0018 ////////////////////////////////////////////////////////////////
#0019
#0020
#0021 static struct {
#0022    SHORT   asVirtKeys[1024];     // Virtual-key codes for playback
#0023    int     nVirtKeyNum;          // Index into asVirtKeys
#0024    int     nMaxVirtKeys;         // Max entries in asVirtKeys
#0025    LPCTSTR pszKeys;              // Pointer to next character in input string
#0026    BOOL    fNoTokensRetrievedYet; // Flag that indicates playback beginning
#0027    HHOOK   hhook;               // Hook handle of playback hook
#0028    HWND    hwndQueueSyncTarget;   // Window to receive WM_QUEUESYNC messages
#0029 } g_SKD = { { 0 }, 0, 0, NULL, TRUE, NULL, NULL };
#0030
#0031
#0032 ////////////////////////////////////////////////////////////////
#0033
#0034
#0035 // We are creating a virtual key that represents the playback of a
#0036 // WM_QUEUESYNC message. We chose -1 as the value for this virtual key because
```

```
#0037  // Windows virtual keys range from 0 to 255.
#0038  #define VK_QUEUESYNC ((SHORT) -1)
#0039
#0040
#0041  // The table of special key codes
#0042  typedef struct {
#0043     SHORT sVirtKey;
#0044     LPCTSTR szKeyName;
#0045  } SPECIALKEY;
#0046
#0047
#0048  static const SPECIALKEY g_SpecialKeys[] = {
#0049     { VK_QUEUESYNC, __TEXT("QUEUESYNC")  },
#0050     { VK_CAPITAL,   __TEXT("CAPSLOCK")   },
#0051     { VK_NUMLOCK,   __TEXT("NUMLOCK")    },
#0052     { VK_SCROLL,    __TEXT("SCROLLOCK")  },
#0053     { VK_ESCAPE,    __TEXT("ESCAPE")     },
#0054     { VK_ESCAPE,    __TEXT("ESC")        },
#0055     { VK_RETURN,    __TEXT("ENTER")      },
#0056     { VK_HELP,      __TEXT("HELP")       },
#0057     { VK_SNAPSHOT,  __TEXT("PRTSC")      },
#0058     { VK_TAB,       __TEXT("TAB")        },
#0059     { VK_CONTROL,   __TEXT("BREAK")      },
#0060     { VK_CLEAR,     __TEXT("CLEAR")      },
#0061     { VK_BACK,      __TEXT("BACKSPACE")  },
#0062     { VK_BACK,      __TEXT("BS")         },
#0063     { VK_BACK,      __TEXT("BKSP")       },
#0064     { VK_DELETE,    __TEXT("DELETE")     },
#0065     { VK_DELETE,    __TEXT("DEL")        },
#0066     { VK_INSERT,    __TEXT("INSERT")     },
#0067     { VK_LEFT,      __TEXT("LEFT")       },
#0068     { VK_RIGHT,     __TEXT("RIGHT")      },
#0069     { VK_UP,        __TEXT("UP")         },
#0070     { VK_DOWN,      __TEXT("DOWN")       },
#0071     { VK_PRIOR,     __TEXT("PGUP")       },
#0072     { VK_NEXT,      __TEXT("PGDN")       },
#0073     { VK_HOME,      __TEXT("HOME")       },
#0074     { VK_END,       __TEXT("END")        },
#0075     { VK_F1,        __TEXT("F1")         },
#0076     { VK_F2,        __TEXT("F2")         },
#0077     { VK_F3,        __TEXT("F3")         },
#0078     { VK_F4,        __TEXT("F4")         },
#0079     { VK_F5,        __TEXT("F5")         },
#0080     { VK_F6,        __TEXT("F6")         },
#0081     { VK_F7,        __TEXT("F7")         },
#0082     { VK_F8,        __TEXT("F8")         },
```

```
#0083    { VK_F9,       __TEXT("F9")        },
#0084    { VK_F10,      __TEXT("F10")       },
#0085    { VK_F11,      __TEXT("F11")       },
#0086    { VK_F12,      __TEXT("F12")       },
#0087    { VK_F13,      __TEXT("F13")       },
#0088    { VK_F14,      __TEXT("F14")       },
#0089    { VK_F15,      __TEXT("F15")       },
#0090    { VK_F16,      __TEXT("F16")       },
#0091    { 0,          NULL              }
#0092  };
#0093
#0094
#0095  ////////////////////////////////////////////////////////////////
#0096
#0097
#0098  // Convert a special key string to its equivalent virtual-key code.
#0099  static SHORT SendKeys_SpecialKeyToVirtKey (LPCTSTR pszSpecialKey) {
#0100
#0101    const SPECIALKEY* psk = g_SpecialKeys;
#0102    SHORT sVirtKey;             // Assumes that pszSpecialKey not found.
#0103
#0104    // Scan the array and compare each of the possible strings.
#0105    while (((sVirtKey = psk->sVirtKey) != 0) &&
#0106           (_tcsicmp(pszSpecialKey, psk->szKeyName) != 0))
#0107      psk++;
#0108
#0109    if ((sVirtKey == 0) && (_tcslen(pszSpecialKey) == 1)) {
#0110
#0111      // The special key was not found in the list.
#0112      // If the special key is a single character, convert that character to
#0113      // its virtual-key code equivalent.
#0114      // Note 1: This must come after the preceding loop so that special single
#0115      //        characters are checked first (i.e.-"~" (tilde)).
#0116      // Note 2: This check is necessary for other special single characters
#0117      //        (i.e.-"+^%{}()).
#0118      sVirtKey = VkKeyScan(*pszSpecialKey);
#0119    }
#0120
#0121    return(sVirtKey);
#0122  }
#0123
#0124
#0125  ////////////////////////////////////////////////////////////////
#0126
#0127
#0128  #define APPENDVIRTKEY(sVirtKey) \
```

```
#0129    {                                                    \
#0130       if (g_SKD.nMaxVirtKeys == adgARRAY_SIZE(g_SKD.asVirtKeys)) { \
#0131          SKErr = SKERR_STRINGTOOLONG;                        \
#0132          break;                                           \
#0133       } else                                              \
#0134          g_SKD.asVirtKeys[g_SKD.nMaxVirtKeys++] = sVirtKey;       \
#0135    }
#0136
#0137
#0138  // Function to preprocess the next token in the input string.
#0139  static SKERR SendKeys_PreprocessKeys (void) {
#0140
#0141    TCHAR cChar;                // Current character being processed
#0142    TCHAR szSpecialKey[16];       // Buffer big enough for a special key name
#0143    LPCTSTR pEndOfToken;          // Pointer to end of current token
#0144    SHORT sVirtKey = 0;           // Virtual key equivalent of token
#0145    int nCount = 1;              // Number of times to play back virtual key
#0146    SKERR SKErr = SKERR_NOERROR;   // Current error state
#0147
#0148    // Get the next character from the input string.
#0149    switch (cChar = *g_SKD.pszKeys++) {
#0150
#0151       case 0:                  // Reached the end of the input string
#0152          g_SKD.pszKeys--;        // Point back to the zero-byte
#0153          break;
#0154
#0155       case __TEXT('('):          // Beginning of subgroup
#0156
#0157          // While not at the end of the input string and not at a close paren
#0158          while ((*g_SKD.pszKeys != 0) && (*g_SKD.pszKeys != __TEXT(')'))) {
#0159
#0160             // Add the next character to the array.
#0161             SKErr = SendKeys_PreprocessKeys();
#0162             if (SKErr != SKERR_NOERROR)
#0163                break;
#0164          }
#0165
#0166          // If terminating because of end of string and not because of right
#0167          // paren, there is an error.
#0168          if (*g_SKD.pszKeys == 0)
#0169             SKErr = SKERR_MISSINGCLOSEPAREN;
#0170          else
#0171             g_SKD.pszKeys++;      // Skips past the close paren.
#0172          break;
#0173
#0174       case __TEXT('~'):          // Presses the Enter key.
```

```
#0175          APPENDVIRTKEY(VK_RETURN); // Presses the Enter key.
#0176          break;
#0177
#0178       case __TEXT('+'):          // Presses the Shift key.
#0179       case __TEXT('^'):          // Presses the Control key.
#0180       case __TEXT('%'):          // Presses the Alt key.
#0181
#0182          cChar = (TCHAR) ((cChar == __TEXT('+')) ? VK_SHIFT :
#0183                        ((cChar == __TEXT('^')) ? VK_CONTROL : VK_MENU));
#0184          APPENDVIRTKEY(cChar);    // Presses the Shift/Control/Alt keys.
#0185
#0186          // Call preprocess keys recursively to get the keys to which we
#0187          // are applying the modifiers here.
#0188          if ((SKErr = SendKeys_PreprocessKeys()) != SKERR_NOERROR)
#0189             break;
#0190
#0191          APPENDVIRTKEY(cChar);       // Releases the Shift/Control/Alt keys.
#0192          break;
#0193
#0194       case __TEXT('{'):         // Beginning of special key text
#0195          if (*g_SKD.pszKeys == __TEXT('}')) {
#0196
#0197             // The special character is a close brace.
#0198             sVirtKey = VkKeyScan(__TEXT('}'));
#0199
#0200             // Point past the virtual key.
#0201             pEndOfToken = ++g_SKD.pszKeys;
#0202          }
#0203
#0204          // Locate the end of the first token in the braced expression.
#0205          // This is either:
#0206          //    1. The end of string,
#0207          //    2. a special word/symbol followed by a close brace, or
#0208          //    3. a special word/symbol followed by a space and a number.
#0209          pEndOfToken = _tcspbrk(g_SKD.pszKeys, __TEXT(" }"));
#0210
#0211          if (pEndOfToken == NULL)  {
#0212             SKErr = SKERR_MISSINGCLOSEBRACE;
#0213             break;
#0214          }
#0215
#0216          if (sVirtKey == 0) {
#0217
#0218             // The key must not be the close brace. We must determine the
#0219             // virtual key corresponding to this special key.
#0220             int nNumChars = pEndOfToken - g_SKD.pszKeys;
```

```
#0221            _tcsncpy(szSpecialKey, g_SKD.pszKeys, nNumChars);
#0222            szSpecialKey[nNumChars] = 0; // Force string termination
#0223            sVirtKey = SendKeys_SpecialKeyToVirtKey(szSpecialKey);
#0224            if (sVirtKey == 0)  {
#0225               SKErr = SKERR_INVALIDKEY;
#0226               break;
#0227            }
#0228         }
#0229
#0230         // Calculate the repeat count for this special character.
#0231         if (*pEndOfToken == __TEXT(' ')) {
#0232            if ((_stscanf(++pEndOfToken, __TEXT("%d"), &nCount) == 0) ||
#0233                (nCount == 0))  {
#0234
#0235               SKErr = SKERR_INVALIDCOUNT;
#0236               break;
#0237            }
#0238         }
#0239
#0240         // Point to char after the closing brace so that parsing may continue.
#0241         g_SKD.pszKeys = _tcschr(pEndOfToken, __TEXT('}'));
#0242         if (g_SKD.pszKeys == NULL) {
#0243            SKErr = SKERR_MISSINGCLOSEBRACE;
#0244            break;
#0245         }
#0246         g_SKD.pszKeys++;       // Skip over the close brace
#0247
#0248         if (sVirtKey == VK_QUEUESYNC) {
#0249
#0250            // Add special VK_QUEUESYNC virtual key that will cause a
#0251            // WM_QUEUESYNC message to be played back to the
#0252            // hwndQueueSyncTarget window.
#0253            APPENDVIRTKEY(VK_QUEUESYNC);
#0254            break;
#0255         }
#0256
#0257         // Add this special key to the virtual-key list by falling through
#0258         // to the following default case.
#0259
#0260      default:                // Just a normal character
#0261         if (cChar != __TEXT('{')) {
#0262
#0263            // We didn't fall through from the preceding case.
#0264            sVirtKey = VkKeyScan(cChar);
#0265         }
#0266
```

```
#0267         if (HIBYTE(sVirtKey) & 1)
#0268            APPENDVIRTKEY(VK_SHIFT); // Presses the Shift key.
#0269
#0270         while (nCount--)
#0271            APPENDVIRTKEY(LOBYTE(sVirtKey));
#0272
#0273         if (HIBYTE(sVirtKey) & 1)
#0274            APPENDVIRTKEY(VK_SHIFT); // Releases the Shift key.
#0275         break;
#0276    }
#0277    return(SKErr);
#0278 }
#0279
#0280
#0281 ////////////////////////////////////////////////////////////////
#0282
#0283
#0284 SKERR WINAPI SendKeys_InitPreprocessKeys (LPCTSTR pszPassedKeys) {
#0285
#0286    SKERR SKErr;
#0287
#0288    // Clear out the array and the index into the array.
#0289    g_SKD.nMaxVirtKeys = g_SKD.nVirtKeyNum = 0;
#0290
#0291    // Use an internal variable for input string parsing.
#0292    g_SKD.pszKeys = pszPassedKeys;
#0293
#0294    // While there are more more characters in the string to be parsed, call
#0295    // the preprocessor to evaluate the next token.
#0296    while (*g_SKD.pszKeys != 0)
#0297       if ((SKErr = SendKeys_PreprocessKeys()) != SKERR_NOERROR)
#0298          break;
#0299
#0300    return(SKErr);
#0301 }
#0302
#0303
#0304 ////////////////////////////////////////////////////////////////
#0305
#0306
#0307 // This function returns the next key event from the array.
#0308 // It returns True if a token was retrieved from the list and
#0309 // False if there were no more tokens to retrieve.
#0310 // This function is also used to initialize playback.  This is done by
#0311 // calling it and passing zero in the psVirtKey parameter.
#0312 static BOOL SendKeys_GetToken (SHORT *psVirtKey,
```

```
#0313    BOOL *pfPressDown, BOOL *pfSysKey) {
#0314
#0315    static SHORT sVirtKey = 0;
#0316    static BOOL fKeyIsDown = FALSE;
#0317    static BOOL fKeyPressWhileAltDown = FALSE;
#0318    static BOOL fShiftDown = FALSE, fControlDown = FALSE, fAltDown = FALSE;
#0319
#0320    if (psVirtKey == NULL) {
#0321
#0322       // Prime the token engine.
#0323       sVirtKey = 0;
#0324       fKeyIsDown = FALSE;
#0325
#0326       // Ensure that none of the shift state keys are down.
#0327       fShiftDown = fControlDown = fAltDown = FALSE;
#0328
#0329       // Start playback from the first (0th) entry in the array.
#0330       g_SKD.nVirtKeyNum = 0;
#0331       return(TRUE);
#0332    }
#0333
#0334    // Make sure that the key is not a shift state key.
#0335    if ((sVirtKey != VK_SHIFT) && (sVirtKey != VK_CONTROL) &&
#0336        (sVirtKey != VK_MENU)) {
#0337
#0338       // If we last played back a key and said that it was down, we now have
#0339       // to play back the same key and say that it is up.
#0340       if (sVirtKey != 0 && fKeyIsDown) {
#0341          *psVirtKey = sVirtKey;   // Same key as last time.
#0342          sVirtKey = 0;          // Next time in, use a new key.
#0343          *pfPressDown = FALSE;    // Release the key.
#0344          *pfSysKey = fAltDown && !fControlDown; // Is it a SYS key?
#0345          fKeyPressWhileAltDown = TRUE;
#0346          return(TRUE);
#0347       }
#0348    }
#0349
#0350    // Have all the key events in the array been played back?
#0351    if (g_SKD.nVirtKeyNum == g_SKD.nMaxVirtKeys)
#0352       return(FALSE);            // No more tokens to playback.
#0353
#0354    // Do special processing if we are playing back a shift state key.
#0355    switch (*psVirtKey = sVirtKey = g_SKD.asVirtKeys[g_SKD.nVirtKeyNum++]) {
#0356
#0357       case VK_SHIFT:
#0358
```

```
#0359         // Toggle the state of the Shift key.
#0360         *pfPressDown = (fShiftDown = !fShiftDown);
#0361         break;
#0362
#0363      case VK_CONTROL:
#0364
#0365         // Toggle the state of the Ctrl key.
#0366         *pfPressDown = (fControlDown = !fControlDown);
#0367         break;
#0368
#0369      case VK_MENU:
#0370
#0371         // Toggle the state of the Alt key.
#0372         *pfPressDown = (fAltDown = !fAltDown);
#0373
#0374         // If the Alt key is going down, reset this flag.
#0375         if (fAltDown)
#0376            fKeyPressWhileAltDown = FALSE;
#0377         break;
#0378
#0379      case VK_QUEUESYNC:
#0380
#0381         // To prevent playing back a VK_QUEUESYNC keydown and keyup, we
#0382         // pretend that a key is played back by setting sVirtKey to zero.
#0383         sVirtKey = 0;
#0384         break;
#0385
#0386      default:
#0387
#0388         // For any other key, make the event a keydown.
#0389         *pfPressDown = fKeyIsDown = fKeyPressWhileAltDown = TRUE;
#0390         break;
#0391   }
#0392
#0393   // Do some special checking to determine if the key is a SYS key or not.
#0394   if ((sVirtKey == VK_MENU) && (!fAltDown))
#0395      *pfSysKey = !fKeyPressWhileAltDown && !fControlDown;
#0396   else
#0397      *pfSysKey = fAltDown && !fControlDown;
#0398
#0399   return(TRUE);
#0400 }
#0401
#0402
#0403 ////////////////////////////////////////////////////////////////
#0404
```

```
#0405
#0406  // This is the journal playback hook callback function. Every time it is
#0407  // called, Windows requests the next keyboard event to be played back.
#0408  // This function determines what the next event is by calling
#0409  // SendKeys_GetToken, fills an EVENTMSG structure with the correct information
#0410  // about the keyboard event and playback time, and returns to let Windows
#0411  // process it. After all events have been played back, the function uninstalls
#0412  // itself and sets the g_SKD's hhook member to NULL.
#0413  LRESULT WINAPI SendKeys_JrnlPlayBackHook (int nCode,
#0414    WPARAM wParam, LPARAM lParam) {
#0415
#0416    PEVENTMSG pEvent;
#0417    static SHORT sVirtKey = 0;
#0418    static BOOL fKeyDown = FALSE;
#0419    static BOOL fSysKey = FALSE;
#0420
#0421    if (g_SKD.fNoTokensRetrievedYet) {
#0422
#0423       // If no tokens have ever been retrieved from the list, we must force
#0424       // ourselves to get the first one in case Windows sends us a HC_GETNEXT
#0425       // notification BEFORE an HC_SKIP notification.
#0426       SendKeys_GetToken(&sVirtKey, &fKeyDown, &fSysKey);
#0427       g_SKD.fNoTokensRetrievedYet = FALSE;
#0428    }
#0429
#0430    switch (nCode) {
#0431
#0432       case HC_SKIP:
#0433
#0434          // Prepare to return the next event the next time the hook code is
#0435          // HC_GETNEXT. If all events have been played, stop playing.
#0436          if (!SendKeys_GetToken(&sVirtKey, &fKeyDown, &fSysKey)) {
#0437
#0438             // There were no more tokens left to get.
#0439             UnhookWindowsHookEx(g_SKD.hhook);
#0440             g_SKD.hhook = NULL;
#0441          }
#0442          break;
#0443
#0444       case HC_GETNEXT:
#0445
#0446          // Copy current event to the EVENTMSG structure pointed to by lParam.
#0447          pEvent = (PEVENTMSG) lParam;
#0448          pEvent->time = GetTickCount();
#0449          if (sVirtKey == VK_QUEUESYNC) {
#0450             pEvent->message = WM_QUEUESYNC;
```

```
#0451            pEvent->paramL = (UINT) g_SKD.hwndQueueSyncTarget;
#0452            pEvent->paramH = 0;
#0453         } else {
#0454            if (!fSysKey)
#0455               pEvent->message = fKeyDown ? WM_KEYDOWN : WM_KEYUP;
#0456            else
#0457               pEvent->message = fKeyDown ? WM_SYSKEYDOWN : WM_SYSKEYUP;
#0458
#0459            // Scan code and virtual key.
#0460            pEvent->paramL = sVirtKey;
#0461
#0462            // Scan code; bit 15 is extended key.
#0463            pEvent->paramH = MapVirtualKey(sVirtKey, 0);
#0464         }
#0465         break;
#0466    }
#0467
#0468    // Number of milliseconds Windows should wait before processing the event.
#0469    return(0);
#0470 }
#0471
#0472
#0473 ////////////////////////////////////////////////////////////////
#0474
#0475
#0476 SKERR WINAPI SendKeys (HWND hwndQueueSyncTarget, LPCTSTR pszKeys) {
#0477
#0478    BYTE abKeyState[256];
#0479    SKERR SKErr = SendKeys_InitPreprocessKeys(pszKeys);
#0480    if (SKErr != SKERR_NOERROR)
#0481       return(SKErr);
#0482
#0483    SendKeys_GetToken(NULL, NULL, NULL); // Primes the token pump.
#0484    g_SKD.fNoTokensRetrievedYet = TRUE;
#0485    g_SKD.hwndQueueSyncTarget = hwndQueueSyncTarget;
#0486
#0487    // Install the journal playback hook.
#0488    g_SKD.hhook = SetWindowsHookEx(WH_JOURNALPLAYBACK,
#0489       SendKeys_JrnlPlayBackHook, GetModuleHandle(NULL), 0);
#0490    if (g_SKD.hhook == NULL)
#0491       return(SKERR_CANTINSTALLHOOK);
#0492
#0493    GetKeyboardState(abKeyState);    // Gets the current state of the keyboard.
#0494    abKeyState[VK_CONTROL] &= 0x7f;  // Forces the Ctrl key off.
#0495    abKeyState[VK_MENU]    &= 0x7f;  // Forces the Alt key off.
#0496    abKeyState[VK_SHIFT]   &= 0x7f;  // Forces the Shift key off.
```

```
#0497    abKeyState[VK_CAPITAL] &= 0x7e;  // Forces the Caps Lock key off.
#0498    abKeyState[VK_NUMLOCK] &= 0x7e;  // Forces the Num Lock key off.
#0499    SetKeyboardState(abKeyState);    // Sets the new state of the keyboard.
#0500
#0501    return(SKERR_NOERROR);
#0502 }
#0503
#0504
#0505 ////////////////////// End of File ////////////////////////////
```

程式列表 8.3　SendKeys.H

```
#0001 /*****************************************************************
#0002 Module name: SendKeys.h
#0003 Written by: Jeffrey Richter
#0004 Notices: Copyright (c) 1995 Jeffrey Richter
#0005 Purpose: Functions to simulate keystrokes to Windows applications
#0006 *****************************************************************/
#0007
#0008
#0009 typedef enum {
#0010    SKERR_NOERROR,
#0011    SKERR_MISSINGCLOSEBRACE,
#0012    SKERR_INVALIDKEY,
#0013    SKERR_MISSINGCLOSEPAREN,
#0014    SKERR_INVALIDCOUNT,
#0015    SKERR_STRINGTOOLONG,
#0016    SKERR_CANTINSTALLHOOK
#0017 } SKERR;
#0018
#0019 ////////////////////////////////////////////////////////////////
#0020
#0021
#0022 SKERR WINAPI SendKeys (HWND hwndQueueSyncTarget, LPCTSTR szKeys);
#0023
#0024
#0025 ////////////////////// End of File ////////////////////////////
```

程式列表 8.4　SKDemo.RC

```
#0001 //Microsoft Visual C++ generated resource script.
#0002 //
#0003 #include "resource.h"
```

```
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #include "windows.h"
#0011
#0012  ////////////////////////////////////////////////////////////////
#0013  #undef APSTUDIO_READONLY_SYMBOLS
#0014
#0015
#0016  #ifdef APSTUDIO_INVOKED
#0017  ////////////////////////////////////////////////////////////////
#0018  //
#0019  // TEXTINCLUDE
#0020  //
#0021
#0022  1 TEXTINCLUDE DISCARDABLE
#0023  BEGIN
#0024      "resource.h\0"
#0025  END
#0026
#0027  2 TEXTINCLUDE DISCARDABLE
#0028  BEGIN
#0029      "#include ""windows.h""\r\n"
#0030      "\0"
#0031  END
#0032
#0033  3 TEXTINCLUDE DISCARDABLE
#0034  BEGIN
#0035      "\r\n"
#0036      "\0"
#0037  END
#0038
#0039  ////////////////////////////////////////////////////////////////
#0040  #endif    // APSTUDIO_INVOKED
#0041
#0042
#0043  ////////////////////////////////////////////////////////////////
#0044  //
#0045  // Dialog
#0046  //
#0047
#0048  IDD_SKDEMO DIALOG DISCARDABLE  -32768, 5, 280, 119
#0049  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
```

```
#0050  CAPTION "SendKeys Demo"
#0051  FONT 8, "MS Sans Serif"
#0052  BEGIN
#0053      CONTROL        "&Windows:",IDC_STATIC,"Static",SS_SIMPLE | WS_GROUP,8,6,
#0054                     40,8
#0055      COMBOBOX       IDC_WINDOWS,48,4,168,120,CBS_DROPDOWNLIST | CBS_SORT |
#0056                     WS_VSCROLL | WS_TABSTOP
#0057      PUSHBUTTON     "&Refresh",IDC_WNDREFRESH,224,3,50,14
#0058      CONTROL        "&Key string:",IDC_STATIC,"Static",SS_SIMPLE | WS_GROUP,
#0059                     8,26,40,8
#0060      COMBOBOX       IDC_KEYSTRING,48,24,168,120,CBS_DROPDOWN |
#0061                     CBS_AUTOHSCROLL | WS_VSCROLL | WS_TABSTOP
#0062      DEFPUSHBUTTON  "&Send keys",IDC_SENDKEYS,224,24,50,14
#0063      GROUPBOX       "Special key reference",IDC_STATIC,4,44,212,72
#0064      LTEXT          "{Backspace}\n{Break}\n{CapsLock}\n{Clear}\n{Delete}\n[End}",
#0065                     IDC_STATIC,8,60,44,48
#0066      LTEXT          "{Enter}\n{Esc}\n{Help}\n{Home}\n{Insert}\n{NumLock}",
#0067                     IDC_STATIC,61,60,44,48
#0068      LTEXT          "{PgDn}\n{PgUp}\n{PrtSc}\n{ScrollLock}\n{Tab}\n{F1} - {F16}",
#0069                     IDC_STATIC,114,60,44,48
#0070      LTEXT          "{Up}\n{Down}\n{Left}\n{Right}\n\n{QueueSync}",
#0071                     IDC_STATIC,167,60,44,48
#0072      GROUPBOX       "Modifiers",IDC_STATIC,224,44,48,40
#0073      LTEXT          "+ = Shift\n^ = Ctrl\n% = Alt",IDC_STATIC,228,56,40,24
#0074  END
#0075
#0076
#0077  /////////////////////////////////////////////////////////////////
#0078  //
#0079  // Icon
#0080  //
#0081
#0082  IDI_SKDEMO            ICON    DISCARDABLE     "SKDemo.ico"
#0083
#0084  #ifndef APSTUDIO_INVOKED
#0085  /////////////////////////////////////////////////////////////////
#0086  //
#0087  // Generated from the TEXTINCLUDE 3 resource.
#0088  //
#0089
#0090
#0091  /////////////////////////////////////////////////////////////////
#0092  #endif    // not APSTUDIO_INVOKED
```

程式列表 8.5　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by SKDemo.rc
#0004  //
#0005  #define IDC_TIME                    100
#0006  #define IDD_CLOCK                   101
#0007  #define IDI_ICON1                   102
#0008  #define IDI_CLOCK                   102
#0009  #define IDD_SKDEMO                  103
#0010  #define IDI_SKDEMO                  104
#0011  #define IDC_WINDOWS                 1000
#0012  #define IDC_KEYSTRING               1001
#0013  #define IDC_SENDKEYS                1002
#0014  #define IDC_WNDREFRESH              1003
#0015  #define IDC_STATIC                  -1
#0016
#0017  // Next default values for new objects
#0018  //
#0019  #ifdef APSTUDIO_INVOKED
#0020  #ifndef APSTUDIO_READONLY_SYMBOLS
#0021  #define _APS_NEXT_RESOURCE_VALUE      105
#0022  #define _APS_NEXT_COMMAND_VALUE       40001
#0023  #define _APS_NEXT_CONTROL_VALUE       1000
#0024  #define _APS_NEXT_SYMED_VALUE         101
#0025  #endif
#0026  #endif
```

SendKeys 函式

SendKeys 是一個常常被用到的函式，無論在 Basic、Visual Basic、Microsoft Excel 巨集語言、Word 的 WordBasic 語言中，都有此函式的存在，但不同版本的 SendKeys 函式在用法上皆有些許差異。我所寫的 SendKeys 函式也是一樣（它極類似 Visual Basic 的 SendKeys 函式）。

當 SendKeys 被呼叫，它所做的第一件事是呼叫 SendKeys_InitPreprocessKeys 函式，處理字元字串，也就是【Key String】combobox 中所指定的字串。這個函式會將字元字串轉換爲一個由虛擬鍵碼構成的陣列。這個陣列會被儲存在全域結構 g_SKD 的 asVirtKeys 陣列中，而結構的 nMaxVirtKeys 欄位會記錄 asVirtKeys 陣列中的最大索引值。當這個

陣列被播放時，nVirtKeyNum 欄位會記錄「陣列之中下一個將被播放的虛擬鍵碼」的索引值。一旦 nVirtKeyNum 欄位的內容爲（nMaxVirtKeys-1），表示整個陣列已被播放完畢。

當 SendKeys_InitPreprocessKeys 函式被呼叫，它會設定結構中的 nMaxVirtKeys 和 nVirtKeyNum 欄位爲 0，並設定 pszKeys 欄位（一個指標）指向字元字串。然後，SendKeys_InitPreprocessKeys 函式會進入一個 while 迴圈，在其中呼叫 SendKeys_PreprocessKeys 函式，直到字串中所有的“token”都被解析出來，才跳出迴圈。如果有錯誤發生，SendKeys_PreprocessKeys 函式會停止，並傳回錯誤值。

舉個例子，想像一下使用者把一長串字串傳給 SendKeys 函式。由於結構中 asVirtKeys 陣列的長度是固定的，所以 SendKeys 函式會傳回 SKERR_STRINGTOOLONG。如果一切都很順利的話，SendKeys_PreprocessKeys 函式會傳回 SKERR_NOERROR，並且 SendKeys 函式會掛上 journal playback hook，用以播放按鍵。稍後我將探討它是如何運作的。現在，先讓我們看看 Send_PreprocessKeys 函式如何填寫 asVirtKeys 陣列。

每當 SendKeys_PreprocessKeys 被呼叫，它會嘗試判斷字串中的下一個“token”是什麼？這有數種可能：

如果字元是一個 0 位元組（zero-byte）

如果下一個 ”token” 的第一個字元是 0 位元組的話，SendKeys 函式會認爲已達字串的尾端，於是傳回 SKERR_NOERROR。

如果字元是一個括弧（parenthesis）

如果 SendKeys_PreprocessKeys 函式確定下一個字元是一個左括弧的話，它會遞迴呼叫自己，直到字串結束，或是直到找到右括弧。很快我就會爲你解釋爲什麼必須這樣做。

如果字元是一個 “~”

“~” 只是一個用來表示【Enter】鍵的特殊符號，所以 SendKeys 函式會在陣列中附加上一個 VK_RETURN。

如果字元是一個修飾字元（modifier）

如果 SendKeys_PreprocessKeys 函式所找到的下一個 “token” 是個特殊的修飾字元，例如 + 或 ^ 或 % 的話，它會分別在陣列中加入 VK_SHIFT 或 VK_CONTROL 或 VK_MENU，並遞迴呼叫自己。在遞迴呼叫期間，SendKeys_PreprocessKeys 函式會將下一個 ”token” 加到 asVirtKeys 陣列中，然後才回返。當 SendKeys_PreprocessKeys 函式從遞迴中抽身，會再一次在 asVirtKeys 陣列中加入 VK_SHIFT 或 VK_CONTROL 或 VK_MENU 虛擬鍵碼 -- 與遞迴呼叫之前所加入的虛擬鍵碼一樣。

如果字元是一個左大括弧（open brace）

如果下一個 “token” 一開始是一個左大括弧的話，SendKeys_PreprocessKeys 函式必須針對幾種不同的情況做檢查：

- 第一種情況是 “{}}”，這種情況允許你在字元資料流（character stream）中指定一個右括弧。
- 第二種情況是一些用以表達特殊鍵的字串，例如 “{BACKSPACE}” 。
- 第三種情況是一個按鍵（可能是個特殊鍵）緊跟著一個空白和一個數字。例如 “{Right 10}”，表示右鍵（方向鍵）應被重複模擬 10 次。

SendKeys_PreprocessKeys 函式會呼叫 SendKeys_SpecialKeyToVirtKey 函式來檢查看看字串是否用以表現一個特殊鍵。此函式只有一個參數，是一個以 0 結尾的字串，傳回值爲 SHORT 整數，代表特殊鍵的虛擬鍵碼。如果按鍵字串與 g_SpecialKeys 表格中的每一個項目比對過之後，沒有任何一個相符的話，SendKeys_SpecialKeyToVirtKey 函式會檢查

看看按鍵字串是否爲單一字元。如果是的話，就傳回這單一字元的虛擬鍵碼。否則傳回 0，表示這個按鍵字串不能被轉換爲一個虛擬鍵碼。

如果找到了特殊鍵的虛擬鍵碼，SendKeys_PreprocessKeys 函式會檢查看看是否在右大括弧出現之前有指定重複次數。最後，虛擬鍵碼會被附加到 asVirtKeys 陣列的尾端。虛擬鍵碼會依「指定的重複次數」加入到陣列中。如果沒有指定重複次數的話，就僅加入一次。就在每個虛擬鍵碼被加入到 asVirtKeys 陣列之前，SendKeys_PreprocessKeys 會檢查虛擬鍵碼的高位元組，看看其最低位元是否設立。如果最低位元爲 1，表示字元轉換爲虛擬鍵碼的過程中，遺失了 shift 狀態。

舉個例子，如果字串中含有小寫的 s，將會產生虛擬鍵碼 VK_S。如果字串中含有大寫的 S，也會產生虛擬鍵碼 VK_S。當 VkKeyScan 函式回返，它會傳回一個字組（word），其中較低位元組記錄著虛擬鍵碼，較高位元組記錄著 shift 狀態旗標。 當 VkKeyScan 函式轉換大寫的 S 時，高位元組的第 0 個位元會被設爲 1。這表示當程式將虛擬鍵碼轉換回 ANSI 字元時，shift 鍵是必要的。由於 asVirtKeys 陣列中僅儲存虛擬鍵碼，沒有儲存 shift 狀態，所以 SendKeys_PreprocessKeys 函式必須將一個 VK_SHIFT 加入陣列之中。

這些都是 SendKeys 函式中有關於播放的部份。到目前爲止，我已爲每一個字元加入其對應的虛擬鍵碼到陣列之中。如果字串內含一個 J，就會有一個 VK_J 被我加到 asVirtKeys 陣列中。播放期間，必須有兩個 events 爲此按鍵碼產生出來：先是按鍵被壓下（keydown），然後是按鍵被放開按鍵（keyup）。但是你不能以同樣的方法來處理 Shift 鍵，如果你以同樣的方法來處理 Shift 鍵，則字串 +j 所產生的按鍵組合如下：

```
shift keydown
shift keyup
j keydown
j keyup
```

最後結果將是送出一個小寫的 j，而不是我們所預期的大寫 J。你必須以下列順序送出，才能達到我們的預期效果：

```
shift keydown
j keydown
j keyup
shift keyup
```

播放期間，asVirtKeys 陣列一次只會被走訪一個字元。如果陣列中的虛擬鍵碼不是一個移位鍵（shift-state key），journal playback filter function 第一次被呼叫時，會傳回此字元的 keydown 事件，第二次被呼叫時會傳回同一字元的 keyup 事件。

如果 journal playback hook 所看到的下一個要被播放的字元是一個移位鍵的話，它會傳回移位鍵的 keydown 事件，並設定一個旗標來提醒它自己說「移位鍵已被壓下且尚未被放開」。當要向 hook 請求較多的虛擬鍵碼時，它會檢查是否要傳回一個移位鍵。如果是的話，它會檢查自己的旗標，看看是否這個鍵已經被壓下。如果是的話，hook function 會傳回 Shift 鍵的 keyup 事件，並重置（reset）自己的旗標。

舉個例，如果你有一個字串是由 “^%(AB)” 所組成，當這個字串經由 SendKeys_PreprocessKeys 函式處理過後，asVirtKeys 陣列中所填入的內容會如**表 8.9** 所示。每一個字元經由 SendKeys_PreprocessKeys 函式剖析後，字元會被轉換爲對應的虛擬鍵碼，並被加入 asVirtKeys 陣列中。

表 8.9　asVirtKeys 陣列內容

索引值	虛擬鍵碼	說明
0	VK_CONTROL	VK_CONTROL 會被加入到陣列中，前處理器（preprocessor）會遞迴呼叫它自己（只有一級深度）。
1	VK_MENU	VK_MENU 會被加入到陣列中，前處理器（preprocessor）會遞迴呼叫它自己（二級深度）。然後解析器（parser）會看到左括弧並遞迴呼叫它自己（三級深度）。對左括弧來說，不會有任何東西被加入到陣列之中。
2	VK_A	加入 VK_A 到陣列中。
3	VK_B	加入 VK_B 到陣列中。
4	VK_MENU	前處理器（preprocessor）看到右括弧後，回返兩層。前處理器(preprocessor)再次回返，處理 %（一層）之後的 “token”。

索引值	虛擬鍵碼	說明
		回返時，前處理器（preprocessor）把另一個 VK_MENU 加到陣列之中。
5	VK_CONTROL	前處理器（preprocessor）回返之後，已經處理完畢在 ^（第 0 層）之後的 "token"。直到回返，前處理器（preprocessor）才會加入另一個 VK_CONTROL 到陣列之中。

是時候了，現在我們來討論虛擬鍵碼陣列是如何被取至系統中播放的。當 SendKeys 函式從 SendKeys_InitPreprocessKeys 函式取回控制權時，它會去呼叫 SendKeys_GetToken 函式，告訴它播放即將開始，並需做一些初始化動作。SendKeys_GetToken 函式會設定其內部旗標，表示目前沒有任何鍵被壓下，並設定讓播放從陣列中的第 0 項開始。

接著，SendKeys 函式會掛上 journal playback hook，並儲存 hook 代碼至 g_SKD 結構中的 hhook 欄位。Hook 被掛上之後，SendKeys 函式會呼叫 GetKeyboardState 函式，得到目前鍵盤的狀態，並將代表 Ctrl、Alt、Shift、Caps Lock、Num Lock 按鍵的位元關掉，接著呼叫 SetKeyboardState 函式，將這些設定值反應到執行緒的「同步按鍵狀態陣列」中。做這件事時務必小心，因爲使用者或許正按著 Shft 鍵，這會影響到它們被播放時的字元形式。

本章先前曾提過，當你想要切換各種按鍵狀態時，你應使用 keybd_event 函式。然而，在 SendKeys 函式中我卻可以使用 SetKeyboardState 函式達成相同效果。這是因爲當 journal playback hook 被掛上後，系統中所有執行緒的 local input state 變數會被繫結在一起 -- 這表示所有執行緒所看到的按鍵狀態資訊是相同的[4]。

現在，SendKeys 函式回返，journal playback hook 尚未被卸除。當 SendKeys 函式回返時，按鍵皆尚未被播放。執行緒應當在訊息迴路中取得按鍵訊息，使 playback hook 定期被呼叫起來播放按鍵。如果你想知道所有按鍵何時已被播放和處理完畢，你應該加入一個

[4] 本書第 6 章的 Capture 程式，對執行緒和其 local input state 有詳盡介紹。你同時也可參閱 ***Advanced Windows*** 中的 **"**Windows Messages and Asynchronous Input" 一章。

{QueueSync} 特殊鍵到你的按鍵字串尾端。當 playback hook 看到這個 {QueueSync}特殊鍵，它會 “post” 一個 WM_QUEUESYNC 訊息到你的視窗中。所有按鍵處理完畢後視窗函式會看到這個訊息。當收到 WM_QUEUESYNC 訊息，SKDemo 程式的主視窗會顯示一個訊息盒如圖 **8.3** 所示。

圖 8.3 SKDemo 視窗已收到 WM_QUEUESYNC 訊息

Journal playback hook 一旦被掛上，每當需要一個硬體事件，Windows 便會去呼叫其 filter functions。而 filter functions 會呼叫 SendKeys_GetToken 函式取得下一個事件的相關資訊。對於每一次呼叫，SendKeys_GetToken 函式皆會判斷下一個事件的虛擬鍵碼，看看這個事件是否是一個 keydown 事件，或 keyup 事件，同時也看看這個鍵是否是系統鍵。當 SendKeys_GetToken 函式回返，其傳回值如果是 TRUE，表示取得了一個事件。如果是 FALSE，表示陣列中的所有按鍵已被播完，沒有更多的 “token” 等候被處理。

當 hook function 藉由 SendKeys_GetToken 函式得知已經沒有 “token” 存在時，它會註銷 hook，並將 g_SKD 結構中的 hhook 變數設爲 NULL。如果 SendKeys_GetToken 函式傳回的是一個事件資訊的話，則 hook function 會準備一個 EVENTMSG 結構來存放 WM_(SYS)KEYDOWN/UP 或 WM_QUEUESYNC 訊息的相關資訊，並將這個事件傳回給 Windows。

SendKeys

SendKeys 函式會送出一個或多個按鍵給目前工作中（active）的視窗，就像是來自鍵盤的按鍵動作一樣：

```
SKERR WINAPI  SendKeys (HWND hwndQueueSyncTaget, LPCTSTR pszKeys);
```

參數

- hwndQueueSyncTarget：每當 {QueueSync} 特殊鍵要被播放時，此參數表示「將收到 WM_QUEUESYNC 訊息之視窗」的視窗代碼。如果這個值是 NULL，WM_QUEUESYNC 訊息會交給執行緒的訊息迴路來處理。
- pszKeys：指定一個文字字串，表示要被模擬的按鍵串列。

傳回值

這個函式會傳回表 **8.10** 列出值的其中之一，用以表示呼叫這個函式所獲得的結果。

表 8.10　SendKeys 函式的傳回值

傳回值	意義
SKERR_NOERROR	函式順利執行完畢。
SKERR_MISSINGCLOSEBRACE	找不到與左大括弧相配的右大括弧。
SKERR_INVALIDKEY	所使用的特殊鍵無法辨識。
SKERR_MISSINGCLOSEPAREN	找不到與左括弧相配的右括弧。
SKERR_INVALIDCOUNT	指定了一個不合法的重複次數。
SKERR_STRINGTOOLONG	所傳入的字串長度太長。
SKERR_CANTINSTALLHOOK	journal playback hook 無法掛上。

每一個按鍵是以一個或數個字元來表示。如果要表示單一鍵盤字元，我們使用字元本身來表示。舉個例，爲了表示 a 字母，pszKeys 應指向一個緩衝區，其內容看起來應該像“a”。如果你想要表示多個字元的話，需先將每一個附加字元加入到緩衝區中。如果要表示 a、b 和 c 字母的話，pszKeys 應指向一個緩衝區，其內容看起來應該像“abc”。加號（+）、逸出字元（^）和百分比符號（%）在這裡有特殊意義，爲了表示這些特殊字元，輸入字元必須放在大括弧內。舉個例，如果你要指定一個加號，應該使用 ”{+}” 表示；爲了送出“{“ 或 “}” 字元，必須分別使用 “{{}” 或 “{}}” 來表示。如果你所壓下的按鍵（像是 Enter 鍵或 Tab 鍵）不會顯示任何字元，或是你所壓下的按鍵表示的是一個

動作而不是一個字元的話，使用的碼如表 **8.11** 所示。

表 **8.11** 除了列出特殊按鍵之外，也列出一個 {QueueSync} 按鍵碼。當 {QueueSync} 按鍵被播放，並不會送出 WM_(SYS)KEY* 訊息，SendKeys 函式會送出 WM_QUEUESYNC 訊息給 hwndQueueSyncTarget 參數所指定的視窗。由於 SendKeys 函式會立即回返，所以在播放任何按鍵之前，程式會先找尋 WM_QUEUESYNC 訊息以判斷何時所有的鍵會被播放完畢。舉個例，以下這一行程式呼叫 SendKeys 函式，並且當所有按鍵都播放完畢後，hwndMain 視窗會收到一個 WM_QUEUESYNC 訊息：

```
SKERR SKErr = SendKeys(hwndMain, "Some string(QueueSync)");
```

一旦收到 WM_QUEUESYNC 訊息，你便獲得保證：所有在 {QueueSync} 這個 "token" 之前的按鍵都已播放並處理完畢。

表 8.11　SendKeys 函式所能辨認出的特殊按鍵碼

按鍵	碼	按鍵	碼	按鍵	碼
Backspace	{Backspace} 或 {Bs} 或 {Bksp}	Help	{Help}	Tab	{Tab}
Break	{Break}	Home	{Home}	F1 to F6	{F1} to {F6}
Caps Lock	{CapsLock}	Insert	{Insert}	Up Arrow	{Up}
Clear	{Clear}	Num Lock	{NumLock}	Down Arrow	{Down}
Delete	{Delete} 或 {Del}	Page Down	{PgDn}	Left Arrow	{Left}
End	{End}	Page Up	{PgUp}	Right Arrow	{Right}
Enter	{Enter} 或 ~	Print Screen	{PrtSc}		
Esc	{Escape} 或 {Esc}	Scroll Lock	{ScrollLock}		

你也可將按鍵與 Shift、Ctrl 和 Alt 鍵組合起來使用：在一般按鍵之前加上表 **8.12** 所示的特殊鍵碼即可。如欲指定 Shift、Ctrl 和（或）Alt 鍵被按下，並且直到數個鍵被按下後，才放開 Shift、Ctrl 和（或）Alt 鍵，那麼你必須使用括弧來區別。舉個例，按下 Shift 鍵，再按下 E 鍵，此時尚未放開 Shift 鍵，接著按下 C 鍵，那麼整個字串表示式應爲 "+(EC)"。如果先按下 Shift 鍵，再按下 E 鍵，然後放開 Shift 鍵，接著按下 C 鍵，那麼整個字串表示式爲 "+EC"。如欲指定重複鍵，必須使用 {按鍵 重複次數} 這種型式，其中按鍵與重複次數之間須以空白隔開。舉個例，"{left 42}" 表示按下左方向鍵 42 次，"{x 10}" 表示按下 x 字元 10 次。

表 8.12 SendKeys 函式所能辨識的特殊鍵碼

按鍵	特殊鍵碼
Shift	+
Ctrl	^
Alt	%

第 9 章

版本控制（Version Control）

從 Win32 軟體開發者的觀點來看，Windows 是由三個主要的動態連結函式庫所組成：User32.DLL、GDI32.DLL 和 Kernel32.DLL。這些 DLLs 分別負責 Windows 的使用者介面、圖形裝置介面和記憶體管理。每當有新的 Windows 版本出現，微軟公司就會加入一些新的功能（像是支援聲音和影像、物件聯結與內嵌（OLE）、對於 shell 的支援，以及遠端程序呼叫（remote procedure call）等等）。微軟公司並未將這些新功能的程式碼與 User32.DLL、GDI32.DLL 和 Kernel32.DLL 封裝在一起，而是另外產生 DLLs 來負責。例如 shell 的支援能力就被包含在 Shell32.DLL 函式庫中，OLE 的支援能力則分別被包含在 OLECli32.DLL 和 OLESvr32.DLL 兩個函式庫中。

如果你使用 Visual C++ 2.0 來撰寫 Win32 程式，你可能會聯結 MSVCRT20.DLL。這個 DLL 內含所有的 C-Runtime 函式，像是 sprintf 和 memset 等等。如果你用的是 Visual C++ 2.1 的話，你也會聯結一個 MSVCRT20.DLL 檔。這個檔雖與 2.0 版的檔名相同，實際內容卻不一樣。Visual C++ 小組允許你任意散播這個 DLLs。這似乎是個優點，但卻隱藏了不少問題。

我們來看一種最典型的情況。有一天有一個人到一家電腦公司購買了一套新的軟體「媒體小金剛」，這套軟體使用 Visual C++ 2.1 來編譯聯結，並配送給購買者一個 MSVCRT20.DLL 檔。當使用者安裝好這套軟體，這個 DLL 檔會被自動拷貝到你的硬碟去。六個月後，他又去一家電腦公司買了另一套軟體「神奇小畫家」，這套軟體是使用

Visual C++ 2.0 版來編譯聯結的，它也配送了一個 MSVCRT20.DLL 檔，但是這個 DLL 檔的版本比使用者原先安裝在硬碟上的那個 DLL 舊。現在，當使用者安裝這個剛買來的「神奇小畫家」時，會發生什麼事呢？

如果「神奇小畫家」將其舊版的 DLL 安裝在新版的 DLL 之上（覆蓋掉），則此後「媒體小金剛」恐怕就不能夠正常運作了。如果使用者是在過了三個月之後才去執行「媒體小金剛」，他會很奇怪爲什麼「媒體小金剛」本來好好的，現在卻不能執行了呢？微軟進行了大量的努力以保證 Visual C++ 2.1 所提供的 MSVCRT20.DLL 是 Visual C++ 2.0 版的 MSVCRT20.DLL 的完整的擴充集合（superset），意思是安裝較新版本的檔案，不應該會影響到舊版程式的執行。

另外還一個潛在性問題。通常，一個程式之中和語系相關的部份會被放在動態聯結函式庫（DLL）之中。然而廠商可能會將所有的 DLLs 檔名設爲相同的名稱，譬如說 Res.DLL。這表示英文版程式使用的是 Res.DLL，德文版程式也用的是 Res.DLL，依此類推。直到安裝時，安裝程式才根據使用者的喜好，決定安裝哪一個版本，並複製正確的 Res.DLL 版本到使用者的硬碟中。這種作法會對使用者造成不少困擾，因爲不同的程式可能會安裝不同版本的檔案，並覆蓋掉使用者先前所安裝的語系 DLLs。於是很可能使用者在德國欲執行一個已有四個月未曾用過的程式時，所有顯示出來的對話盒和選單皆是英文而不是德文 -- 因爲使用者安裝了其它語系的程式之故。

所有這些問題的答案是：使用 VERSIONINFO 資源。這項資源被放置在每一個執行檔或 DLL 檔的資源檔（RC）中，就像圖示（icon）、游標、對話盒面板（dialog template）、字串表格等等資源一樣。

爲什麼你應該在你的二進位檔案中含入一份版本資源呢？有幾個原因：第一，它能提供額外的資訊，像是使用者已安裝了何種語系的檔案，以及程式在何種電腦上須使用什麼樣的驅動程式或 DLLs 等等。第二，這些資訊可被用來做些診斷工作。如果使用者在程式的使用上遇到了某些問題，它可能會提供充分的原因。舉個例，如果有一個程式需要使用一個 3.0 版的 DLL，程式首先會去 DLL 的資源中讀取版本控制資訊，並在程式繼續

執行下去之前先判斷程式是否能夠正確地被執行。如果你的程式沒有能力自行檢查那些 DLL 的版本資訊，你可以利用某些工具來達成，像是 VerShow 程式（本章的範例程式），它可以檢查出版本資訊（譯註：包括 EXE、DLL、DRV…等）並做詳細的報告。

最後，你可以使用程式資源檔中的版本資訊來作爲你的 About 對話盒內容。我非常贊成你這麼做，因爲你只須更新你資源檔中的版本資訊即可，毋需同時對版本控制資源和 About 對話盒面板做修改。

以下是版本資源（VERSIONINFO）的一個範例：

```
// Version stamp for this DLL

#ifdef "winver.h"

#ifdef _DEBUG

// Version Info for MFC30[U]D.DLL
VS_VERSION_INFO    VERSIONINFO
  FILEVERSION     3,1,0,0
  PRODUCTVERSION   2,1,0,0
  FILEFLAGSMASK    VS_FFI_FILEFLAGSMASK
#ifndef RELEASE
  FILEFLAGS        VS_FF_DEBUG|VS_FF_PRERELEASE
#else
  FILEFLAGS        VS_FF_DEBUG
#endif
  FILEOS           VOS_NT_WINDOWS32
  FILETYPE         VFT_DLL
  FILESUBTYPE      0   // not used
BEGIN
   BLOCK "StringFileInfo"
   BEGIN
      BLOCK "040904E4" // Lang=US English, CharSet=Windows Multilual
      BEGIN
         VALUE "CompanyName",     "Microsoft Corporation\0"
         VALUE "FileDescription", "MFCDLL Shared Library - Debug Version\0"
         VALUE "FileVersion",    "3.1.000\0"
         VALUE "InternalName",    "MFCDLLD\0"
         VALUE "LegalCopyright",  "Copyright (C) Microsoft Corp. 1993-1994\0"
         VALUE "LegalTrademarks", "\0"
         VALUE "OriginalFilename","MFC30D.DLL\0"
```

```
        VALUE "ProductName",     "Microsoft (R) Visual C++\0"
        VALUE "ProductVersion",  "2.1.000\0"
      END
   END
   BLOCK "VarFileInfo"
   BEGIN
      VALUE "Translation", 0x409, 1252
        // English language (0x409) and the Windows ANSI codepage (1252)
   END
END

#else // RETAIL

// Version Info for MFC30[U].DLL
VS_VERSION_INFO    VERSIONINFO
  FILEVERSION     3,1,0,0
  PRODUCTVERSION   2,1,0,0
  FILEFLAGSMASK    VS_FFI_FILEFLAGSMASK
#ifndef RELEASE
  FILEFLAGS        VS_FF_PRERELEASE
#else
  FILEFLAGS        0 // Final version
#endif
  FILEOS        VOS_NT_WINDOWS32
  FILETYPE         VFT_DLL
  FILESUBTYPE      0   // Not used
BEGIN
   BLOCK "StringFileInfo"
   BEGIN
      BLOCK "040904E4" // Lang=US English, CharSet=Windows Multilual
      BEGIN
        VALUE "CompanyName",     "Microsoft Corporation\0"
        VALUE "FileDescription", "MFCDLL Shared Library - Retail Version\0"
        VALUE "FileVersion",    "3.1.000\0"
        VALUE "InternalName",    "MFCDLL\0"
        VALUE "LegalCopyright",  "Copyright (C) Microsoft Corp. 1993-1994\0"
        VALUE "LegalTrademarks", "\0"
        VALUE "OriginalFilename","MFC30.DLL\0"
        VALUE "ProductName",     "Microsoft (R) Visual C++\0"
        VALUE "ProductVersion",  "2.1.000"
      END
   END
   BLOCK "VarFileInfo"
   BEGIN
      VALUE "Translation", 0x409, 1252
        // English language (0x409) and the Windows ANSI codepage (1252)
```

```
    END
END

#endif
```

這份版本資源來自 Visual C++ 2.1 版的 MFC 函式庫所提供的 MFCDLL.RC 檔。你可能注意到在最開始的地方含入了一個 WinVer.H 檔。這個檔案非常重要，因爲它包含了 VERSIONINFO 資源中所使用到的識別字定義。資源的前半部份（從 FILEVERSION 到 FILESUBTYPE）屬於固定長度。表 **9.1** 中描述了那些欄位的意義。

表 9.1　VERSIONINFO 資源中各欄位的意義

欄位	意義
FILEVERSION	指定檔案的版本號碼，它由 4 個 16 位元整數所組成。如果你指定的整數不足 4 個，那麼資源編譯器會自動將其餘整數填爲 0。舉個例子，你可以指定 “3,51,0,61”，表示版本 3.51.0.61。
PRODUCTVERSION	指定產品的版本號碼，它內含 4 個 16 位元整數所組成的欄位，和 FILEVERSION 一樣。
FILEFLAGSMASK	指出 FILEFLAGS 欄位中哪一個位元是有意義的。這個值固定爲 VS_FFI_FILEFLAGSMASK（定義在 WinVer.H 表頭檔中，其值爲 0x0000003FL)。
FILEFLAGS	此欄位是由一組旗標 ”OR” 在一起所構成的。以下所列爲可能的旗標值：VS_FF_DEBUG、 VS_FF_INFOINFERRED、VS_FF_PATCHED、VS_FF_PRERELEASE、VS_FF_PRIVATEBUILD、VS_FF_SPECIALBUILD
FILEOS	指出「程式針對何種作業系統而設計」。以下所列爲可能的旗標值，定義在 WinVer.H 檔中：VOS_DOS_WINDOWS16、VOS_DOS_WINDOWS32、VOS_OS216_PM16、VOS_OS232_PM32、VOS_NT_WINDOWS32
FILETYPE	指定檔案類型。以下所列爲可能的識別字，定義在 WinVer.H 檔中：VFT_UNKNOWN、VFT_APP、VFT_DLL、VFT_DRV、VFT_FONT、VFT_VXD、VFT_STATIC_LIB
FILESUBTYPE	指定檔案的次類型。如果檔案類型不是 VFT_DRV 或 VFT_FONT 或 VFT_VXD 的話，這個欄位會被設爲 VFT2_UNKNOWN。如果檔案類型是 VFT_DRV 的話，這個欄位會是下列識別字之一：

欄位	意義
	VFT2_UNKNOWN、VFT2_DRV_PRINTER、VFT2_DRV_KEYBOARD、VFT2_DRV_LANGUAGE、VFT2_DRV_DISPLAY、VFT2_DRV_MOUSE、VFT2_DRV_NETWORK、VFT2_DRV_SYSTEM、VFT2_DRV_INSTALLABLE、VFT2_DRV_SOUND、VFT2_DRV_COMM、VFT2_DRV_INPUTMETHOD 如果檔案類型是 VFT_FONT 的話，這個欄位會是下列識別字之一： VFT2_UNKNOWN、VFT2_FONT_RASTER、VFT2_FONT_VECTOR、VFT2_FONT_TRUETYPE 如果檔案類型是 VFT_VXD 的話，這個欄位將是包含在虛擬裝置控制區塊中的虛擬裝置識別字。

緊跟在 VERSIONINFO 資源中「固定長度資料」之後的，是可變長度資料，而這一部份又可分爲 VarFileInfo 和 StringFileInfo 兩部份。

VerFileInfo 這部份描述資源的變數資料。目前，這部份變數資料僅有一種類型，即 "Translation"，表示檔案所支援的語系。緊跟在字串 "Translation" 之後的是一系列的「數對」（兩個一組的整數）。就拿先前所列的 VERSIONINFO 資源來說，「數對」的第一個數字是 0x0409（語系 ID），表示 StringFileInfo 中的資料應以何種語系顯現。其實語系 ID 是由一個主語系 ID 和一個次語系 ID 組成。**表 9.2** 列出目前定義在 WinNT.H 表頭檔中的主語系 ID 和次語系 ID。

表 9.2　主語系 ID 和次語系 ID

主語系 ID	主語系	次語系 ID	次語系
0x02	LANG_BULGARIAN 保加利亞語	0x01	SUBLANG_CHINESE_TRADITIONAL 中文繁體
0x04	LANG_CHINESE 中國（華語）	0x02	SUBLANG_CHINESE_SIMPLIFIED 中文簡體
0x1a	LANG_CROATIAN 克羅埃西亞語	0x03	SUBLANG_CHINESE_HONGKONG 香港
0x05	LANG_CZECH 捷克語	0x04	SUBLANG_CHINESE_SINGAPORE 新加坡
0x06	LANG_DANISH 丹麥語	0x01	SUBLANG_DUTCH 荷蘭

主語系 ID	主語系	次語系 ID	次語系
0x13	LANG_DUTCH　荷蘭語	0x02	SUBLANG_DUTCH_BELGIAN 比利時荷語
0x09	LANG_ENGLISH　英語	0x01	SUBLANG_ENGLISH_US　美國
0x0b	LANG_FINNISH	0x02	SUBLANG_ENGLISH_UK　英國
0x0c	LANG_FRENCH　法語	0x03	SUBLANG_ENGLISH_AUS　澳洲
0x07	LANG_GERMAN　德語	0x04	SUBLANG_ENGLISH_CAN　加拿大
0x08	LANG_GREEK　希臘語	0x05	SUBLANG_ENGLISH_NZ　紐西蘭
0x0e	LANG_HUNGARIAN 匈牙利語	0x06	SUBLANG_ENGLISH_EIRE　愛爾蘭
0x0f	LANG_ICELANDIC　冰島語	0x01	SUBLANG_FRENCH　法國
0x10	LANG_ITALIAN　意大利語	0x02	SUBLANG_FRENCH_BELGIAN　比利時法語
0x11	LANG_JAPANESE　日本語	0x03	SUBLANG_FRENCH_CANADIAN 加拿大法語
0x12	LANG_KOREAN　韓國語	0x04	SUBLANG_FRENCH_SWISS　瑞士法語
0x14	LANG_NORWEGIAN　挪威語	0x01	SUBLANG_GERMAN　德語
0x15	LANG_POLISH　波蘭語	0x02	SUBLANG_GERMAN_SWISS　瑞士德語
0x16	LANG_PORTUGUESE　葡萄牙語	0x03	SUBLANG_GERMAN_AUSTRIAN　奧地利德語
0x18	LANG_ROMANIAN　羅馬尼亞語	0x01	SUBLANG_ITALIAN　意大利語
0x19	LANG_RUSSIAN　俄語	0x02	SUBLANG_ITALIAN_SWISS　瑞士意語
0x1b	LANG_SLOVAK　斯洛法克語	0x01	SUBLANG_NORWEGIAN_BOKMAL
0x24	LANG_SLOVENIAN 斯洛維尼亞語	0x02	SUBLANG_NORWEGIAN_NYNORSK
0x0a	LANG_SPANISH　西班牙語	0x02	SUBLANG_PORTUGUESE　葡萄牙語
0x1d	LANG_SWEDISH　瑞典語	0x01	SUBLANG_PORTUGUESE_BRAZILIAN 巴西葡語
0x1f	LANG_TURKISH　土耳其語	0x01	SUBLANG_SPANISH　西班牙語
		0x02	SUBLANG_SPANISH_MEXICAN　墨西哥語
		0x03	SUBLANG_SPANISH_MODERN

你可以使用 MAKELANGID 巨集來建立一個語系 ID：

```
#define MAKELANGID(p, s) ((((WORD)(s)) << 10)｜(WORD)(p))
```

舉個例，欲建立美國英語（U.S English）語系 ID，作法如下：

```
LANGID langidUSEnglish = MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US);
```

此行被執行後，langidUSEnglish 變數的值將爲 0x0409[1]。

先前所列的 VERSIONINFO 資源的「數對」中的第二個數字是 1252，表示檔案資料所使用的字元集（character set）。對於 Win32 程式來說，你應使用 Code Page 爲 1200 的 Unicode，因爲資源編譯器所輸出的資源字串固定爲 Unicode。表 **9.3** 列出目前 Windows 所能辨識的字元集。

如果你想要顯示其它語系和（或）字元集的額外字串資訊，只須在 VERSIONINFO 資源的 "Translation" 之後再加入其它「數對」即可。舉個例，欲加入支援希臘語系和希臘字元集，作法如下：

```
VALUE "Translation", 0x0409, 1252, 0x0408, 1253
```

注意，字串 "Translation" 僅可以在 VarFileInfo 這一部份出現一次，且語系和字元集的資料應該總是成雙成對呈現才是。

[1] 請注意，MAKELANGID 巨集會將次語系 ID 向左移 10 個位元，而不是 8 個位元。

表 9.3　目前 Windows 所能辨識的字元集：

Code Page	說明
0	7-bit ASCII
932	Windows, Japan (Shift-JIS X-0208)
949	Windows, Korea (Shift-KSC 5601)
950	Windows, Taiwan (GB5)
1200	Unicode
1250	Windows, Latin-2 (Eastern European)
1251	Windows, Cyrillic
1252	Windows, Multilingual (ANSI)
1253	Windows, Greek
1254	Windows, Turkish
1255	Windows, Hebrew
1256	Windows, Arabic

在 VarFileInfo 這部份之後的是 StringFileInfo。這個部份是由以 0 結尾的字串所組成的，用來描述檔案中較細節的版本資訊。StringFileInfo 這一部份可被區分爲數個區塊，這些小區塊和 VarFileInfo 之中所指定的語系和字元集呈現一對一的關係。以前一個例子的 VERSIONINFO 實例來說，StringFileInfo 只有一個小區塊，且以字串 “040904E4” 做爲識別碼。字串的前 4 個字元（”0409”）表示語系 ID 的 16 進位值，0x0409 就是 U.S English。後 4 個字元（”04E4”）表示字元集的 16 進位值。就這個例子來說，16 進位的 0x04E4 等於 10 進位的 1252，表示 Windows Multilingual（ANSI）字元集。

這個區塊之內是檔案的字串資料。微軟公司爲了要讓這部份資料保有最佳彈性，所以決定利用字串名稱來標示每個可能的欄位。這個字串名稱會在關鍵字 VALUE 右邊出現。目前，共有 12 個可辨識的字串名稱，於**表 9.4** 中列出。

表 9.4　字串欄位名稱

字串名稱	必須嗎？	說明
"Comments"	N	額外的資訊，作爲診斷用。
"CompanyName"	Y	檔案製造者的公司名稱。
"FileDescription"	Y	給使用者看的檔案描述字串
"FileVersion"	Y	檔案的版本號碼。
"InternalName"	Y	檔案的內部名稱。
"LegalCopyright"	N	檔案的版權聲明。
"LegalTrademarks"	N	檔案所屬商品的註冊商標。
"OriginalFilename"	Y	檔案的原始檔名，避免使用者將其更名。
"PrivateBuild"	*	解釋這個版本爲何屬於非正式版本。
"ProductName"	Y	檔案所屬之產品名稱。
"ProductVersion"	Y	檔案所屬之版本號碼。
"SpecialBuild"	*	解釋這個版本與一般版本不同的地方。

當你想要建立一個字串區塊，並不須包含表 **9.4** 中所有的字串欄位。哪些是必須的？你可以從表中的「必須嗎？」這一欄得知。其中 "PrivateBuild" 和 "SpecialBuild" 兩項比較特殊，只有在位於固定長度資料區中的 FILEFLAGS 欄位所對應的位元爲 on 時（亦即設定了 VS_FF_PRIVATEBUILD 和 VS_FF_SPECIALBUILD 旗標），才須定義其對應欄位。至於這些字串出現的順序和大小寫，並無硬性規定。

定義字串區塊時，你必須注意，要在每一個字串的尾端放置一個 0 字元。舉個例：

```
VALUE "ProductName", "Microsoft (R) Visual C++\0"
```

你必須自行在字串尾端加入 "\0" ，因爲資源編譯器不像 C 語言編譯器一樣會自動在每一個字串的尾端加入 0 字元。然而如果你是使用 Visual C++ 編輯你的版本資源，Visual C++ 會自動幫你加入 0 字元。

當你使用資源編譯器來編譯 VERSIONINFO 資源時，資源編譯器會將所有的資訊連結起來到一個大記憶體區塊中。你可以使用 Version.DLL 函式庫所提供的各個函式來讀取記憶體區塊中的資訊，並剖析它。

每當程式要載入版本資訊到記憶體時，首先必須呼叫 GetFileVersionInfoSize 函式：

```
DWORD GetFileVersionInfoSize(LPTSTR lpszFile, lPDWORD lpdwHandle);
```

第一個參數爲包含有 VERSIONINFO 資源的檔案路徑名稱。第二個參數是一個指向 DWORD 變數的指標，其值通常爲 0[2]。函式傳回的是檔案版本資訊所佔的位元組數。如果找不到指定的檔案或檔案中沒有包含任何版本資訊的話，GetFileVersionInfoSize 函式會傳回 0。

一旦知道了版本資訊所佔的大小，接下來必須配置一塊足夠大的記憶體來存放這些資訊，然後便可以呼叫 GetFileVersionInfo 函式來取得版本資訊了：

```
BOOL GetFileVersionInfo(LPTSTR lpszFile, DWORD dwHandle,
   DWORD cbBuf, LPVOID lpvData);
```

這個函式會先將 VERSIONINFO 資源載入到你先前所配置的記憶體區塊中。其第一個參數與先前函式一樣，是你想要從中取得版本資訊的那個檔案的路徑名稱。第二個參數被忽略。第三個參數指定緩衝區的大小，此大小必須足以存放版本資訊，且不得小於先前由 GetFileVersionInfoSize 函式所傳回來的值。最後一個參數爲用來存放版本資訊的記憶體的位址。如果資料順利被載入，GetFileVersionInfo 函式會傳回 TRUE；否則它會傳回 FALSE。

---

[2] 這個函式的 16 位元版會傳回一個版本資源的 handle 值。Win32 並不需要這項資訊，但是當此函式移至 Win32 時，微軟公司並沒有改變此函式的原型，所以你可將此參數設爲 NULL。

一旦版本資訊被載入到記憶體後，接下來可以使用 VerQueryValue 函式取得包含在版本資訊中的片段資料。VerQueryValue 函式的原型宣告如下：

```
BOOL VerQueryValue(const LPVOID pBlock, LPTSTR lpSubBlock,
  LPVOID *lplpBuffer, PUINT puLen);
```

第一個參數記錄著版本資訊記憶體的位址，這個位址與 GetFileVersionInfo 函式的最後一個參數所指的位址相同。第二個參數 lpSubBlock，是一個指向以 0 結尾之字串的指標，其內容爲你想取得的資料，稍後我會再加以討論。第三個參數是一個指向 void 指標的指標。當 VerQueryValue 函式被呼叫時，它會搜尋記憶體區塊以求找出你想要的資料。如果找到，VerQueryValue 函式會將這份資訊所在的位址填入 void 指標中，然後你就可以利用這個位址來取得你所需要的資料了。最後一個參數 puLen，是一個指向未帶正負號之整數的指標 -- 如果函式找到了欲取得的資料，此一整數即被填入所取得的資訊的大小。舉個例，如果你要求 VerQueryValue 函式找出 "CompanyName" 資訊，並且如果該字串對應之資料爲 "ABC" 的話，VerQueryValue 函式會將 puLen 所指之整數設定爲 4（3 個字元再加上一個結尾 0 字元）。

如果找到了欲取得的資訊，VerQueryValue 函式會傳回一個非 0 值；否則傳回 0，表示欲取得的資訊不存在或是所指定的記憶體區塊不合法。

現在，讓我們再回去看看函式的第二個參數 lpSubBlock，此參數所指字串可爲三種形式之一。第一種形式是只包含了一個反斜線（\）字元的字串。舉個例，你可以下列方式來呼叫 VerQueryValue 函式：

```
PVOID pvVerInfo;
VS_FIXEDFILEINFO *pvsffi;
UINT uLen;
.
.
.
// Get the size of the version information.
uLen = GetFileVersionInfoSize(szPathname, NULL);

// Allocate a buffer to hold it.
```

```
pvVerInfo = malloc(uLen);

// Read the version information into the buffer.
GetFileVersionInfo(szPathname, NULL, uLen, pvVerInfo);

// Find the memory address where the fixed portion begins.
VerQueryValue(lpVerInfo, "\\", &pvsffi, &uLen);
```

這時候 VerQueryValue 函式會先找到版本資訊中「固定長度資料」之所在，並將記憶體區塊中該資訊之所在（記憶體位址）設定給 pvsffi 指標。這個指標是以一個 VS_FIXEDFILEINFO 結構（定義在 WinVer.H 檔中）的形式宣告出來的：

```
typedef struct _VS_FIXEDFILEINFO {  // vsffi
   DWORD   dwSignature;             // e.g., VS_FFI_SIGNATURE = 0xFEEE04BD
   DWORD   dwStrucVersion;          // e.g., VS_FFI_STRUCVERSION = 0x00010000
   DWORD   dwFileVersionMS;         // e.g., 0x00020000 = "2.0"
   DWORD   dwFileVersionLS;         // e.g., same as dwFileVersionMS
   DWORD   dwProductVersionMS;      // e.g., same as dwFileVersionMS
   DWORD   dwProductVersionLS;      // e.g., same as dwFileVersionMS
   DWORD   dwFileFlagsMask;         // e.g., VS_FFI_FILEFLAGSMASK = 0x0000003F
   DWORD   dwFileFlags;             // e.g., VS_FF_DEBUG | VS_FF_PRERELEASE
   DWORD   dwFileOS;                // e.g., VOS_DOS_WINDOWS32 (Windows 95)
   DWORD   dwFileType;              // e.g., VFT_DRIVER
   DWORD   dwFileSubtype;           // e.g., VFT2_DRV_KEYBOARD
   DWORD   dwFileDateMS;            // e.g., 0
   DWORD   dwFileDateLS;            // e.g., 0
} VS_FIXEDFILEINFO;
```

結構中的前兩個欄位 dwSignature 和 dwStrucVersion 的內容由資源編譯器在編譯時設定。版本控制函式就是利用這兩個欄位來確認它們運作的記憶體區塊所包含的版本資訊（透過 dwSignature）以及結構版本（透過 dwStrucVersion）。同樣地，結構中最後兩個欄位 dwFileDateMS 和 dwFileDateLS 的值也是由資源編譯器設定。目前的資源編譯器會將這些欄位設爲 0。結構中的其餘所有欄位會和 VERSIONINFO 資源中固定長度資料的欄位呈現一對一的對應關係。

一旦 VerQueryValue 函式將這個結構的位址指定給 pvsffi 變數之後，你就可以存取這結構中任何一個欄位的值，以取得與檔案版本有關的資訊。舉個例，如果你想要取得檔案

中是否有包含「偵錯資訊」，可以這樣做：

```
BOOL fHasDebugInfo = (pvsffi->dwFileFlags & VS_FF_DEBUG);
```

回到剛剛所說的參數形式。lpSubBlock 的第二種形式是指向一個包含有 "\VarFileInfo \Translation" 的字串。舉個例，如果你以下列方式來呼叫 VerQueryValue 函式：

```
PDWORD pdwTranslationInfo;
UNIT uLen, uCharSetID;
LANGID langid;
.
.
.
VerQueryValue(pvVerInfo, "\\VarFileInfo\\Translation",
   (PVOID *) &pdwTranslationInfo, &uLen);

langid = LOWORD(*pdwTranslationInfo);
uCharSetId = HIWORD(*pdwTranslationInfo);
```

那麼 VerQueryValue 首先會找到版本資源資訊中 VarFileInfo 區段所包含的 Translation 欄位的所在，並將所在位址指定給 pdwTranslationInfo 變數。當 VerQueryValue 函式返回，其 pdwTranslationInfo 參數會指向語系和字元集的「數對」，每一個「數對」的低位元組記錄著語系 ID，高位元組爲字元集 ID。當 VerQueryValue 函式回返，uLen 變數會被設定爲 Translation 欄位長度的總位元組數，所以你只須將這個值除以 4，就可得知此欄位有多少組「數對」了。

如果想要以字串形式來顯示這份語系資訊的話，你可以使用 VerLanguageName 函式：

```
DWORD VerLanguageName(DWORD idLang, LPTSTR lpszLang, DWORD cbLang);
```

函式的第一個參數爲語系 ID，可從 Translation 欄位中取得。第二個參數爲一個指向緩衝區的指標，當函式回返，緩衝區內會記錄著代表此語系的字串。最後一個參數爲緩衝區的最大長度。舉個例，如果 idLang 參數被設爲 0x0814，則 lpszLang 緩衝區將會被填入

一個 “Norwegian-Nynorsk” 字串。如果所傳入的語系 ID 不能被辨識的話，VerLanguageName 函式會將 “Language-Neutral” 字串填入緩衝區中。注意，這個函式會傳回「複製到緩衝區中的字串的長度」，所以即使是一個不能被辨認的語系 ID，函式也會傳回 “Norwegian-Nynorsk” 字串的長度，而且不會給你任何錯誤資訊。

lpSubBlock 參數的最後一種形式是指向一個包含有 “\StringFileInfo\Lang-CharSet\StringName” 的字串。如果是這種形式，VerQueryValue 函式會自版本資源的 StringFileInfo 區段中的一個小區段找尋一個特定語系的字串所在。Lang-CharSet 爲一個「語系與字元集的 ID 數對」，可由先前呼叫 VerQueryValue 函式時自 Translation 欄位中取得，它必須是 16 進位值。整個字串的最後一部份是 StringName，指出表 **9.4** 所列出的字串名稱之一。舉個例，如果你以下列方式來呼叫 VerQueryValue 函式：

```
PDWORD pdwTranslationInfo;
UNIT uLen;
char szStringFileInfo[50];
LPCSTR szCompanyName;
.
.
.
VerQueryValue(pvVerInfo, "\\VarFileInfo\\Translation",
  (PVOID *) &pdwTranslationInfo, &uLen);

wsprintf(szStringFileInfo, "\\StringFileInfo\\%04x%04x\\CompanyName",
  LOWORD(*pdwTranslationInfo), HIWORD(*pdwTranslationInfo));

VerQueryValue(pvVerInfo, szStringFileInfo,
  (PVOID *) &szCompanyName, &uLen);
```

VerQueryValue 函式首先會找出 Translation 中的第一組數對，並在其所指定的語系區塊中找出欄位 “CompanyName” 字串資料之所在，再將其所在位址指定給 szCompanyName 變數。

VerShow 程式

VerShow 程式（Vershow.EXE）的原始碼顯示於**程式 列表 9.1 ~ 9.3** 中，示範如何摘錄出含在檔案之中的版本資訊。當你執行此程式，其主視窗如**圖 9.1** 所示。

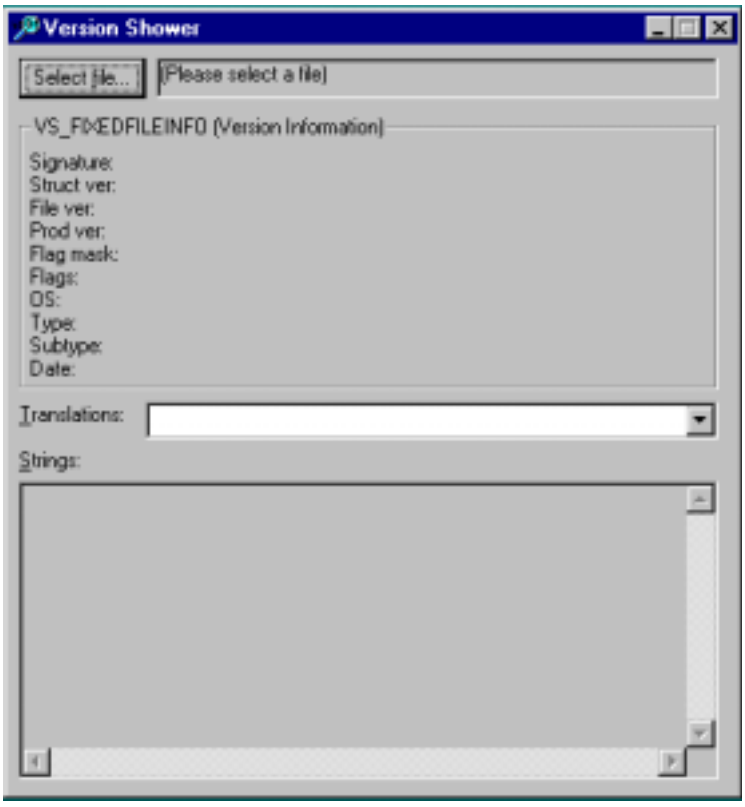


圖 9.1　VerShow 程式的外觀

爲了檢閱含在檔案內的版本資訊，你首先必須按下【Select File】鈕來選取一個檔案。按下之後會顯現一個【開啓舊檔】對話盒，讓你選取一個你想要檢閱其版本資訊的檔案。爲了使檔案的選取更容易些，故對話盒中的「檔案類型」區只列出可能含入「版本控制

資訊」的檔案類型：可執行檔（EXEs）、動態連結函式庫（DLLs）、裝置驅動程式（DRVs）、字形檔（FONs）、虛擬裝置驅動程式（VxDs）和靜態連結函式庫（LIBs）。

選取檔案之後，VerShow 程式會顯示你所選取的檔案的版本資訊。如果你選取的檔案是 VerShow.EXE 自己，它將顯示出圖 **9.2** 的資訊。

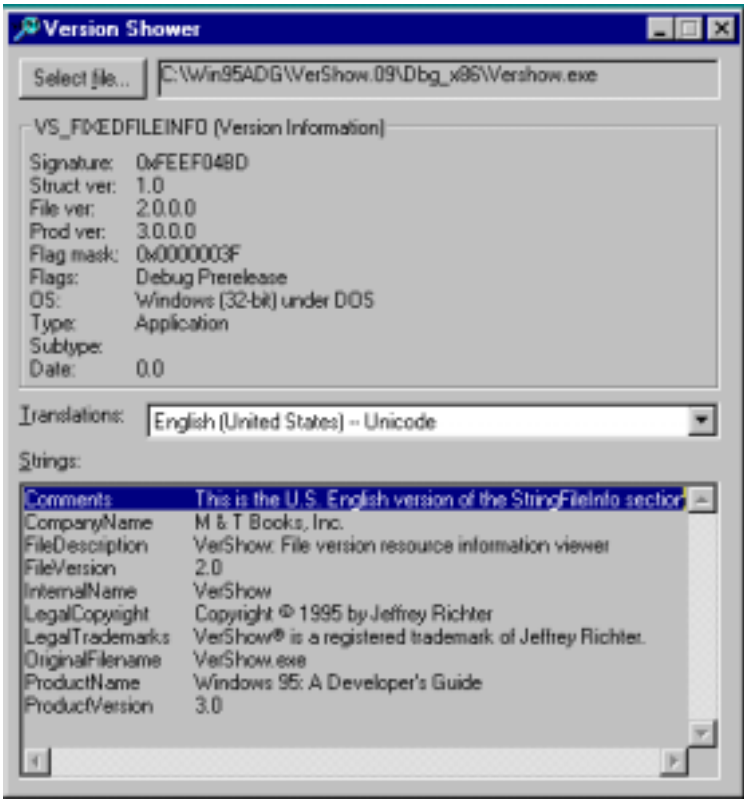


圖 9.2　VerShow 程式的「檔案資源」資訊

圖 **9.2** 的 “Translations” combobox 會列出版本資訊中所有可利用的語系。VerShow 程式本身在版本資訊中包含了三種不同語系。如果你選擇 German，VerShow 的 ”Strings” listbox 會改變爲德語版的版本資訊（取自其 StringFileInfo 區段），如圖 **9.3** 所示。

不幸地是，大部份公司並未將版本資訊放入到檔案內。這並不是一個好習慣，因爲這麼一來程式就不能辨識這些檔案的版本資訊，於是可能造成在安裝過程中有舊版檔案覆蓋了新版檔案。如果你要求 VerShow 程式檢閱一個沒有包含任何版本資訊的檔案，會出現如圖 9.4 的畫面。

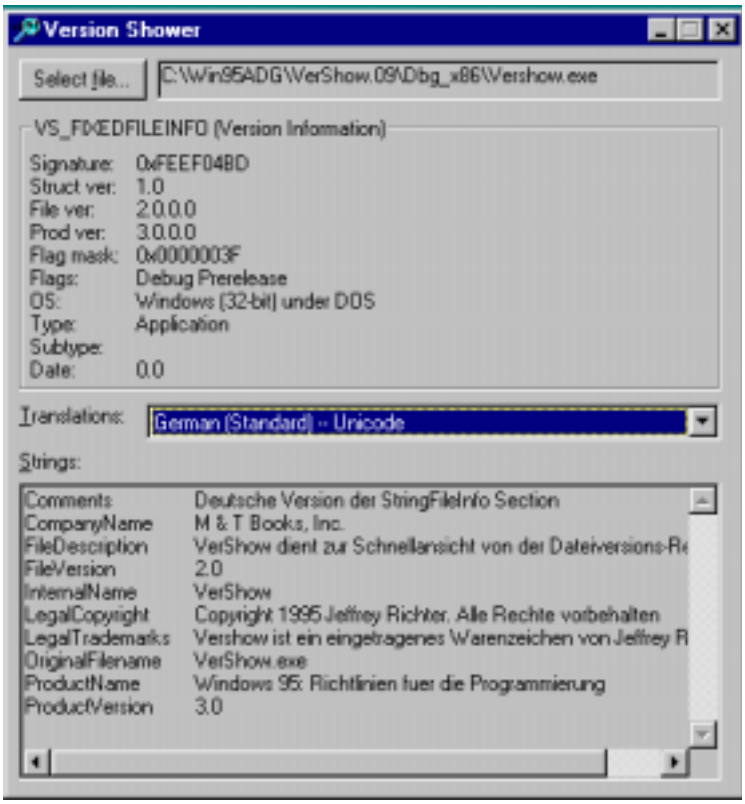


圖 9.3　德語系資源

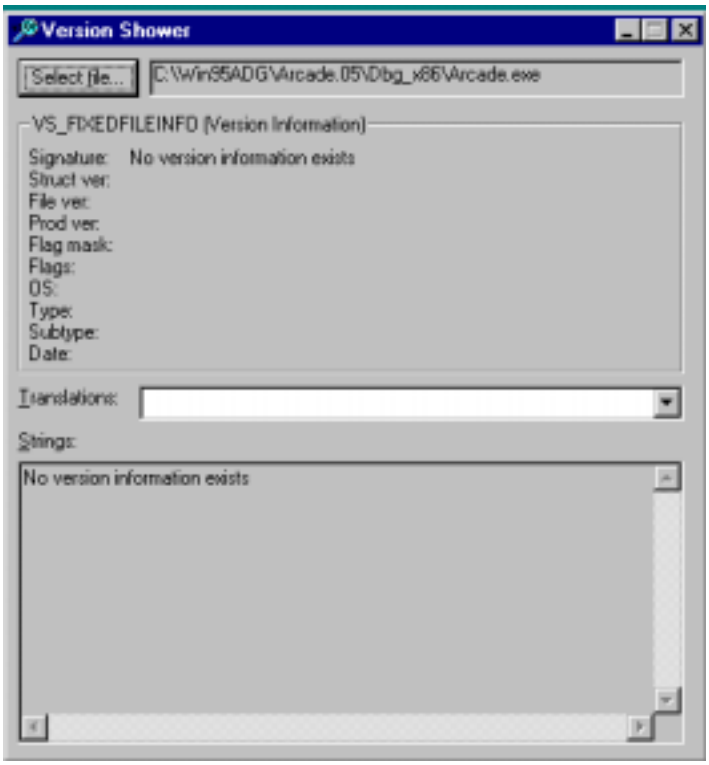


圖 9.4　一個沒有任何版本資訊的檔案

Windows 95 的 shell 也支援了某些程度的版本資源資訊檢驗能力。如果你要求 shell 檢閱一個包含「版本資訊」的檔案的 “properties”（譯註：中文版 Windows 95 把 “properties” 譯爲「內容」），出現的對話盒中會有一個 Version 附頁。**圖 9.5** 就是 shell 檢閱 VerShow.EXE 的 “properties” 時出現的對話盒畫面。

不幸的是，這個對話盒僅能列出英語版的版本資訊，不能去看其它語系的版本資訊。

譯註：原書說「VerShow 只能列出英文版的版本資訊」，這是錯誤的，事實上它只能列出德文版的版本資訊。這是因爲 shell 程式所檢閱的版本資訊是各字串之中值最小者。舉個例，如果版本字串爲 "040704e4" 和 "040904e4"，則 shell 程式會顯示 "040704e4" 的版本資訊。

建立 VerShow 程式所需用到的檔案列在表 **9.5** 中。

譯註：如果你詳細檢查 VerShow.RC，你會發現其中 #0014 行含入一個 VerShow.h 檔。但表 **9.5** 沒有列出，碟片中也沒有，不是很奇怪嗎？透過 email 詢問作者 Jeff，他說：

```
> VerShow.h is not required and should be removed from VerShow.RC
```

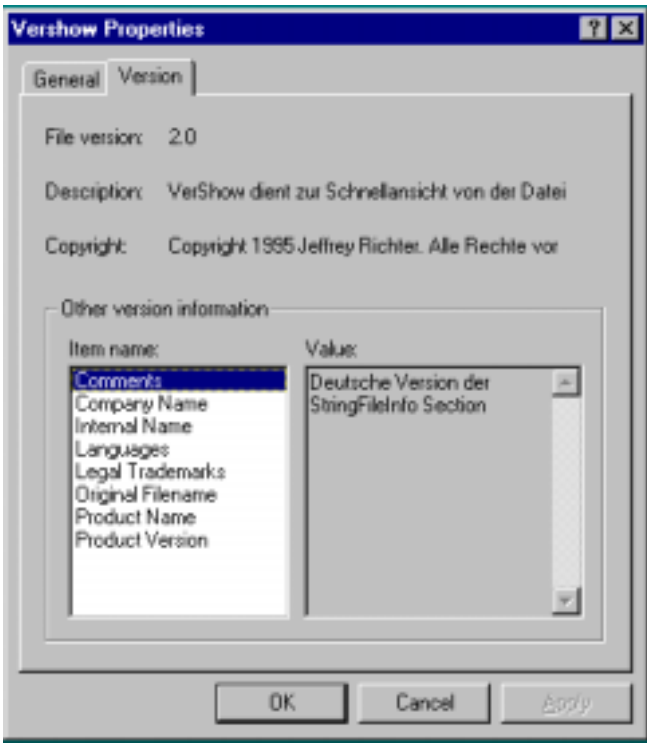


圖 9.5　Windows 95 shell 檢閱「版本資訊」

表 9.5　建立 VerShow 程式所需用到的檔案

檔案	說明
VerShow.C	內含 WinMain 函式、對話盒函式及訊息剖析函式。
VerShow.RC	內含主視窗的對話盒面板（template）和圖示（icon）。
Resource.H	內含 VerShow.RC 檔中所有資源的 ID。
VerShow.ICO	主視窗圖示（icon）。
VerShow.MAK	Visual C++ 的 MAK 檔。

VerShow.ICO

程式列表 9.1　VerShow.C

```
#0001  /************************************************************
#0002  Module name: VerShow.c
#0003  Written by: Jeffrey Richter
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Version resource shower
#0006  ************************************************************/
#0007
#0008
#0009  #include "..\Win95ADG.h"        /* See Appendix A for details. */
#0010  #include <windows.h>
#0011  #include <windowsx.h>
#0012  #pragma warning(disable: 4001)    /* Single-line comment */
#0013  #include <tchar.h>
#0014  #include "resource.h"
#0015
#0016
#0017  ////////////////////////////////////////////////////////////////
#0018
#0019
#0020  // The Version functions require the Version.lib library.  Because
#0021  // VC++ 2.x doesn't link with Version.lib by default, I must force it.
#0022  #pragma comment(lib, "Version.lib")
#0023
#0024
#0025  ////////////////////////////////////////////////////////////////
#0026
#0027
#0028  #define  VERBUFSIZE  2048
```

```
#0029
#0030
#0031 ////////////////////////////////////////////////////////////
#0032
#0033
#0034 typedef struct {
#0035    DWORD dwID;
#0036    LPTSTR szName;
#0037 } VALUELIST;
#0038
#0039
#0040 const VALUELIST g_vlFileType[] = {
#0041    { VFT_APP,         __TEXT("Application")        },
#0042    { VFT_DLL,         __TEXT("Dynamic-link library") },
#0043    { VFT_DRV,         __TEXT("Device driver")       },
#0044    { VFT_FONT,        __TEXT("Font")              },
#0045    { VFT_VXD,         __TEXT("Virtual device")      },
#0046    { VFT_STATIC_LIB,   __TEXT("Static library")      },
#0047    { 0,             NULL                      }
#0048 };
#0049
#0050 const VALUELIST g_vlFileOSAppType[] = {
#0051    { VOS__WINDOWS16,    __TEXT("Windows (16-bit)")    },
#0052    { VOS__PM16,        __TEXT("PM (16-bit)")        },
#0053    { VOS__PM32,        __TEXT("PM (32-bit)")        },
#0054    { VOS__WINDOWS32,    __TEXT("Windows (32-bit)")    },
#0055    { 0,              NULL                       }
#0056 };
#0057
#0058 const VALUELIST g_vlFileOSPlatform[] = {
#0059    { VOS_DOS,     __TEXT("DOS")         },
#0060    { VOS_OS216,   __TEXT("OS/2 (16-bit)") },
#0061    { VOS_OS232,   __TEXT("OS/2 (32-bit)") },
#0062    { VOS_NT,     __TEXT("Windows NT")   },
#0063    { 0,          NULL                }
#0064 };
#0065
#0066 const VALUELIST g_vlDriverSubType[] = {
#0067    { VFT2_DRV_PRINTER,      __TEXT("Printer")       },
#0068    { VFT2_DRV_KEYBOARD,     __TEXT("Keyboard")      },
#0069    { VFT2_DRV_LANGUAGE,     __TEXT("Language")      },
#0070    { VFT2_DRV_DISPLAY,      __TEXT("Display")       },
#0071    { VFT2_DRV_MOUSE,        __TEXT("Mouse")        },
#0072    { VFT2_DRV_NETWORK,      __TEXT("Network")       },
#0073    { VFT2_DRV_SYSTEM,       __TEXT("System")        },
#0074    { VFT2_DRV_INSTALLABLE,   __TEXT("Installable")   },
```

```
#0075    { VFT2_DRV_SOUND,        __TEXT("Sound")        },
#0076    { VFT2_DRV_COMM,         __TEXT("Comm")         },
#0077    { VFT2_DRV_INPUTMETHOD,  __TEXT("Input method")  },
#0078    { 0,                  NULL                  }
#0079 };
#0080
#0081 const VALUELIST g_vlFontSubType[] = {
#0082    { VFT2_FONT_RASTER,    __TEXT("Raster")    },
#0083    { VFT2_FONT_VECTOR,    __TEXT("Vector")    },
#0084    { VFT2_FONT_TRUETYPE,  __TEXT("TrueType")  },
#0085    { 0,                NULL              }
#0086 };
#0087
#0088
#0089 ////////////////////////////////////////////////////////////////
#0090
#0091
#0092 // This function appends the text names of the styles to a string buffer. It
#0093 // is used for the class styles, window styles, and extended window styles.
#0094 LPCTSTR VerShow_FindValueString (const VALUELIST Values[], DWORD dwValue) {
#0095
#0096    int nValueIndex;
#0097
#0098    for (nValueIndex = 0; Values[nValueIndex].szName != NULL; nValueIndex++) {
#0099       if (Values[nValueIndex].dwID == dwValue) {
#0100
#0101          // If we found the value, return the address of the string.
#0102          return(Values[nValueIndex].szName);
#0103       }
#0104    }
#0105    return(__TEXT("Unknown"));    // We didn't find the value.
#0106 }
#0107
#0108
#0109 ////////////////////////////////////////////////////////////////
#0110
#0111
#0112 void VerShow_ConstructFixedStr (LPTSTR szBuf, PVOID pVerInfo) {
#0113
#0114    VS_FIXEDFILEINFO *pvsffi;
#0115    UINT uLen;
#0116    TCHAR szFlags[1024], szVxdID[20];
#0117    LPCTSTR szSubType = __TEXT("");
#0118    DWORD dwTemp;
#0119
#0120    if (pVerInfo == NULL) {
```

```
#0121       lstrcpy(szBuf, __TEXT("No version information exists"));
#0122       return;
#0123    }
#0124
#0125    // Get the address to the VS_FIXEDFILEINFO structure data.
#0126    VerQueryValue(pVerInfo, __TEXT("\\"), (PVOID *) &pvsffi, &uLen);
#0127
#0128    // Create the file flag string.
#0129    szFlags[0] = 0;
#0130    dwTemp = pvsffi->dwFileFlags;
#0131    if (dwTemp & VS_FF_DEBUG)       lstrcat(szFlags, __TEXT("Debug "));
#0132    if (dwTemp & VS_FF_PRERELEASE)   lstrcat(szFlags, __TEXT("Prerelease "));
#0133    if (dwTemp & VS_FF_PATCHED)     lstrcat(szFlags, __TEXT("Patched "));
#0134    if (dwTemp & VS_FF_PRIVATEBUILD) lstrcat(szFlags, __TEXT("PrivateBuild "));
#0135    if (dwTemp & VS_FF_INFOINFERRED) lstrcat(szFlags, __TEXT("InfoInferred "));
#0136    if (dwTemp & VS_FF_SPECIALBUILD) lstrcat(szFlags, __TEXT("SpecialBuild "));
#0137
#0138    // Determine the file subtype.
#0139    switch (pvsffi->dwFileType) {
#0140       case VFT_DRV:
#0141          szSubType = VerShow_FindValueString(g_vlDriverSubType,
#0142             pvsffi->dwFileSubtype);
#0143          break;
#0144
#0145       case VFT_FONT:
#0146          szSubType = VerShow_FindValueString(g_vlFontSubType,
#0147             pvsffi->dwFileSubtype);
#0148          break;
#0149
#0150       case VFT_VXD:
#0151          wsprintf(szVxdID, __TEXT("%ld"), pvsffi->dwFileSubtype);
#0152          szSubType = szVxdID;
#0153          break;
#0154
#0155       case VFT_APP: case VFT_DLL: case VFT_STATIC_LIB: default:
#0156          szSubType = __TEXT("");
#0157          break;
#0158    }
#0159
#0160    wsprintf(szBuf,
#0161       __TEXT("0x%08lX\n")      // Signature
#0162       __TEXT("%d.%d\n")        // Structure version
#0163       __TEXT("%d.%d.%d.%d\n")   // File version
#0164       __TEXT("%d.%d.%d.%d\n")   // Product version
#0165       __TEXT("0x%08lX\n")      // File flags mask
#0166       __TEXT("%s\n")           // File flags
```

```
#0167       __TEXT("%s under %s\n")   // File OS
#0168       __TEXT("%s\n")           // File type
#0169       __TEXT("%s\n")           // File subtype
#0170       __TEXT("%d.%d"),         // File date converted to something
#0171
#0172       pvsffi->dwSignature,
#0173       HIWORD(pvsffi->dwStrucVersion),    LOWORD(pvsffi->dwStrucVersion),
#0174
#0175       HIWORD(pvsffi->dwFileVersionMS),   LOWORD(pvsffi->dwFileVersionMS),
#0176       HIWORD(pvsffi->dwFileVersionLS),   LOWORD(pvsffi->dwFileVersionLS),
#0177
#0178       HIWORD(pvsffi->dwProductVersionMS), LOWORD(pvsffi->dwProductVersionMS),
#0179       HIWORD(pvsffi->dwProductVersionLS), LOWORD(pvsffi->dwProductVersionLS),
#0180       pvsffi->dwFileFlagsMask,          szFlags,
#0181
#0182       VerShow_FindValueString(g_vlFileOSAppType,
#0183          pvsffi->dwFileOS & 0x0000FFFF),
#0184       VerShow_FindValueString(g_vlFileOSPlatform,
#0185          pvsffi->dwFileOS & 0xFFFF0000),
#0186       VerShow_FindValueString(g_vlFileType, (UINT) pvsffi->dwFileType),
#0187       szSubType,
#0188       pvsffi->dwFileDateMS,             pvsffi->dwFileDateLS);
#0189  }
#0190
#0191
#0192  ////////////////////////////////////////////////////////////////
#0193
#0194
#0195  void VerShow_ConstructVariableStr (LPTSTR szBuf, PVOID pVerInfo,
#0196    UINT uTranslationNum) {
#0197
#0198    TCHAR szFieldPath[200];
#0199    LPCTSTR szData, const *szField;
#0200    DWORD *pdwTranslation;
#0201    UINT uLen;
#0202    static const LPCTSTR g_szFields[] = {
#0203       __TEXT("Comments"),
#0204       __TEXT("CompanyName"),
#0205       __TEXT("FileDescription"),
#0206       __TEXT("FileVersion"),
#0207       __TEXT("InternalName"),
#0208       __TEXT("LegalCopyright"),
#0209       __TEXT("LegalTrademarks"),
#0210       __TEXT("OriginalFilename"),
#0211       __TEXT("PrivateBuild"),
#0212       __TEXT("ProductName"),
```

```
#0213       __TEXT("ProductVersion"),
#0214       __TEXT("SpecialBuild"),
#0215       NULL
#0216    };
#0217
#0218    if (pVerInfo == NULL) {
#0219       lstrcpy(szBuf, __TEXT("No version information exists"));
#0220       return;
#0221    }
#0222
#0223    szBuf[0] = 0;  // Clears the contents of the string
#0224
#0225    VerQueryValue(pVerInfo, __TEXT("\\VarFileInfo\\Translation"),
#0226       (PVOID *) &pdwTranslation, &uLen);
#0227    pdwTranslation += uTranslationNum;
#0228
#0229    for (szField = &g_szFields[0]; *szField != NULL; szField++) {
#0230       wsprintf(szFieldPath, __TEXT("\\StringFileInfo\\%04x%04x\\%s"),
#0231          LOWORD(*pdwTranslation), HIWORD(*pdwTranslation), *szField);
#0232
#0233       // NOTE: The cast to (LPTSTR) is necessary because VerQueryValue
#0234       //       is prototyped incorrectly in WINVER.h.
#0235       if (VerQueryValue(pVerInfo, (LPTSTR) szFieldPath,
#0236             (PVOID *) &szData, &uLen) && (uLen > 0)) {
#0237
#0238          wsprintf(_tcschr(szBuf, 0), __TEXT("%s\t%s\r\n"), *szField, szData);
#0239       }
#0240    }
#0241 }
#0242
#0243
#0244 ////////////////////////////////////////////////////////////////
#0245
#0246
#0247 void VerShow_InitFileVerInfo (HWND hwnd, LPCTSTR pszPathname) {
#0248
#0249    DWORD dwVerSize, *pdwTranslation;
#0250    PVOID pVerInfo;
#0251    UINT uLen, x;
#0252    TCHAR szBuf[VERBUFSIZE];
#0253
#0254    // If the user was already viewing a file's resource data, clean up.
#0255    pVerInfo = (PVOID) GetWindowLong(hwnd, GWL_USERDATA);
#0256    if (pVerInfo != NULL) {
#0257       free(pVerInfo);
#0258       SetWindowLong(hwnd, GWL_USERDATA, (LONG) (pVerInfo = NULL));
```

```
#0259    }
#0260
#0261    SetDlgItemText(hwnd, IDC_PATHNAME, pszPathname);
#0262    if (pszPathname == NULL) {
#0263
#0264       // If a NULL path name is passed, we should just clean up.
#0265       return;
#0266    }
#0267
#0268    // Calculate the size of the version control resource information.
#0269    // NOTE: The cast to (LPTSTR) is necessary because GetFileVersionInfoSize
#0270    //       is prototyped incorrectly in WINVER.h.
#0271    dwVerSize = GetFileVersionInfoSize((LPTSTR) pszPathname, NULL);
#0272    if (dwVerSize > 0) {
#0273
#0274       // Allocate a memory block large enough to hold the version information.
#0275       if ((pVerInfo = malloc(dwVerSize)) != NULL) {
#0276
#0277          // Load the version information into memory.
#0278          // NOTE: The cast to the following (LPTSTR) is necessary because
#0279          //       GetFileVersionInfo is prototyped incorrectly in WINVER.h.
#0280          GetFileVersionInfo((LPTSTR) pszPathname, 0, dwVerSize, pVerInfo);
#0281          SetWindowLong(hwnd, GWL_USERDATA, (LONG) pVerInfo);
#0282       } else {
#0283          adgMB(__TEXT("Insufficient memory."));
#0284       }
#0285    }
#0286
#0287
#0288    // Update the VS_FIXEDFILEINFO information.
#0289    VerShow_ConstructFixedStr(szBuf, pVerInfo);
#0290    SetDlgItemText(hwnd, IDC_FIXEDFILEINFODATA, szBuf);
#0291
#0292    // Update the Translations combobox.
#0293    ComboBox_ResetContent(GetDlgItem(hwnd, IDC_TRANSLATIONS));
#0294    if (pVerInfo != NULL) {
#0295
#0296       VerQueryValue(pVerInfo, __TEXT("\\VarFileInfo\\Translation"),
#0297          (PVOID *) &pdwTranslation, &uLen);
#0298       for (x = 0; x < uLen; x += 4, pdwTranslation++) {
#0299
#0300          // Get the string name for the language number.
#0301          VerLanguageName(LOWORD(*pdwTranslation), szBuf, adgARRAY_SIZE(szBuf));
#0302          lstrcat(szBuf, __TEXT(" -- "));
#0303
#0304          // Get the string name for the code page number.
```

```
#0305          LoadString(GetWindowInstance(hwnd), HIWORD(*pdwTranslation),
#0306             _tcschr(szBuf, 0),
#0307             (UINT) (adgARRAY_SIZE(szBuf) - (_tcschr(szBuf, 0) - szBuf)));
#0308
#0309          // Add the string to the combobox.
#0310          ComboBox_AddString(GetDlgItem(hwnd, IDC_TRANSLATIONS), szBuf);
#0311       }
#0312       // Select the first translation entry as the default.
#0313       ComboBox_SetCurSel(GetDlgItem(hwnd, IDC_TRANSLATIONS), 0);
#0314    }
#0315
#0316    // Update the string information for the default translation.
#0317    VerShow_ConstructVariableStr(szBuf, pVerInfo, 0);
#0318    SetDlgItemText(hwnd, IDC_STRINGS, szBuf);
#0319 }
#0320
#0321
#0322 ////////////////////////////////////////////////////////////////
#0323
#0324
#0325 BOOL VerShow_OnInitDialog (HWND hwnd, HWND hwndFocus, LPARAM lParam) {
#0326
#0327    adgSETDLGICONS(hwnd, IDI_VERSHOW, IDI_VERSHOW);
#0328    return(TRUE);               // Accepts default focus window.
#0329 }
#0330
#0331
#0332 ////////////////////////////////////////////////////////////////
#0333
#0334
#0335 void VerShow_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) {
#0336    OPENFILENAME ofn;
#0337    TCHAR szPathname[128], szBuf[VERBUFSIZE];
#0338
#0339    switch (id) {
#0340
#0341       case IDCANCEL:            // Allows dialog box to close
#0342          VerShow_InitFileVerInfo(hwnd, NULL);
#0343          EndDialog(hwnd, 0);
#0344          break;
#0345
#0346       case IDC_SELECTFILE:
#0347          adgINITSTRUCT(ofn, TRUE);
#0348          ofn.hwndOwner = hwnd;
#0349          ofn.lpstrFilter =
#0350             __TEXT("Executables (*.exe)\0*.EXE\0")
```

```
#0351           __TEXT("DLLs (*.dll)\0*.DLL\0")
#0352           __TEXT("Device drivers (*.drv)\0*.DRV\0")
#0353           __TEXT("Fonts (*.fon)\0*.FON\0")
#0354           __TEXT("Virtual devices (*.386)\0*.386\0")
#0355           __TEXT("Static libraries (*.lib)\0*.LIB\0")
#0356           __TEXT("All files (*.*)\0*.*\0");
#0357         ofn.lpstrFile = szPathname; ofn.lpstrFile[0] = 0;
#0358         ofn.nMaxFile = adgARRAY_SIZE(szPathname);
#0359         ofn.Flags = OFN_HIDEREADONLY | OFN_FILEMUSTEXIST | OFN_EXPLORER;
#0360         if (GetOpenFileName(&ofn)) {
#0361           VerShow_InitFileVerInfo(hwnd, ofn.lpstrFile);
#0362         }
#0363         break;
#0364
#0365       case IDC_TRANSLATIONS:
#0366         if (codeNotify == CBN_SELCHANGE) {
#0367
#0368           // Update the string information for the selected translation.
#0369           VerShow_ConstructVariableStr(szBuf,
#0370             (PVOID) GetWindowLong(hwnd, GWL_USERDATA),
#0371             ComboBox_GetCurSel(hwndCtl));
#0372           SetDlgItemText(hwnd, IDC_STRINGS, szBuf);
#0373         }
#0374         break;
#0375     }
#0376 }
#0377
#0378
#0379 ////////////////////////////////////////////////////////////////
#0380
#0381
#0382 BOOL WINAPI VerShow_DlgProc (HWND hwnd, UINT uMsg,
#0383   WPARAM wParam, LPARAM lParam) {
#0384
#0385   switch (uMsg) {
#0386
#0387     // Standard Windows messages
#0388     adgHANDLE_DLGMSG(hwnd, WM_INITDIALOG, VerShow_OnInitDialog);
#0389     adgHANDLE_DLGMSG(hwnd, WM_COMMAND,    VerShow_OnCommand);
#0390   }
#0391   return(FALSE);
#0392 }
#0393
#0394
#0395 ////////////////////////////////////////////////////////////////
#0396
```

```
#0397
#0398  int WINAPI WinMain (HINSTANCE hinstExe, HINSTANCE hinstPrev,
#0399     LPSTR lpszCmdLine, int nCmdShow) {
#0400
#0401     adgWARNIFUNICODEUNDERWIN95();
#0402     adgVERIFY(-1 != DialogBox(hinstExe,
#0403        MAKEINTRESOURCE(IDD_VERSHOW), NULL, VerShow_DlgProc));
#0404
#0405     return(0);
#0406  }
#0407
#0408
#0409  ////////////////////////// End of File ///////////////////////////
```

程式列表 9.2　VerShow.RC

```
#0001  //Microsoft Visual C++ generated resource script.
#0002  //
#0003  #include "resource.h"
#0004
#0005  #define APSTUDIO_READONLY_SYMBOLS
#0006  ////////////////////////////////////////////////////////////////
#0007  //
#0008  // Generated from the TEXTINCLUDE 2 resource.
#0009  //
#0010  #define APSTUDIO_HIDDEN_SYMBOLS
#0011  #include "windows.h"
#0012  #undef APSTUDIO_HIDDEN_SYMBOLS
#0013  #include "ver.h"
#0014  #include "vershow.h"
#0015
#0016  ////////////////////////////////////////////////////////////////
#0017  #undef APSTUDIO_READONLY_SYMBOLS
#0018
#0019
#0020  ////////////////////////////////////////////////////////////////
#0021  //
#0022  // Icon
#0023  //
#0024
#0025  IDI_VERSHOW           ICON    DISCARDABLE     "VerShow.ico"
#0026
#0027  ////////////////////////////////////////////////////////////////
#0028  //
#0029  // Version
```

```
#0030  //
#0031
#0032  VS_VERSION_INFO VERSIONINFO
#0033   FILEVERSION 2,0,0,0
#0034   PRODUCTVERSION 3,0,0,0
#0035   FILEFLAGSMASK 0x3fL
#0036  #ifdef _DEBUG
#0037   FILEFLAGS 0x3L
#0038  #else
#0039   FILEFLAGS 0x2L
#0040  #endif
#0041   FILEOS 0x10004L
#0042   FILETYPE 0x1L
#0043   FILESUBTYPE 0x0L
#0044  BEGIN
#0045     BLOCK "StringFileInfo"
#0046     BEGIN
#0047        BLOCK "040704b0"
#0048        BEGIN
#0049           VALUE "Comments", "Deutsche Version der StringFileInfo Section\0"
#0050           VALUE "CompanyName", "M & T Books, Inc.\0"
#0051             VALUE "FileDescription", "VerShow dient zur Schnellansicht von der Dateiversions-
Resource\0"
#0052           VALUE "FileVersion", "2.0\0"
#0053           VALUE "InternalName", "VerShow\0"
#0054           VALUE "LegalCopyright", "Copyright 1995 Jeffrey Richter. Alle Rechte vorbehalten\0"
#0055           VALUE "LegalTrademarks", "Vershow ist ein eingetragenes Warenzeichen von Jeffrey
Richter\0"
#0056           VALUE "OriginalFilename", "VerShow.exe\0"
#0057           VALUE "ProductName", "Windows 95: Richtlinien fuer die Programmierung\0"
#0058           VALUE "ProductVersion", "3.0\0"
#0059        END
#0060        BLOCK "040904b0"
#0061        BEGIN
#0062           VALUE "Comments", "This is the U.S. English version of the StringFileInfo section\0"
#0063           VALUE "CompanyName", "M & T Books, Inc.\0"
#0064           VALUE "FileDescription", "VerShow: File version resource information viewer\0"
#0065           VALUE "FileVersion", "2.0\0"
#0066           VALUE "InternalName", "VerShow\0"
#0067           VALUE "LegalCopyright", "Copyright \251 1995 by Jeffrey Richter\0"
#0068           VALUE "LegalTrademarks", "VerShow\256 is a registered trademark of Jeffrey Richter.\0"
#0069           VALUE "OriginalFilename", "VerShow.exe\0"
#0070           VALUE "ProductName", "Windows 95: A Developer's Guide\0"
#0071           VALUE "ProductVersion", "3.0\0"
#0072        END
#0073        BLOCK "040c04b0"
```

```
#0074        BEGIN
#0075           VALUE "Comments", "Version Fran\347aise de la Section
StringFileInfo\0"
#0076           VALUE "CompanyName", "M & T Books, Inc.\0"
#0077           VALUE "FileDescription", "VerShow sert \340 obtenir l'Information de la Ressource
de Version d'un Fichier\0"
#0078           VALUE "FileVersion", "2.0\0"
#0079           VALUE "InternalName", "VerShow\0"
#0080           VALUE "LegalCopyright", "(C) Copyright 1995 Jeffrey Richter. Tous droits
r\351serv\351s\0"
#0081           VALUE "LegalTrademarks", "VerShow est une marque d\351pos\351e de Jeffrey Richter\0"
#0082           VALUE "OriginalFilename", "VerShow.exe\0"
#0083           VALUE "ProductName", "Windows 95: Directives de Programmation\0"
#0084           VALUE "ProductVersion", "3.0\0"
#0085        END
#0086     END
#0087     BLOCK "VarFileInfo"
#0088     BEGIN
#0089        VALUE "Translation", 0x407, 1200, 0x409, 1200, 0x40c, 1200
#0090     END
#0091  END
#0092
#0093
#0094
#0095  /////////////////////////////////////////////////////////////////
#0096  //
#0097  // Dialog
#0098  //
#0099
#0100  IDD_VERSHOW DIALOG DISCARDABLE  -32768, 5, 275, 261
#0101  STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
#0102  CAPTION "Version Shower"
#0103  FONT 6, "Helv"
#0104  BEGIN
#0105     PUSHBUTTON     "Select &file...",IDC_SELECTFILE,4,4,48,14
#0106     CONTROL        "(Please select a file)",IDC_PATHNAME,"Static",
#0107                    SS_LEFTNOWORDWRAP | SS_NOPREFIX | WS_BORDER | WS_GROUP,
#0108                    56,4,212,14
#0109     GROUPBOX       "VS_FIXEDFILEINFO (Version Information)",IDC_STATIC,4,24,
#0110                    264,96
#0111     LTEXT          "Signature:\nStruct ver:\nFile ver:\nProd ver:\nFlag
mask:\nFlags:\nOS:\nType:\nSubtype:\nDate:",
#0112                    IDC_FIXEDFILEINFOFIELDS,8,36,40,80,SS_NOPREFIX
#0113     LTEXT          "",IDC_FIXEDFILEINFODATA,48,36,216,80,SS_NOPREFIX
#0114     LTEXT          "&Translations:",IDC_STATIC,4,124,44,8
#0115     COMBOBOX       IDC_TRANSLATIONS,52,124,216,80,CBS_DROPDOWNLIST |
```

```
#0116                   WS_VSCROLL | WS_GROUP | WS_TABSTOP
#0117     LTEXT          "&Strings:",IDC_STATIC,4,140,32,10
#0118     EDITTEXT       IDC_STRINGS,4,152,264,104,ES_MULTILINE | ES_AUTOHSCROLL |
#0119                    ES_READONLY | WS_VSCROLL | WS_HSCROLL
#0120 END
#0121
#0122
#0123 #ifdef APSTUDIO_INVOKED
#0124 ////////////////////////////////////////////////////////////////////
#0125 //
#0126 // TEXTINCLUDE
#0127 //
#0128
#0129 1 TEXTINCLUDE DISCARDABLE
#0130 BEGIN
#0131     "resource.h\0"
#0132 END
#0133
#0134 2 TEXTINCLUDE DISCARDABLE
#0135 BEGIN
#0136     "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0137     "#include ""windows.h""\r\n"
#0138     "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
#0139     "#include ""ver.h""\r\n"
#0140     "#include ""vershow.h""\r\n"
#0141     "\0"
#0142 END
#0143
#0144 3 TEXTINCLUDE DISCARDABLE
#0145 BEGIN
#0146     "\r\n"
#0147     "\0"
#0148 END
#0149
#0150 ////////////////////////////////////////////////////////////////////
#0151 #endif    // APSTUDIO_INVOKED
#0152
#0153
#0154 ////////////////////////////////////////////////////////////////////
#0155 //
#0156 // String Table
#0157 //
#0158
#0159 STRINGTABLE DISCARDABLE
#0160 BEGIN
#0161     0                   "7-bit ASCII"
```

```
#0162  END
#0163
#0164  STRINGTABLE DISCARDABLE
#0165  BEGIN
#0166      932                 "Windows, Japan (Shift - JIS X-0208)"
#0167  END
#0168
#0169  STRINGTABLE DISCARDABLE
#0170  BEGIN
#0171      949                 "Windows, Korea (Shift - KSC 5601)"
#0172      950                 "Windows, Taiwan (GB5)"
#0173  END
#0174
#0175  STRINGTABLE DISCARDABLE
#0176  BEGIN
#0177      1200                "Unicode"
#0178  END
#0179
#0180  STRINGTABLE DISCARDABLE
#0181  BEGIN
#0182      1250                "Windows, Latin-2 (Eastern Europe)"
#0183      1251                "Windows, Cyrillic"
#0184      1252                "Windows, Multilingual"
#0185      1253                "Windows, Greek"
#0186      1254                "Windows, Turkish"
#0187      1255                "Windows, Hebrew"
#0188      1256                "Windows, Arabic"
#0189  END
#0190
#0191
#0192  #ifndef APSTUDIO_INVOKED
#0193  ////////////////////////////////////////////////////////////////
#0194  //
#0195  // Generated from the TEXTINCLUDE 3 resource.
#0196  //
#0197
#0198
#0199  ////////////////////////////////////////////////////////////////
#0200  #endif    // not APSTUDIO_INVOKED
```

程式列表 9.3　Resource.H

```
#0001  //{{NO_DEPENDENCIES}}
#0002  // Microsoft Visual C++ generated include file.
#0003  // Used by VerShow.rc
```

```
#0004  //
#0005  #define IDD_VERSHOW                 101
#0006  #define IDI_VERSHOW                 102
#0007  #define IDC_PATHNAME                1000
#0008  #define IDC_FIXEDFILEINFOFIELDS       1001
#0009  #define IDC_FIXEDFILEINFODATA         1002
#0010  #define IDC_TRANSLATIONS             1003
#0011  #define IDC_STRINGS                 1004
#0012  #define IDC_SELECTFILE               1005
#0013  #define IDC_STATIC                  -1
#0014
#0015  // Next default values for new objects
#0016  //
#0017  #ifdef APSTUDIO_INVOKED
#0018  #ifndef APSTUDIO_READONLY_SYMBOLS
#0019  #define _APS_NO_MFC                 1
#0020  #define _APS_NEXT_RESOURCE_VALUE       103
#0021  #define _APS_NEXT_COMMAND_VALUE        40001
#0022  #define _APS_NEXT_CONTROL_VALUE        1007
#0023  #define _APS_NEXT_SYMED_VALUE         101
#0024  #endif
#0025  #endif
```

當你按下【Select File】鈕時，VerShow_OnCommand 函式會初始化一個 OPENFILENAME 結構，並呼叫 GetOpenFileNmae 函式，讓你可以選取檔案。在你選取一個檔案之後，VerShow_OnCommand 函式會以所選取的檔案的版本資訊，藉由呼叫 VerShow_InitFileVerInfo 函式，將對話盒中所有的控制元件內容予以更新：

```
void VerShow_InitFileVerInfo (HWND hwnd, LPCTSTR pszPathname)
```

這個函式首先檢查是否記憶體區塊中已經記錄了先前所選取的檔案的版本資訊：

```
// If the user was already viewing a file's resource data, clean up.
pVerInfo = (PVOID) GetWindowLong(hwnd, GWL_USERDATA);
if (pVerInfo != NULL) {
   free(pVerInfo);
   SetWindowLong(hwnd, GWL_USERDATA, (LONG) (pVerInfo = NULL));
}
```

注意，「被選取檔案之版本資訊」的記憶體位址，被存放在對話盒的 GWL_USERDATA

視窗位元組中。在釋放這個記憶體區塊之後，VerShow_InitFileVerInfo 函式會針對這個新被選取的檔案來做處理。

首先，它會將檔案的路徑名稱放置於對話盒上方的 static 控制元件之中。接著，它會配置一塊記憶體，並讀取新檔案的版本資訊到記憶體中：

```
// Calculate the size of the version control resource information.
// NOTE: The cast to (LPTSTR) is necessary because GetFileVersionInfoSize
//       is prototyped incorrectly in WINVER.h.
dwVerSize = GetFileVersionInfoSize((LPTSTR) pszPathname, NULL);
if (dwVerSize > 0) {

   // Allocate a memory block large enough to hold the version information.
   if ((pVerInfo = malloc(dwVerSize)) != NULL) {

      // Load the version information into memory.
      // NOTE: The cast to the following (LPTSTR) is necessary because
      //       GetFileVersionInfo is prototyped incorrectly in WINVER.h.
      GetFileVersionInfo((LPTSTR) pszPathname, 0, dwVerSize, pVerInfo);
      SetWindowLong(hwnd, GWL_USERDATA, (LONG) pVerInfo);
   } else {
      adgMB(__TEXT("Insufficient memory."));
   }
}
```

你可以看到，GetFileVersionInfoSize 函式先被呼叫，以計算出檔案的版本資訊需要多少個位元組才足夠存放。然後 malloc 函式會被呼叫，用以配置記憶體區塊。最後會呼叫 GetFileVersionInfo 函式，將檔案的版本資訊載入記憶體中。載入之後，此記憶體區塊的位址會被存放在對話盒的 GWL_USERDATA 視窗位元組中。

注意，使用版本控制相關函式，會有幾個典型的小問題存在。許多版本控制函式的原型被宣告爲傳入一個「指向某一字串的指標」。但事實上它們應傳入一個「指向某一定值（constant）字串的指標」。就因爲這個原因，所以我必須在我的程式碼中加入一些型別轉換，以防止編譯器產生警告訊息。希望微軟公司能在新版的 Windows 表頭檔中修復這個函式原型宣告上的錯誤。

現在，檔案的版本資訊已載入記憶體中了，你可以呼叫 VerShow_InitFileVerInfo 函式來更新對話盒中的控制元件內容。首先呼叫 VerShow_ConstructFixedStr 函式來更新 VS_FIXEDFILLEINFO 中的資訊，它會剖析 VS_FIXEDFILLEINFO 結構的各個欄位，並將其結果填入一個字串緩衝區中。這個字串緩衝區稍後會被放置在對話盒中的一個 static 控制元件中：

```
// Update the VS_FIXEDFILEINFO information.
VerShow_ConstructFixedStr(szBuf, pVerInfo);
SetDlgItemText(hwnd, IDC_FIXEDFILEINFODATA, szBuf);
```

接著，VerShow_InitFileVerInfo 函式會去呼叫 VerQueryValue 函式以確定在 Translation 欄位中有多少個位元組，並將其值所代表的意義（即語系和字元集）填入到 ”Translations” combobox 中。這個函式也會傳回 Translation 資料所在的記憶體位址。接下來會有一迴路用來取得 Translation 欄位中每一個語系的值。迴路中的每一次反覆皆會呼叫 VerLanguageName 函式以獲得語系的字串名稱，然後呼叫 LoadString 函式從 VerShow 程式自己的字串格式（侯俊傑註：資源檔中的另一種資源）中取得語系名稱。每一個被建構的字串都會被加到 combobox 中。當迴路中所有動作都完成後，第 0 項資訊會成爲預設項目：

```
// Update the Translations combobox.
ComboBox_ResetContent(GetDlgItem(hwnd, IDC_TRANSLATIONS));
if (pVerInfo != NULL) {

   VerQueryValue(pVerInfo, __TEXT("\\VarFileInfo\\Translation"),
     (PVOID *) &pdwTranslation, &uLen);
   for (x = 0; x < uLen; x += 4, pdwTranslation++) {
      // Get the string name for the language number.
      VerLanguageName(LOWORD(*pdwTranslation), szBuf, adgARRAY_SIZE(szBuf));
      lstrcat(szBuf, __TEXT(" -- "));

      // Get the string name for the code page number.
      LoadString(GetWindowInstance(hwnd), HIWORD(*pdwTranslation),
          _tcschr(szBuf, 0),
         (UINT) (adgARRAY_SIZE(szBuf) - (_tcschr(szBuf, 0) - szBuf)));
```

```
      // Add the string to the combobox.
      ComboBox_AddString(GetDlgItem(hwnd, IDC_TRANSLATIONS), szBuf);
   }
   // Select the first translation entry as the default.
   ComboBox_SetCurSel(GetDlgItem(hwnd, IDC_TRANSLATIONS), 0);
}
```

就在 VerShow_InitFileVerInfo 函式回返之前，它必須從檔案的版本資源中取得字串資料，填入 "Strings" 這個唯讀的 edit 控制元件中。這個動作可藉由呼叫 VerShow_ConstructVariableStr 函式來完成，其參數分別爲版本資訊被剖析後的字串緩衝區位址、存放版本資訊記憶體區塊的位址、以及 combobox 中用以識別「使用者所選擇的語系」的索引值。當 VerShow_ConstructVariableStr 函式回返，會將字串緩衝區中的資料填入唯讀的 edit 控制元件中：

```
// Update the string information for the default translation.
VerShow_ConstructVariableStr(szBuf, pVerInfo, 0);
SetDlgItemText(hwnd, IDC_STRINGS, szBuf);
```

附錄 A

Win95ADG.H 表頭檔

本書所有的範例程式在含入（include）其它的表頭檔之前，都含入了一個 Win95ADG.H 表頭檔。這個表頭檔的存在是爲了使程式發展更容易，其中包含了巨集、連結器指示項目（directive）和各個範例程式共用的碼。在 Win95ADG.H 表頭檔中的所有巨集名稱，皆以 “adg” （A Developer’s Guide）做爲開頭，以便你在閱讀程式碼時很快就能辨識之。建造（building）程式時，你可以試用不同的 build option，但記得要修改 Win95ADG.H 表頭檔然後再重新 building。

本附錄的其餘內容將解釋此 Win95ADG.H 表頭檔中各部份的意義。

建造選項（Build options）

第四級警告（Warning Level 4）

開發軟體的時候，我總是儘可能讓我的程式碼在編譯時不產生任何的錯誤或警告訊息。我儘可能以最嚴酷的警告等級來編譯，如此一來，編譯器會幫我做掉大部份的工作，並且幫我檢查出程式碼中較細微需要注意的地方。就 Microsoft C/C++ 編譯器而言，也就是使用第四級警告等級（warning level 4）。

不幸的是，Microsoft 作業系統開發小組並沒有使用同樣的等級，所以當我以第四級警告

來編譯程式，Windows 本身的表頭檔會產生許多警告。幸好這些警告並不表示程式有錯，通常是由於沒有遵照 C 語言的標準用法所致。

這部份的 Win95ADG.H 表頭檔明白告訴我們使用 #pragma 警告指示（warning directive）來忽略一些一般常見的警告訊息。

版本定義

本書範例程式所用到的特色幾乎都是 Windows 95 和 Windows NT 3.51[1] 有支援的。爲了利用這些新特色，在含入 Windows.h 表頭檔之前必須先定義 WINVER 巨集：

```
#define WINVER 0x0400
```

如果沒有這個定義，每當編譯時，如果我們的程式碼有參考到 WM_CAPTURECHANGED 訊息，編譯器總是會提出抱怨。此外，我們也必須告訴 Windows 說我們的程式是 Windows 4.0-aware，讓系統將其所有特色提供我們使用，也就是，把我們的程式當作一個 Windows 4.0 應用程式。

```
// Force all EXEs/DLLs to be built for Windows 4.0. Comment out the following
// #pragma line to create applications that run under Windows NT 3.1 or Win32s.
// NOTE: Windows NT 3.5 and 3.51 run Win32 programs marked as 4.0.
#pragma comment(lib, "kernel32 " "-subsystem:Windows,4.0 ")
```

這種方式可以欺騙編譯器，吐出我們所要的聯結器指示項目。也就是說系統會將「聯結器指示項目」放入 OBJ 檔中，然後聯結器便知道將這個程式標記爲 Windows 4.0 應用程式。過去我曾使用一個比較直接的方法，但 Microsoft 聯結器 2.55 版不允許我這樣做[2]。

---

1 某些情況下甚至 Windows NT 3.51 也未支援這些特色。

2 如果你想要知道爲什麼，請參閱 ***Advanced Windows***（Microsoft Press, 1995）一書的附錄 B。

STRICT 型別檢查

所有範例程式的編譯動作皆使用了嚴格（STRICT）的型別檢查特性。這項特性確保程式中的 HWND 型態資料只指定給 HWND，HDC 型態資料只指定給 HDC…等等等。有了 STRICT 定義，如果你還企圖將 HDC 型態資料指定給 HWND，編譯器會產生警告訊息。

爲了開啓嚴格（STRICT）型別檢查特性，編譯器必須在含入 Windows 表頭檔之前先定義 STRICT 巨集，這就是爲什麼在所有範例程式碼中皆先含入 Win95ADG.H 表頭檔再含入 Windows.H 的原因。

寬字元組（Unicode）

所有範例程式都可以選用以 Unicode 或 ANSI 來編譯。預設情況下程式是以 ANSI 方式來編譯字串和字元，但是可藉由定義 UNICODE 和 _UNICODE 巨集，改採用 Unicode 字串和字元。藉由 Win95ADG.H 表頭檔中定義的 UNICODE 巨集，我們很容易可以控制我們要以何種方式來編譯。如果你想多瞭解 Unicode，請參閱 ***Advanced Windows*** 一書。

CPU 可移植性（Portability）巨集

CPU 可移植性巨集使你可以利用 #pragma 敘述句，強制聯結器自動將一個 DLL 的 import library 聯結進來。舉個例，KeyCount 程式之所以和 KybdHk.dll 聯結，並不是因爲在 MAK 檔中有動作，而是因爲將聯結指示放入程式碼中，方式如下：

```
// Force a link with the import library for KybdHk.dll
#pragma comment(lib, adgLIBBUILDTYPE adgLIBCPUTYPE "\\" "KybdHk")
```

依據你的程式要建造成 debug 版或 release 版，adgLIBBUILDTYPE 巨集會被擴展爲 ”Dbg_” 或 “Rel_”。adgLIBCPPUTYPE 巨集則會依據你的 CPU 平台（platform）擴展爲 “x86”、”MIPS”、”Alpha” 或 “PPC”。於是，如果你在 Intel x86 機器上爲 KeyCount

建造一個 debug 版程式，上述的 #pragma 將會被擴展爲：

```
#pragma comment(lib, "Dbg_x86\\KybdHk")
```

前引的巨集

adgARRAY_SIZE 巨集

這個巨集幾乎在每一個範例程式中都被使用到。它會傳回陣列之中有多少個元素。它首先以 sizeof 運算子（operator）計算出這個陣列全部有多少個位元組，接著再除以「一個元素有多少個位元組」。這個巨集定義如下：

```
// This macro evaluates to the number of elements in an array.
#define adgARRAY_SIZE(Array) (sizeof(Array) / sizeof((Array)[0]))
```

adgINRANGE 巨集

這個巨集會判斷一個值是否落在某個範圍之中（兩個端點皆包括）。如果是，會傳回 TRUE。

```
// This macro evaluates to TRUE if val is between lo and hi inclusive.
#define adgINRANGE(lo, val, hi) (((lo) <= (val)) && ((val) <= (hi)))
```

adgASSERT 巨集和 adgVERIFY 巨集

發展程式時，我們常常會在程式碼的任何地方使用這個巨集來幫助我們找出潛在的問題。adgASSERT 巨集定義如下：

```
// Put up a message box if an assertion fails in a debug build
#ifdef _DEBUG
#define adgASSERT(x) if (!(x)) adgASSERTFAIL(__FILE__, __LINE__, #x)
#else
#define adgASSERT(x)
```

```
#endif
```

如果 x 是 TRUE 的話，就做測試動作。如果不是 TRUE，則顯示一個訊息指出錯誤的檔案、行號和程式碼。就 release 版程式而言，這個巨集不會做任何事。另一個巨集 adgVERIFY 幾乎和 adgASSERT 一樣，但可以在 release 版和 debug 版之下工作，定義如下：

```
// Assert in debug builds, but don't remove the code in retail builds
#ifdef _DEBUG
#define adgVERIFY(x) adgASSERT(x)
#else
#define adgVERIFY(x) (x)
#endif
```

adgHANDLE_DLGMSG 巨集

在對話盒函式中使用訊息剖析器（message crackers）時，你不應該用到 WindowsX.H 表頭檔中的 HANDLE_MSG 巨集（我曾經在 ***Advanced Windows*** 書中犯了這個錯誤），其原因是 HANDLE_MSG 巨集不會傳回 TRUE 或 FALSE 來表示對話盒函式是否已處理了該訊息。以下的 adgHANDLE_DLGMSG 巨集可以解決這個問題：

```
// The normal HANDLE_MSG macro in WINDOWSX.H does not work properly for dialog
// boxes because DlgProc's return a BOOL instead of an LRESULT (like
// WndProcs). This adgHANDLE_DLGMSG macro corrects the problem:
#define adgHANDLE_DLGMSG(hwnd, message, fn)                       \
   case (message): return (SetDlgMsgResult(hwnd, uMsg,             \
      HANDLE_##message((hwnd), (wParam), (lParam), (fn))))
```

視窗額外位元組（Window Extra Bytes）巨集

「視窗額外位元組」巨集使我們很容易就能夠存取「視窗額外位元組」中的資料。爲了使用這些巨集，程式首先必須宣告一個結構來儲存每一項資料。結構中的每項成員應爲 2 或 4 個位元組。舉個例，如果要儲存一個類別視窗代碼，和一個 DWORD 的值，應定義結構如下：

```
typedef struct {
   HWND hwndChild;
   DWORD dwLastEventTime;
} CLS_WNDEXTRABYTES;
```

當程式將 WNDCLASSEX 結構初始化並要傳遞給 RegisterClassEx 函式時，結構中的 cbWndExtra 成員應設定如下：

```
WNDCLASSEX wc;
adgINITSTRUCT(wc, TRUE);
wc.cbWndExtra = sizeof(CLS_WNDEXTRABYTES);
```

這保證會配置出適當的額外位元組大小。如果你事後想要加入、刪除或更改結構中的成員，當程式被重新編譯時，sizeof 運算子（operator）會傳回正確的結構長度。

欲將資料儲存在視窗的額外位元組中，可使用下列敘述：

```
adgSETWINDOWLONG(hwnd, CLS_WNDEXTRABYTES, dwLastEventTime, GetTickCount());
```

其中 adgSETWINDOWLOG 巨集定義如下：

```
#define adgSETWINDOWLONG(hwnd, structure, member, value) \
   SetWindowLong(hwnd, adgMEMBEROFFSET(structure, member), (LONG) (value))
```

這個巨集會呼叫 SetWindowLong 函式，並傳入藉由 adgMEMBEROFFSET 巨集所計算出來的結構欄位的偏移位址。結構中的欄位的偏移位址和視窗額外位元組中的偏移位址是一樣的。因此，這個偏移位址會在編譯時就被決定。如果結構的欄位改變，只要重新編譯這個程式就會自動產生出正確的偏移位址。

我們可以使用 adgGETWINDOWLONG 巨集來取得「視窗額外位元組」中的值。此巨集定義如下：

```
#define adgGETWINDOWLONG(hwnd, structure, member) \
```

```
GetWindowLong(hwnd, adgMEMBEROFFSET(structure, member))
```

這個巨集和 adgSETWINDOWLONG 巨集很類似，但它呼叫的是 GetWindowLong 函式，不需要多一個額外參數指定欲設定的新值。

adgMB 巨集

adgMB 巨集會顯示出一個訊息盒，其標題欄將爲執行檔的完整路徑名稱。

```
#define adgMB(s) {                                          \
     TCHAR szTMP[128];                                      \
     GetModuleFileName(NULL, szTMP, adgARRAY_SIZE(szTMP));  \
     MessageBox(GetActiveWindow(), s, szTMP, MB_OK);        \
  }
```

adgINITSTRUCT 巨集

在 Win95ADG.H 表頭檔中，adgINITSTRUCT 巨集是我最喜愛的巨集之一。我愛這個巨集！因爲在撰寫 Win32 程式時，我常常需要配置一個資料結構，並將這個結構的欄位初始化爲 0 以及設定第一個欄位爲「此一結構的大小」。我曾看過許多程式員忘了做其中一項動作，或者兩項動作都忘了，導致呼叫 Win32 函式以失敗收場。

如果你習慣使用 adgINITSTRUCT 巨集，你就可以確保結構的各個欄位總是會被清爲 0。你必須指出你是否想要設定結構中的第一個欄位爲「此一結構的大小」。如果你設定此巨集的第二個參數爲 TRUE，就會自動地這麼做，這樣你就不會忘記了。

```
// Zero out a structure. If fInitSize is TRUE then initialize the first int
to
// the size of the structure. Many structures like WNDCLASSEX and STARTUPINFO
// require that their first member be set to the size of the structure itself.
#define adgINITSTRUCT(structure, fInitSize)                     \
  (ZeroMemory(&(structure), sizeof(structure)),                 \
  fInitSize ? (*(int*) &(structure) = sizeof(structure)) : 0)
```

adgSETDLGICONS 巨集

由於大部份程式皆是使用 modal 對話盒做爲主視窗，所以我們必須手動改變對話盒的圖示（icon），使它能夠被正確地顯示在工作列（taskbar）上、"task switch" 視窗（侯俊傑註：按下 Alt-Tab 後獲得的一個小視窗，用來切換程式）上及程式標題欄中。當我們的對話盒函式取得 WM_INITDIALOG 訊息，總是會去呼叫 adgSETDIALOGICONS 巨集來正確設定圖示，巨集定義如下：

```
// The call to SetClassLong is for Windows NT
// Sending the WM_SETICON messages is for Windows 95
#define adgSETDLGICONS(hwnd, idiLarge, idiSmall)                        \
   SetClassLong(hwnd, GCL_HICON, (LONG)                                 \
      LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiLarge)));    \
   SendMessage(hwnd, WM_SETICON, TRUE,  (LPARAM)                        \
      LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiLarge)));    \
   SendMessage(hwnd, WM_SETICON, FALSE, (LPARAM)                        \
      LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiSmall)));    \
```

adgWARNIFUNICODEUNDERWIN95 巨集

Windows 95 並不像 Windows NT 一樣地完全支援 Unicode。事實上，在 Windows 95 之下呼叫 Unicode 函式的程式將全然不會被執行！不幸地是 Windows 95 並沒有任何文件加以說明。就本書程式而言，這表示當程式企圖執行起來時，其開始和結束不會有任何的顯示跡象。我們全然無知！因此我建立了 adgWARNIFUNICODEUNDERWIN95 巨集，使我們知道在 Windows 95 下如果企圖去執行一個 Unicode 程式，會產生一個訊息盒警告我們。此巨集定義如下：

```
#ifdef UNICODE

#define adgWARNIFUNICODEUNDERWIN95()                                    \
   if (GetWindowsDirectoryW(NULL, 0) <= 0)                              \
      MessageBoxA(NULL, "This operating system doesn't support Unicode.", \
         NULL, MB_OK)

#else
```

```
#define adgWARNIFUNICODEUNDERWIN95()

#endif
```

如果沒有定義 UNICODE，這個巨集不會做任何事。否則，它會宣告一個 ANSI 版的 OSVERSIONINFO 結構，然後會去呼叫 ANSI 版的 GetVersionEx 函式。這個巨集會檢查看看是否這個程式是在 Windows 95 下執行。如果是的話，就會呼叫 ANSI 版的 MessageBox 函式來通知我們說程式不能正確運作。注意，如果 Windows 96（或者無論它被叫做什麼）（侯俊傑註：可能叫作 Windows 98）對 Unicode 仍沒有更好的支援，這個巨集只好繼續它的任務。

WM_CAPTURECHANGED 訊息剖析器

定義 HANDLE_WM_CAPTURECHANGED 和 FORWARD_WM_CAPTURECHANGED 訊息剖析器的原因是：微軟公司並沒有把這些巨集加入到 WindowsX.H 表頭檔中。此一巨集定義如下：

```
/* void Cls_OnCaptureChanged(HWND hwnd, HWND hwndNewCapture) */
#define HANDLE_WM_CAPTURECHANGED(hwnd, wParam, lParam, fn) \
   ((fn)((hwnd), (HWND)(lParam)), 0L)
#define FORWARD_WM_CAPTURECHANGED(hwnd, hwndNewCapture, fn) \
   (void)(fn)((hwnd), WM_CAPTURECHANGED, (WPARAM)(HWND)(hwndNewCapture), 0L)
```

程式列表 A.1　Win95ADG.H

```
#0001  /************************************************************
#0002  Module name: Win95ADG.h
#0003  Written by: Jeffrey Richter and Jonathan Locke
#0004  Notices: Copyright (c) 1995 Jeffrey Richter
#0005  Purpose: Common header file for "Windows 95: A Developer's Guide"
#0006          Contains handy macros and definitions used throughout
#0007          the applications.
#0008  ************************************************************/
#0009
#0010  /* Disable ridiculous warnings so that the code */
#0011  /* compiles cleanly using warning level 4.     */
```

```
#0012
#0013  /* nonstandard extension 'single line comment' was used */
#0014  #pragma warning(disable: 4001)
#0015
#0016  // nonstandard extension used : nameless struct/union
#0017  #pragma warning(disable: 4201)
#0018
#0019  // nonstandard extension used : bit field types other than int
#0020  #pragma warning(disable: 4214)
#0021
#0022  // Note: Creating precompiled header
#0023  #pragma warning(disable: 4699)
#0024
#0025  // unreferenced inline function has been removed
#0026  #pragma warning(disable: 4514)
#0027
#0028  // unreferenced formal parameter
#0029  #pragma warning(disable: 4100)
#0030
#0031  // indirection to slightly different base types
#0032  #pragma warning(disable: 4057)
#0033
#0034  // named type definition in parentheses
#0035  #pragma warning(disable: 4115)
#0036
#0037  // nonstandard extension used : benign typedef redefinition
#0038  #pragma warning(disable: 4209)
#0039
#0040
#0041  ////////////////// Windows Version Build Option ////////////////////
#0042
#0043
#0044  #define WINVER 0x0400
#0045
#0046
#0047  ////////////////// Application Build Option /////////////////////
#0048
#0049
#0050  // Force all EXEs/DLLs to be built for Windows 4.0. Comment out the following
#0051  // #pragma line to create applications that run under Windows NT 3.1 or Win32s.
#0052  // NOTE: Windows NT 3.5 and 3.51 run Win32 programs marked as 4.0.
#0053  #pragma comment(lib, "kernel32 " "-subsystem:Windows,4.0 ")
#0054
#0055
#0056  ////////////////////// STRICT Build Option /////////////////////
#0057
```

```
#0058
#0059  // Force all EXEs/DLLs to use STRICT type checking.
#0060  #define STRICT
#0061
#0062
#0063  ////////////////////// CPU Portability Macros /////////////////////
#0064
#0065
#0066  // If no CPU platform was specified, default to the current platform.
#0067  #if !defined(_PPC_) && !defined(_ALPHA_) && !defined(_MIPS_) && !defined(_X86_)
#0068     #if defined(_M_IX86)
#0069        #define _X86_
#0070     #endif
#0071     #if defined(_M_MRX000)
#0072        #define _MIPS_
#0073     #endif
#0074     #if defined(_M_ALPHA)
#0075        #define _ALPHA_
#0076     #endif
#0077     #if defined(_M_PPC)
#0078        #define _PPC_
#0079     #endif
#0080  #endif
#0081
#0082  #if defined(_DEBUG)
#0083     #define adgLIBBUILDTYPE "Dbg_"
#0084  #else
#0085     #define adgLIBBUILDTYPE "Rel_"
#0086  #endif
#0087
#0088  #if defined(_X86_)
#0089     #define adgLIBCPUTYPE "x86"
#0090  #elif defined(_MIPS_)
#0091     #define adgLIBCPUTYPE "MIPS"
#0092  #elif defined(_ALPHA_)
#0093     #define adgLIBCPUTYPE "Alph"
#0094  #elif defined(_PPC_)
#0095     #define adgLIBCPUTYPE "PPC"
#0096  #else
#0097     #error Win95ADG.h : Unknown CPU platform.
#0098  #endif
#0099
#0100
#0101  ////////////////////// Unicode Build Option ///////////////////
#0102
#0103
```

```
#0104  // If we are not compiling for an x86 CPU, we always compile using Unicode.
#0105  #ifndef _X86_
#0106  #define UNICODE
#0107  #endif
#0108
#0109
#0110  // To compile using Unicode on the x86 CPU, uncomment the line below.
#0111  //#define UNICODE
#0112
#0113  // When using Unicode Win32 functions, use Unicode C-Runtime functions too.
#0114  #ifdef UNICODE
#0115  #define _UNICODE
#0116  #endif
#0117
#0118
#0119  ////////////////////// adgARRAY_SIZE Macro ///////////////////////
#0120
#0121
#0122  // This macro evaluates to the number of elements in an array.
#0123  #define adgARRAY_SIZE(Array) (sizeof(Array) / sizeof((Array)[0]))
#0124
#0125
#0126  ////////////////////////// adgINRANGE Macro ///////////////////////
#0127
#0128
#0129  // This macro evaluates to TRUE if val is between lo and hi inclusive.
#0130  #define adgINRANGE(lo, val, hi) (((lo) <= (val)) && ((val) <= (hi)))
#0131
#0132
#0133  ////////////////////// Assert/Verify Macros ///////////////////////
#0134
#0135
#0136  #define adgFAIL(szMSG) {                                          \
#0137       MessageBox(GetActiveWindow(), szMSG,                         \
#0138          __TEXT("Assertion Failed"), MB_OK | MB_ICONERROR);        \
#0139       DebugBreak();                                                \
#0140     }
#0141
#0142  // Put up an assertion failure message box
#0143  #define adgASSERTFAIL(file,line,expr) {                           \
#0144       TCHAR sz[128];                                               \
#0145       wsprintf(sz, __TEXT("File %hs, line %d : %hs"), file, line, expr);  \
#0146       adgFAIL(sz);                                                 \
#0147     }
#0148
#0149  // Put up a message box if an assertion fails in a debug build
```

```
#0150  #ifdef _DEBUG
#0151  #define adgASSERT(x) if (!(x)) adgASSERTFAIL(__FILE__, __LINE__, #x)
#0152  #else
#0153  #define adgASSERT(x)
#0154  #endif
#0155
#0156  // Assert in debug builds, but don't remove the code in retail builds
#0157  #ifdef _DEBUG
#0158  #define adgVERIFY(x) adgASSERT(x)
#0159  #else
#0160  #define adgVERIFY(x) (x)
#0161  #endif
#0162
#0163
#0164  ////////////////////// adgHANDLE_DLGMSG Macro ///////////////////
#0165
#0166
#0167  // The normal HANDLE_MSG macro in WINDOWSX.H does not work properly for dialog
#0168  // boxes because DlgProc's return a BOOL instead of an LRESULT (like
#0169  // WndProcs). This adgHANDLE_DLGMSG macro corrects the problem:
#0170  #define adgHANDLE_DLGMSG(hwnd, message, fn)                      \
#0171     case (message): return (SetDlgMsgResult(hwnd, uMsg,           \
#0172        HANDLE_##message((hwnd), (wParam), (lParam), (fn))))
#0173
#0174
#0175  ////////////////////// Window Extra Byte Macros ////////////////////
#0176
#0177
#0178  // Macros to compute the size and offset of structure members
#0179  #define adgMEMBEROFFSET(structure, member) (int) (&(((structure *)0)->member))
#0180
#0181  // Macros to compute offsets and get/set window values based on the layout of
#0182  // a structure.
#0183  #define adgSETWINDOWWORD(hwnd, structure, member, value) \
#0184     SetWindowWord(hwnd, adgMEMBEROFFSET(structure, member), (WORD) (value))
#0185  #define adgSETWINDOWLONG(hwnd, structure, member, value) \
#0186     SetWindowLong(hwnd, adgMEMBEROFFSET(structure, member), (LONG) (value))
#0187  #define adgGETWINDOWWORD(hwnd, structure, member) \
#0188     GetWindowWord(hwnd, adgMEMBEROFFSET(structure, member))
#0189  #define adgGETWINDOWLONG(hwnd, structure, member) \
#0190     GetWindowLong(hwnd, adgMEMBEROFFSET(structure, member))
#0191
#0192
#0193  ////////////////////// Quick MessageBox Macro ////////////////////
#0194
#0195
```

```
#0196  #define adgMB(s) {                                              \
#0197       TCHAR szTMP[128];                                          \
#0198       GetModuleFileName(NULL, szTMP, adgARRAY_SIZE(szTMP));      \
#0199       MessageBox(GetActiveWindow(), s, szTMP, MB_OK);            \
#0200    }
#0201
#0202
#0203  ////////////////////// Zero Variable Macro //////////////////////
#0204
#0205
#0206  // Zero out a structure. If fInitSize is TRUE then initialize the first int to
#0207  // the size of the structure. Many structures like WNDCLASSEX and STARTUPINFO
#0208  // require that their first member be set to the size of the structure itself.
#0209  #define adgINITSTRUCT(structure, fInitSize)                      \
#0210    (ZeroMemory(&(structure), sizeof(structure)),                  \
#0211    fInitSize ? (*(int*) &(structure) = sizeof(structure)) : 0)
#0212
#0213
#0214  ////////////////// Dialog Box Icon Setting Macro ////////////////////
#0215
#0216
#0217  // The call to SetClassLong is for Windows NT 3.51 or less.  The WM_SETICON
#0218  // messages are for Windows 95 and future versions of NT.
#0219  #define adgSETDLGICONS(hwnd, idiLarge, idiSmall)                    \
#0220    {                                                                \
#0221       OSVERSIONINFO VerInfo;                                        \
#0222       adgINITSTRUCT(VerInfo, TRUE);                                 \
#0223       GetVersionEx(&VerInfo);                                       \
#0224       if ((VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT) &&         \
#0225           (VerInfo.dwMajorVersion <= 3 && VerInfo.dwMinorVersion <= 51)) {  \
#0226          SetClassLong(hwnd, GCL_HICON, (LONG)                       \
#0227             LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiLarge)));  \
#0228       } else {                                                      \
#0229          SendMessage(hwnd, WM_SETICON, TRUE,  (LPARAM)              \
#0230             LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiLarge)));  \
#0231          SendMessage(hwnd, WM_SETICON, FALSE, (LPARAM)              \
#0232             LoadIcon(GetWindowInstance(hwnd), MAKEINTRESOURCE(idiSmall)));  \
#0233       }                                                             \
#0234    }
#0235
#0236
#0237  ////////////////////// UNICODE Check Macro //////////////////////
#0238
#0239
#0240  #ifdef UNICODE
#0241
```

```
#0242  #define adgWARNIFUNICODEUNDERWIN95()                                  \
#0243     if (GetWindowsDirectoryW(NULL, 0) <= 0)                            \
#0244        MessageBoxA(NULL, "This operating system doesn't support Unicode.",  \
#0245           NULL, MB_OK)
#0246
#0247  #else
#0248
#0249  #define adgWARNIFUNICODEUNDERWIN95()
#0250
#0251  #endif
#0252
#0253
#0254  ////////////// WM_CAPTURECHANGED Message Cracker Macros //////////////
#0255
#0256
#0257  // I have defined message cracker macros for the WM_CAPTURECHANGED message
#0258  // because Microsoft did not do this in the WINDOWSX.H header file.
#0259
#0260  /* void Cls_OnCaptureChanged(HWND hwnd, HWND hwndNewCapture) */
#0261  #define HANDLE_WM_CAPTURECHANGED(hwnd, wParam, lParam, fn) \
#0262     ((fn)((hwnd), (HWND)(lParam)), 0L)
#0263  #define FORWARD_WM_CAPTURECHANGED(hwnd, hwndNewCapture, fn) \
#0264     (void)(fn)((hwnd), WM_CAPTURECHANGED, (WPARAM)(HWND)(hwndNewCapture), 0L)
#0265
#0266
#0267  ////////////////////////// End of File //////////////////////////
```

附錄 B

MsgCrack 工具

當你使用 C 語言（而非 C++）來撰寫 Windows 程式時，訊息剖析器（message crackers）會使你的視窗函式比一個通常由很大的 switch 敘述所組成的視窗函式更具可讀性，也更易維護。不幸的是目前並沒有任何適當的訊息剖析器能夠讓程式員自行定義訊息，每位程式員都必須自行撰寫「自定訊息（user-defined message）」的訊息剖析器。我可以告訴你，這是非常花時間而且非常容易出錯的。本書的 MsgCrack 工具[1] 允許你很容易地就能爲你自行定義的訊息產生出對應的訊息剖析器。

使用 MsgCrack 工具，需指定兩項敘述：一個輸入檔（通常其副檔名爲 msg）和一個輸出檔（通常其副檔名爲 h）。MsgCrack 會根據輸入檔中所定義的每一項訊息，產生此訊息的訊息定義、訊息 API、以及一對訊息剖析器（一個是 HANDLE_?M_*，另一個是 FORWARD_?M_*）。

[1] 你可以在書附碟片的 MsgCrack.0B 目錄下找到一個以 Perl script 撰寫的 MsgCrack.bat。同一個目錄下，有一個 MsgCrack 輸入檔範例（MsgCrack.msg），你可以拿它來做實驗。其它的範例分佈在各個章節的目錄之中，包括：CustCntl.04、Aecade.05、Echo.06 和 KeyCount.0B。如果你想知道有關 Perl 的資訊，請參閱 MsgCrack.0B 目錄下的 ReadMe.txt 檔。

如果輸出檔案不存在，MsgCrack 會自動產生一個輸出檔。如果輸出檔已存在，MsgCrack 不會覆蓋掉它，而是會增加下列內容到檔案中：

```
////////////////////////////// Messages //////////////////////////////
//{{adgMSGCRACK_MESSAGES
   <MsgCrack generated message definitions>
//}}adgMSGCRACK_MESSAGES
//////////////////////////// Message APIs ////////////////////////////
//{{adgMSGCRACK_APIS
   <MsgCrack generated message APIs>
//}}adgMSGCRACK_APIS
//////////////////////// Message Crackers ////////////////////////
//{{adgMSGCRACK_CRACKERS
   <MsgCrack generated message cracks>
//}}adgMSGCRACK_CRACKERS
```

MsgCrack 會以新的資訊來取代每一個被標示的區段的內容，檔案的其餘部份原封不動。你在區段之間所做的任何修改，都會被拋棄掉。

一個非常簡單的 MsgCrack 輸入檔看起來像這樣：

```
MessageBase WM_USER
Message EM_SETBRUSH Example_SetBrush - Set brush and returns old brush.
wParam HBRUSH hbrNew - New brush
Returns HBRUSH - old brush
.
```

譯註：以上程式片段應修改如下，才不會產生錯誤：

```
   MessageBase WM_USER
   MessageClass Example （MsgCrack.msg 檔中有說明，必須定義 MessageClass ）
   Message EM_SETBRUSH Example_SetBrush - Set brush and returns old brush.
   wParam HBRUSH hbrNew - New brush
   Returns HBRUSH - old brush
.
```

如果以 MsgCrack 來解析這個輸入檔，你會得到一個輸出檔如下：

```
////////////////////////////// Messages /////////////////////////////////

//{{adgMSGCRACK_MESSAGES

// Purpose: Set brush and returns old brush.
// wParam:  HBRUSH hbrNew - New brush
// lParam:  N/A
// Returns: HBRUSH - old brush
#define EM_SETBRUSH         (WM_USER + 0)

//}}adgMSGCRACK_MESSAGES

////////////////////////////// Message APIs /////////////////////////////

//{{adgMSGCRACK_APIS

#define Example_Example_SetBrush(hwnd, hbrNew) \
   ((HBRUSH)SendMessage((hwnd), EM_SETBRUSH, (WPARAM)(DWORD)(hbrNew), 0))

//}}adgMSGCRACK_APIS

////////////////////////////// Message Crackers /////////////////////////////

//{{adgMSGCRACK_CRACKERS

// HBRUSH Cls_OnExample_SetBrush (HWND hwnd, HBRUSH hbrNew)
#define HANDLE_EM_SETBRUSH(hwnd, wParam, lParam, fn) \
   (LRESULT)(fn)((hwnd), (HBRUSH)(wParam))
#define FORWARD_EM_SETBRUSH(hwnd, hbrNew, fn) \
   (HBRUSH)((fn)((hwnd), EM_SETBRUSH, (WPARAM)(DWORD)(hbrNew), 0))

//}}adgMSGCRACK_CRACKERS
```

表 **B.1** 列出了 MsgCrack 輸入檔中有效的指令。

表 B.1　MsgCrack 輸入檔的有效指令

指令敘述	參數型式	說明
//	<註解>	允許你在輸入檔中加入註釋。
MessageBase	<識別字串>	允許你定義一個基底（base）訊息，其後所定義的訊息皆會依此累加，有點像 WM_USER。在你定義任何訊息之前，你必須先在輸入檔中定義 MessageBase，那會自動將目前的 MessageOffset 值重置爲 0。
MessageOffset	<偏移值>	定義 MessageBase 的起始偏移值。只要你喜歡，你可以在你的輸入檔中定義一個新的 MessageOffset。
Message	<訊息名稱> <函式名稱> - <說明>	定義一個訊息。訊息名稱像這樣：UM_MYMESSAGE。函式名稱會被用來對訊息的 API 巨集命名。「說明」用來描述這個訊息做什麼用；它會被包含在訊息的宣告裡。
wParam	<型式> <名稱> - <說明>	定義 wParam 所表示的意義。舉個例， "UNIT nBoxes - Number of boxes to lift."
lParam	<型式> <名稱> - <說明>	定義 lParam 所表示的意義。舉個例， "HBRUSH hbrBack - New background brush."
Returns	<型式> - <說明>	定義傳回值的型式和其所表示的意義。不像之前 wParam 和 lParam 敘述那樣，此傳回值敘述不允許有名稱（因爲沒有這個必要）。
	None	這個敘述會終止輸入檔中的所有訊息。

每一個指令都必須在一行之內完成，如果一行不足以表示的話，則需在這一行的尾端加上一個反斜線（\），表示以下敘述與此行連接。表 **B.1** 中的連接符號（-）也是必要的。如果輸入檔有錯誤發生，MsgCrack 會結束，並指示發生錯誤的那一行。

作者簡介

Jeffrey Richter

Jeffrey Richter 生於美國賓夕凡尼亞州的費城，1987 年畢業於 Drexel 大學，獲得資訊科學學士學位。Jeff 自 1990 年起就開始撰寫關於 Windows 程式設計方面的書籍。他也是 ***Microsoft Systems Journal***（MSJ）的專欄作家，目前負責 Win32 Q&A 專欄。

Jeff 常常在工業界的研討會上主持課程，主題包括軟體開發、WINDEV 和 COMDEX。Jeff 大部份的時間花在各公司（包括 AT&T、DEC、Intel、Microsoft、Pitney Bowes）的 Windows 95 和 Windows NT 研發人員培訓工作上。如果你有任何問題，可以 E-Mail 和他聯絡，他的 E-Mail 地址是 v-jeffrr@microsoft.com。

Jonathan Locke

Jonathan Locke 生於美國康乃狄格州西南部的一個城市，斯坦福。1988 年畢業於華盛頓大學，獲得資訊科學學士學位。Jonathan 對電腦相關領域的學門都非常感興趣，像是編譯器、作業系統和電腦圖學等等。他自 1989 年就開始從事 Windows 應用軟體的開發工作，並且寫了許多非常有用的 Windows 工具程式。他是 Sealevel Software 公司的創辦人之一，這家公司設立於華盛頓。如果你有任何問題，可以 E-Mail 和他聯絡，他的 E-Mail 地址是 JonL@SealevelSoftware.com。