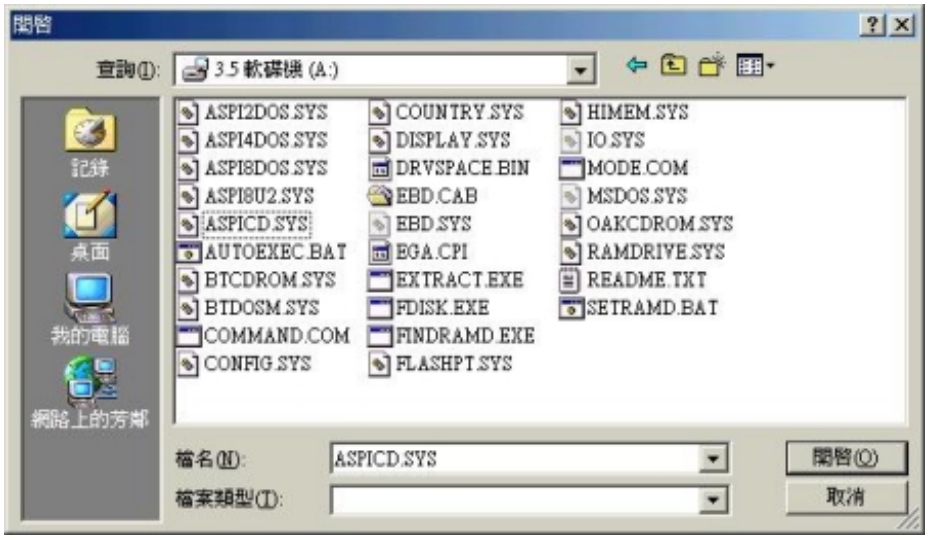
深入 C++Builder 5

使用 Windows 2000 及 Windows Me 所提供的新版檔案對話盒

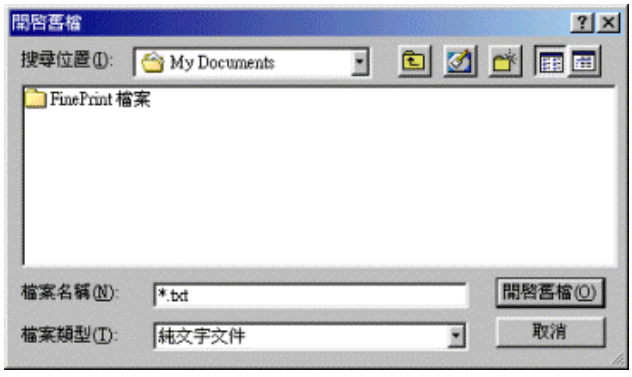
蕭永哲 martins1@ms3.hinet.net

前言

相信用過 Windows 2000 及 Windows Me 或是 Office 2000 的使用者都有發現，當應用程式開啓檔案時，使用已久的檔案對話盒已經變了一個新模樣（如圖一，Windows Me 上的新式開啓舊檔對話盒）；筆者本來以爲是新系統所提供的 comdlg32.dll 中所加上的新功能，但好奇的我寫的一個小小的測試程式來呼叫一個開啓檔案對話盒卻發現似乎不是這麼一回事，沒有想像中的簡單，因爲即使把程式搬到新的系統如 Windows Me 下，咱們寫的程式依舊是呼叫原先的那一個舊式的開啓檔案對話盒（如圖二，使用 Visual C++6.0 的 CFileDialog 類別所寫出來的測試程式，使用平台爲 Windows 2000）！怎樣才能夠讓我們即使用現有的編譯器如 Borland C++Builder 5.0 或 Microsoft Visual C++ 6.0，卻讓我們寫出來的應用程式擁有這個新式的開啓檔案對話盒的功能呢？咱們這一次就來談談這個主題吧！



圖一、Windows Me 的新式檔案對話盒



圖二、舊式的檔案對話盒

開啓檔案對話盒 SDK 基本功

先來探討當在寫一個 Windows 程式必須使用到開啓檔案對話盒時應該有的必要流程，咱們先從 SDK 的標準方法來著手。要在 Windows 程式裡頭呼叫系統的開啓檔案對話盒就得呼叫 `GetOpenFileName()`（表一）這個 Windows API，並且把一個填好資訊的 `OPENFILENAME` 結構（表二）傳入 `GetOpenFileName()`函示，如此一來即可依照

我們所填寫的OPENFILENAME 結構內容來使用開啓檔案對話盒上的功能，在了解了這些基本知識後讓筆者用SDK寫個簡單的範例來達成呼叫系統的開啓檔案對話盒功能（在範例中僅有開啓檔案部份）：

```
BOOL GetOpenFileName(
  LPOPENFILENAME lpofn //initialization data
);
```

表一、GetOpenFileName API 宣告

```
#001    typedef 結構 tagOFN {
#002    DWORD           lStructSize;
#003    HWND            hwndOwner;
#004    HINSTANCE       hInstance;
#005    LPCTSTR         lpstrFilter;
#006    LPTSTR          lpstrCustomFilter;
#007    DWORD           nMaxCustFilter;
#008    DWORD           nFilterIndex;
#009    LPTSTR          lpstrFile;
#010    DWORD           nMaxFile;
#011    LPTSTR          lpstrFileTitle;
#012    DWORD           nMaxFileTitle;
#013    LPCTSTR         lpstrInitialDir;
#014    LPCTSTR         lpstrTitle;
#015    DWORD           Flags;
#016    WORD            nFileOffset;
#017    WORD            nFileExtension;
#018    LPCTSTR         lpstrDefExt;
#019    LPARAM          lCustData;
#020    LPOFNHOOKPROC   lpfnHook;
#021    LPCTSTR         lpTemplateName;
#022    #if (_WIN32_WINNT >= 0x0500)
#023    void *          pvReserved;
#024    DWORD           dwReserved;
#025    DWORD           FlagsEx;
#026    #endif // (_WIN32_WINNT >= 0x0500)
#027    } OPENFILENAME, *LPOPENFILENAME;
```

表二、OPENFILENAME 結構宣告

程式列表一、GetOpenFileName 範例

```
#0001 BOOL OpenTheFile( HWND hWnd, HWND hWndEdit )
#0002 {
#0003    HANDLE hFile;
#0004    TCHAR       szFile[MAX_PATH]     = "\0";
#0005    strcpy( szFile, "");
#0006
#0007      // Fill in the OPENFILENAME 結構ure to support a template and hook.
#0008    OPENFILENAME OpenFileName;
#0009    OpenFileName.lStructSize      = sizeof(OPENFILENAME);
#0010    OpenFileName.hwndOwner        = hWnd;
#0011    OpenFileName.hInstance        = g_hInst;
#0012    OpenFileName.lpstrFilter      = NULL;
#0013    OpenFileName.lpstrCustomFilter = NULL;
#0014    OpenFileName.nMaxCustFilter    = 0;
#0015    OpenFileName.nFilterIndex     = 0;
#0016    OpenFileName.lpstrFile        = szFile;
#0017    OpenFileName.nMaxFile         = sizeof(szFile);
#0018    OpenFileName.lpstrFileTitle    = NULL;
#0019    OpenFileName.nMaxFileTitle     = 0;
#0020    OpenFileName.lpstrInitialDir   = NULL;
#0021    OpenFileName.lpstrTitle       = "Open a File";
#0022    OpenFileName.nFileOffset      = 0;
#0023    OpenFileName.nFileExtension    = 0;
#0024    OpenFileName.lpstrDefExt      = NULL;
#0025    OpenFileName.lCustData        = (LPARAM)&sMyData;
#0026    OpenFileName.lpfnHook            = ComDlg32DlgProc;
#0027    OpenFileName.lpTemplateName    = MAKEINTRESOURCE(IDD_COMDLG32);
#0028    OpenFileName.Flags           = OFN_SHOWHELP | OFN_EXPLORER | OFN_ENABLEHOOK |
#0029                                   OFN_ENABLETEMPLATE;
#0030
#0031    if (GetOpenFileName(&OpenFileName)) // Call the common dialog function.
#0032    {
#0033          // Open the file.
#0034          if ((hFile = CreateFile((LPCTSTR)OpenFileName.lpstrFile,
#0035               GENERIC_READ,
#0036               FILE_SHARE_READ,
#0037               NULL,
#0038               OPEN_EXISTING,
#0039               FILE_ATTRIBUTE_NORMAL,
#0040               (HANDLE)NULL)) == (HANDLE)-1)
#0041          {
```

```
#0042            MessageBox( hWnd, "File open failed.", NULL, MB_OK );
#0043            return FALSE;
#0044          }
#0045
#0046       // Do something to process file here.
#0047
#0048          // Close the file.
#0049          CloseHandle(hFile);
#0050          return TRUE;
#0051    }
#0052 }
#0053
#0054 BOOL CALLBACK ComDlg32DlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
#0055 {
#0056     switch (uMsg)
#0057     {
#0058          case WM_INITDIALOG:
#0059               // Save off the long pointer to the OPENFILENAME stcucture.
#0060               SetWindowLong(hDlg, DWL_USER, lParam);
#0061               break;
#0062          case WM_DESTROY:
#0063               {
#0064               LPOPENFILENAME lpOFN = (LPOPENFILENAME)GetWindowLong(hDlg, DWL_USER);
#0065               LPMYDATA psMyData = (LPMYDATA)lpOFN->lCustData;
#0066
#0067               GetDlgItemText(hDlg, IDE_PATH, psMyData->szTest1, sizeof(psMyData->szTest1));
#0068               GetDlgItemText(hDlg, IDE_SELECTED, psMyData->szTest2, sizeof(psMyData->szTest2));
#0069               }
#0070               break;
#0071          case WM_NOTIFY:
#0072               TestNotify(hDlg, (LPOFNOTIFY)lParam);
#0073          default:
#0074               if (uMsg == cdmsgFileOK)
#0075               {
#0076                    SetDlgItemText(hDlg, IDE_SELECTED, ((LPOPENFILENAME)lParam)->lpstrFile);
#0077                    if (MessageBox(hDlg, "Got the OK button message.\n\nShould I open it?",
#0078                              "ComDlg32 Test", MB_YESNO) == IDNO)
#0079                    {
#0080                         SetWindowLong(hDlg, DWL_MSGRESULT, 1L);
#0081                    }
#0082                    break;
#0083               }
#0084               else if (uMsg == cdmsgShareViolation)
#0085               {
#0086                    SetDlgItemText(hDlg, IDE_SELECTED, (LPSTR)lParam);
#0087                    MessageBox(hDlg, "Got a sharing violation message.", "ComDlg32 Test", MB_OK);
#0088                    break;
#0089               }
#0090               return FALSE;
#0091     }
#0092     return TRUE;
#0093 }
#0094
```

程式列表一之中的第 8 行至第 28 行都是在填寫 OPENFILENAME 結構 裡頭的資訊，而第 31 行則是把 OPENFILENAME 結構資訊傳給 GetOpenFileName()函式，而第 33 行至第 50 行則是做一個簡單的開檔動作；在這個範例中有一個值得注意的是第 26 行所指定的 Callback 函式 ComDlg32DlgProc()，ComDlg32DlgProc()將處理傳遞至開啓檔案對話盒的視窗訊息如 WM_NOTIFY。

緊接著，咱們來討論這一次的主題，如何把這個呼叫開啓檔案對話盒的動作變成呼叫新式開啓檔案對話盒的動作呢？說穿了就是在新版 Windows 中 comdlg32.dll 裡頭，有著兩個版本的開啓檔案對話盒功能，一個就是原來的對話盒（圖二），令一個就是新版本的開啓檔案對話盒（圖一），同樣的程式碼若搭配 Windows 2000 SDK 或 Windows Me SDK 重新編譯後（因爲新版本的 SDK 有著最新的標頭檔，Header File），在 Windows 2000 平台上或 Windows Me 平台上執行則自動會呈現新版本的對話盒。但我們總不能爲了使用小小的一個新功能，進而把我們的程式針對特定的一個平台搭配該平台的 SDK 重新編譯出一個特定的程式版本吧，這樣未免太小題大做了吧！還是放點心思來研究看看到底系統是怎樣判定你要用新版對話盒還是原來舊的呢？到底系統怎麼判定呢？

說穿了關鍵技術就在於 OPENFILENAME 這個結構裡頭的 lStructSize 這個成員，當使用 GetOpenFileName 這個 API 時，系統會檢查傳入的 OPENFILENAME 結構裡頭 lStructSize 的值，一般設定 lStructSize 的方法如程式列表的第九行，系統若是檢查到 lStructSize 的值是原先 OPENFILENAME 結構的大小，也就是 sizeof(OPENFILENAME) = 76，就呼叫出原先的對話盒若是 lStructSize 的值是新版 Windows SDK 裡頭的 OPENFILENAME 結構大小，則秀出新版的對話盒，由於新版本的 OPENFILENAME 加入 3 個新的成員 pvReserved，dwReserved 以及 FlagsEx（參照表二），因此新版的對話盒中 lStructSize 值應該是：sizeof(OPENFILENAME)爲 76+3*4 = 88。這個關鍵的資訊是筆者從 Windows 2000 SDK 中給猜出來的，並大膽的假設小心的求證所試驗出來的，其實每一個新版的 Windows 所提供的新功能以及新技術都可以從新平台的

SDK 以及新版的 MSDN 技術光碟中求得。如果讀者們想再 Windows 平台上努力的耕耘，那麼 SDK 以及 MSDN 技術光碟是必備的法寶了。

撇開題外話，由上頭的資訊告訴我們若要讓原先的程式能在 Windows 2000 或 Windows Me 下頭執行時能夠秀出新式的對話盒，則僅需將 `lStructSize` 的值設爲 `sizeof(OPENFILENAME)+3*4` 即可；但在此筆者發現了一個小小的問題，那就是若把 `lStructSize` 的值設定成新版本的 `OPENFILENAME` 的大小 88 以後，將編譯好的程式拿到 `Windows 98SE` 甚至更早的 `Windows` 平台上頭執行就會發現開啓檔案對話盒根本不能夠啓動了，因此！咱們的程式必須因地制宜，必須透過一些系統的 `API` 來檢查現行的 Windows 版本，根據所查詢得到的 Windows 來動態的改變 `lStructSize` 值的大小，達到各個平台上有相容性以及該有的效果，檢查 Windows 平台的版本的這部份的程式碼到後頭的實作部份再詳述，先來探討 C++Builder VCL 中的 TOpenDialog 類別。

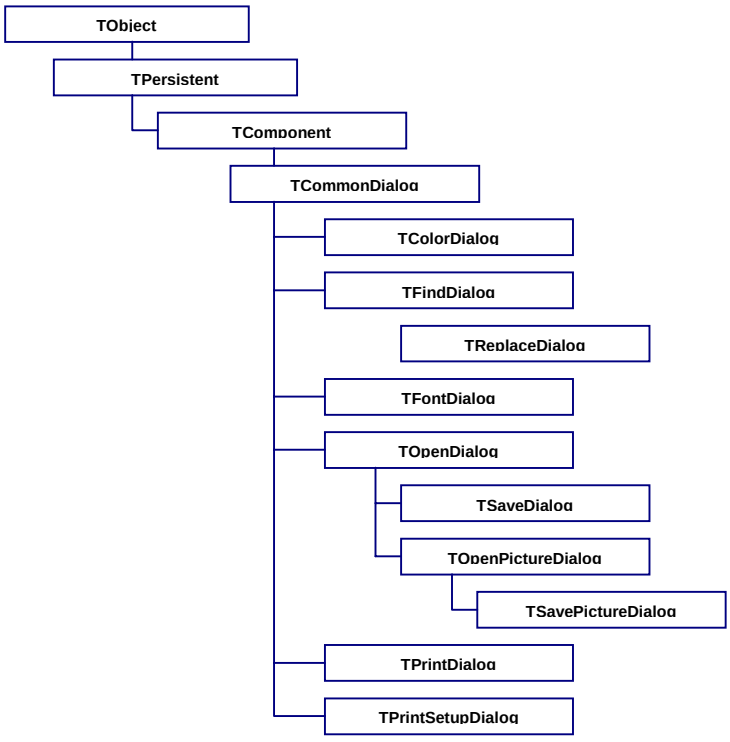
C++Builder 中 TOpenDialog 類別探訪

在 VCL（Visual Component Library）這個 Application Framework 中，TOpenDialog 包裝了 GetOpenFileName()函式，並且提供更簡單的使用方法。使用的方法如下：

在 Form 上頭建立一個 TButton 按鈕 Button1 與 TOpenDialog 對話盒 OpenDialog1，在 Button1 的 OnClick 事件裡：

```
#0001.  void __fastcall TForm1::Button1Click(TObject *Sender)
#0002.  {
#0003.    if (OpenDialog1->Execute())
#0004.    {
#0005.      //Ok button be pressed;
#0006.      //do something here
#0007.      AnsiString szFileName = OpenDialog1->FileName;
#0008.    }
#0009.  }
```

編譯好程式後執行，按下 Button 後就可以看到開啓檔案對話盒被開啓了，這就是 OOP 的魔力，省去了一大堆繁雜的程式碼，僅需呼叫 TOpenDialog 類別的 `Execute()`函式即可不必再煩瑣的先建立 `OPENFILENAME` 這個結構後再費心的將結構內容傳送給 GetOpenFileName()函示，但也因 OOP 的包裝效果進而隱藏了一大堆應該被我們控制或應該被我們了解的祕密。稍待讓我們把 VCL 神祕的面紗給解開來，看看在 VCL 中實作 TOpenDialog 的方式。在此之前咱們先得了解 TOpenDialog 在 VCL 這個 AF（Application Framework）之中的繼承架構，以其其扮演的角色：



圖三、TCommonDialog 繼承架構圖

由圖三可以看出 TOpenDialog 是繼承至 TCommonDialog，所有的系統對話盒如『列印對話盒』(TPrintDialog)、『字形對話盒』(TFontDialog)也都是繼承自 TCommonDialog，另外 TSaveDialog 這個『儲存檔案對話盒』則是從 TOpenDialog 繼承下來的，因爲長相一般僅在功能上有小小的不同罷了，最後還可以看到 TOpenPictureDialog 以及 TSavePictureDialog 這些個具有圖檔預覽功能的開檔存檔對話盒也都是繼承至 TOpenDialog。

在了解了 TOpenDialog 於 VCL 架構中扮演的角色後，有一件事情是必須先讓讀者們了解，只要是讀者手邊的 C++Builder 是 Professional 專業版以上等級的 C++Builder，讀者手邊應該就會有 VCL 的原始碼（若安裝 C++Builder 時沒有安裝到硬碟上，一樣可以在光碟片上找到），但很可惜的是這個 VCL 原始碼是由 Object Pascal 所撰寫的，筆者在探討 TOpenDialog 的實作部份會引用到 TOpenDialog 的 Object Pascal 版原始碼，由於原始碼過長，筆者僅在文章中段落式列出 VCL 原始碼，讀者們請自行參照手邊的 VCL 原始碼，但對使用 C++Builder Standard 版本的讀者們，筆者只能夠說聲抱歉了。OK！咱們開始來深入了解這一次的主題 TOpenDialog 的 VCL 原始碼裡，先從最常使用的 `Execute()`函式 VCL 原始碼中開始追蹤：

程式列表二、dialog.pas（部份列表）

```
#0831     function TOpenDialog.Execute: Boolean;
#0832     begin
#0833        Result := DoExecute(@GetOpenFileName);
#0834     end;
```

注意，以下的函式宣告 declearation 皆使用 C++版的 VCL 宣告（可以從 dialog.hpp 中查到），因此會與 VCL 原始碼程式列表中的長相有點不同（因爲 VCL 原始碼是 pascal 嘛！），讀者們請自行參照，筆者不再多家著墨，因爲畢竟咱們討論的主題是 C++Builder 而不是 Delphi 嘛！

OK！咱們先看第 831 至 834 行的 `virtual bool __fastcall Execute(void)`函式的實作內容其實就是去呼叫另一個函式 `BOOL __fastcall DoExecute(void * Func)`函式，但是傳給 `DoExecute` 函式的卻是一個函式指標，這個函式指標不是別人，正是真正呼叫出開啓檔案對話盒的 Windows API `GetOpenFileName()`，且讓我們繼續追下去 `DoExecute()`：

程式列表三、dialog.pas（部份列表）

```
#0622     function TOpenDialog.DoExecute(Func: Pointer): Bool;
#0623     const
//----略過
#0633     var
#0634       Option: TOpenOption;
#0635       OpenFilename: TOpenFilename;
#0636
//----略過
#0657     begin
#0658       FFiles.Clear;
#0659       FillChar(OpenFileName, SizeOf(OpenFileName), 0);
#0660       with OpenFilename do
#0661       begin
#0662         lStructSize := SizeOf(TOpenFilename);
#0663         hInstance := SysInit.HInstance;
#0664         TempFilter := AllocFilterStr(FFilter);
#0665         lpstrFilter := PChar(TempFilter);
#0666         nFilterIndex := FFilterIndex;
#0667         FCurrentFilterIndex := FFilterIndex;
#0668         if ofAllowMultiSelect in FOptions then
#0669           nMaxFile := MultiSelectBufferSize else
#0670           nMaxFile := MAX_PATH;
#0671         SetLength(TempFilename, nMaxFile + 2);
#0672         lpstrFile := PChar(TempFilename);
#0673         FillChar(lpstrFile^, nMaxFile + 2, 0);
#0674         StrLCopy(lpstrFile, PChar(FFileName), nMaxFile);
#0675         if (FInitialDir = '') and ForceCurrentDirectory then
#0676           lpstrInitialDir := '.'
```

```
#0677       else
#0678         lpstrInitialDir := PChar(FInitialDir);
#0679       lpstrTitle := PChar(FTitle);
#0680       Flags := OFN_ENABLEHOOK;
#0681       for Option := Low(Option) to High(Option) do
#0682         if Option in FOptions then
#0683           Flags := Flags or OpenOptions[Option];
#0684       if NewStyleControls then
#0685         Flags := Flags xor OFN_EXPLORER
#0686       else
#0687         Flags := Flags and not OFN_EXPLORER;
#0688       TempExt := FDefaultExt;
#0689       if (TempExt = '') and (Flags and OFN_EXPLORER = 0) then
#0690       begin
#0691         TempExt := ExtractFileExt(FFilename);
#0692         Delete(TempExt, 1, 1);
#0693       end;
#0694       if TempExt <> '' then lpstrDefExt := PChar(TempExt);
#0695       if (ofOldStyleDialog in Options) or not NewStyleControls then
#0696         lpfnHook := DialogHook
#0697       else
#0698       begin
#0699         lpfnHook := ExplorerHook;
#0700       end;
#0701       if Template <> nil then
#0702       begin
#0703         Flags := Flags or OFN_ENABLETEMPLATE;
#0704         lpTemplateName := Template;
#0705       end;
#0706       hWndOwner := Application.Handle;
#0707       Result := TaskModalDialog(Func, OpenFileName);
```

可以在這一段 DoExecute()函式中看到從 661 行開始還是在做原先 SDK 該做的事情，就是把 OPENFILENAME 結構內的資訊給填寫完畢，最後在把 GetOpenFileName()這個 API 的函式指標與 OPENFILENAME 結構再丟給 TaskModalDialog 這個函式去處裡，有一點必須注意的是 TaskModalDialog()函式並不是 TOpenDialog 的成員函式（member function）而是其父類別 TCommonDialog 的虛擬成員函式（virtual member function），其函式宣告如下：

```
virtual BOOL __fastcall TaskModalDialog(void * DialogFunc, void *DialogData);
```

其實作方法如下：

程式列表四、dialog.pas（部份列表）

```
#0464   function TCommonDialog.TaskModalDialog(DialogFunc: Pointer; var DialogData): Bool;
#0465   type
#0466     TDialogFunc = function(var DialogData): Bool stdcall;
#0467   var
#0468     ActiveWindow: HWnd;
#0469     WindowList: Pointer;
#0470     FPUControlWord: Word;
#0471     begin
#0472       ActiveWindow := GetActiveWindow;
#0473       WindowList := DisableTaskWindows(0);
#0474       try
#0475         Application.HookMainWindow(MessageHook);
#0476         asm
#0477         // Avoid FPU control word change in NETRAP.dll, NETAPI32.dll, etc
#0478           FNSTCW  FPUControlWord
#0479       end;
#0480       try
#0481         CreationControl := Self;
#0482         Result := TDialogFunc(DialogFunc)(DialogData);
#0483       finally
#0484       asm
#0485         FNCLEX
#0486         FLDCW FPUControlWord
#0487       end;
#0488       Application.UnhookMainWindow(MessageHook);
#0489     end;
#0490     finally
#0491     EnableTaskWindows(WindowList);
#0492     SetActiveWindow(ActiveWindow);
#0493     end;
#0494   end;
```

在 466 與 482 行可以看到最後還是把原先使用的 GetOpenFileName()函式指標傳到 TaskModalDialog()之中並且處理掉 OPENFILENAME 這個結構，你或許會認爲幹嘛繞這一大圈，直接在 DoExecute()函式裡頭之皆使用

GetOpenFileName() API 不就得了嗎？VCL 在這 TaskModalDialog()函式之中把視窗訊息給導入到 VCL 的訊息處理清單中，經由 VCL 來作訊息的分派。（註一）

清楚了 TOpenDialog 在 VCL 中的實作以及了解了新式對話盒的呼叫方式後，咱們最後要了解的就是該怎麼去判別 Windows 的版本。其實偵測 Windows 版本的方法很簡單，僅需使用 GetVersionEx 這個 Windows API 即可查詢的到，在此不做詳述，其偵測的程式碼可以由最後的主程式部份窺知一二。

自動判定 Windows 版本的 TOpenDialogEx 類別

看到 TOpenDialogEx 不難看出這是一個 TOpenDialog 的衍生類別，但他提供了一個自動檢查 Windows 版本的功能，當使用最新版本的 Windows（如 Windows Me 以及 Windows 2000 以後的 Windows）時，會自動的呼叫出新版本的開啓檔案對話盒，但若爲 Windows 98SE、Windows NT 4.0 甚至更老舊的 Windows（當然不包括了 Windows 3.1）時，則使用原先的檔案開啓對話盒。

那在程式碼上該怎麼下手去實作呢？看看上頭把 TOpenDialog 分解的片段原始碼，你覺得怎麼做比較好呢？最快的方法就是寫一個新的類別 TOpenDialogEx 繼承於原先的 TOpenDialog，並把呼叫 GetOpenFilename API 部份的程式碼修改一下，讓 OPENFILENAME 結構裡頭的 lStructSize 依據不同的 Windows 版本而有不同的值，而在 TOpenDialg 中哪一部份才是真正去呼叫 GetOpenFileName API 呢？就是 TaskModalDialog()函式，從 TaskModalDialog()函式的原始宣告可以看出，TaskModalDialog()函式是個虛擬函式，我們可以在衍生類別下作改寫的動作，因此在筆者腦袋裡很快的勾勒出 TOpenDialogEx 列別的基礎模型了：

程式列表五、TOpenDialogEx.h

```
#0001       #include <dialogs.hpp>
#0002
#0003       //自己定義的 OPENFILENAMEEX 結構
#0004       //避免與原先系統的 OPENFILENAME 互相衝突
#0005       typedef 結構 tagOFNEX {
#0006         DWORD        lStructSize;
#0007         HWND         hwndOwner;
#0008         HINSTANCE     hInstance;
#0009         LPCTSTR       lpstrFilter;
#0010         LPTSTR        lpstrCustomFilter;
#0011         DWORD        nMaxCustFilter;
#0012         DWORD        nFilterIndex;
#0013         LPTSTR        lpstrFile;
#0014         DWORD        nMaxFile;
#0015         LPTSTR        lpstrFileTitle;
#0016         DWORD        nMaxFileTitle;
#0017         LPCTSTR       lpstrInitialDir;
#0018         LPCTSTR       lpstrTitle;
#0019         DWORD        Flags;
#0020         WORD         nFileOffset;
#0021         WORD         nFileExtension;
#0022         LPCTSTR       lpstrDefExt;
#0023         LPARAM        lCustData;
#0024         LPOFNHOOKPROC lpfnHook;
#0025         LPCTSTR       lpTemplateName;
#0026         void *        pvReserved;  //新版 Windows 才有的 Member
#0027         DWORD        dwReserved;  //新版 Windows 才有的 Member
#0028         DWORD        FlagsEx;     //新版 Windows 才有的 Member
#0029       } OPENFILENAMEEX, *LPOPENFILENAMEEX;
#0030
#0031       //檢查 Windows 版本的函式,若爲 Windows Me 或 Windows 2000 則傳回 true,其餘則爲 false
#0032       bool __fastcall CheckIsWindows2K();
#0033       class TOpenDialogEx : public TOpenDialog
#0034       {
#0035         private:
#0036         protected:
#0037           //新的 TaskModalDialog,用來改變 OPENFILENAME 裡 lStructSize 的大小
#0038           virtual BOOL __fastcall TaskModalDialog(void * DialogFunc, void *DialogData);
#0039         public:
#0040           __fastcall TOpenDialogEx(TComponent* AOwner);
#0041           __fastcall ~TOpenDialogEx(){};
```

```
#0042       __published:
#0043     };
```

筆者在程式的 Header File 中另外定義了一個 OPENFILENAMEEX 結構，OPENFILENAMEEX 結構是用來充當新版本的 OPENFILENAME 結構，並且可以避免在程式中搞不清楚正確的 OPENFILENAME 結構內容；此外，透過 C++ 的多形與虛擬機制，咱們僅需要在重寫一份 TOpenDialogEx::TaskModalDialog()函示來改寫的原先在 TCommonDialog::TaskModalDialog()函示的動作即可；最後，CheckIsWindows2k()函式則是用來檢查 Windows 平台的版本，若是 Windows Me 或 Windows 2000 則回傳 true，其餘平台則回傳 false，這兩個函式的改寫及實作如下：

程式列表六、TOpenDialogEx.cpp

```
#0001  //---------------------------------------------------------------------------
#0002  __fastcall TOpenDialogEx::TOpenDialogEx(TComponent* AOwner)
#0003  : TOpenDialog(Owner)
#0004  {
#0005  }
#0006  //---------------------------------------------------------------------------
#0007  BOOL __fastcall TOpenDialogEx::TaskModalDialog(void * DialogFunc, void *DialogData)
#0008  {
#0009    BOOL bResult;
#0010    BOOL b2k = CheckIsWindows2K();//檢查 Windows 版本是否爲 2000 或 Me
#0011
#0012    if (b2k)//如果是 Windows 2000 或 Windows Me
#0013    {
#0014      OPENFILENAMEEX m_ofnEx;
#0015      OPENFILENAME *pOri = (OPENFILENAME*)DialogData;
#0016      memset(&m_ofnEx,0,sizeof(m_ofnEx));
#0017      memcpy(&m_ofnEx,pOri,sizeof(OPENFILENAME));
#0018      m_ofnEx.lStructSize = sizeof(OPENFILENAMEEX);
#0019      //把新的 OPENFILENAMEEX 轉交給 TCommonDialog 的 TaskModalDialog 處理
#0020      bResult = TCommonDialog::TaskModalDialog(DialogFunc,(PVOID)(&m_ofnEx));
#0021      //還原原始的 OPENFILENAME
#0022      memcpy(pOri,&m_ofnEx,sizeof(OPENFILENAME));
#0023      pOri->lStructSize = sizeof(OPENFILENAME);
#0024    }
#0025    else//若不是新版的 Windows 直接將訊息交給 TCommonDialog 的 TaskModalDialog 處理
#0026      bResult = TCommonDialog::TaskModalDialog(DialogFunc,DialogData);
#0027    return bResult;
#0028  }
#0029  //---------------------------------------------------------------------------
#0030  bool __fastcall CheckIsWindows2K()
#0031  {
#0032    //檢查作業系統版本
#0033    //若爲 Windows Me 或 Windows 2000 回傳 true
#0034    //其餘平台則回傳 false
#0035    bool bResult = false;
#0036    // Get OS Version
#0037    OSVERSIONINFOEX osvi;
#0038    BOOL bOsVersionInfoEx;
#0039    ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
#0040    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
#0041    bOsVersionInfoEx = GetVersionEx((OSVERSIONINFO *) &osvi);
#0042    if(!bOsVersionInfoEx)
#0043    {
#0044      osvi.dwOSVersionInfoSize = sizeof (OSVERSIONINFO);
#0045      GetVersionEx ( (OSVERSIONINFO *) &osvi);
#0046    }
#0047
#0048    switch (osvi.dwPlatformId)
#0049    {
#0050      case VER_PLATFORM_WIN32_WINDOWS:
#0051      {
#0052        //Windows 9x, Windows Me 平台
#0053        if ((osvi.dwMajorVersion > 4) ||
#0054            ((osvi.dwMajorVersion == 4) && (osvi.dwMinorVersion > 0)))
#0055        {
#0056          if (osvi.dwMinorVersion == 90)
#0057            bResult = true;//Windows Me
#0058          /*
#0059          else
#0060          if (osvi.dwMinorVersion == 10)
#0061            bResult = false;//Windows 98SE
#0062          else
#0063            bResult = false;//Windows 98
#0064         */
#0065        }
#0066        /*
#0067        else
```

```
#0068         bResult = false;//Windows 95
#0069       */
#0070      }
#0071      case VER_PLATFORM_WIN32_NT:
#0072      {
#0073       //Windows NT,Windows 2000 平台
#0074       if ( osvi.dwMajorVersion == 5 )
#0075         bResult = true;//Windows 2000
#0076       /*
#0077       else
#0078       if ( osvi.dwMajorVersion <= 4 )
#0079          bResult = false;//Windows NT
#0080       */
#0081       break;
#0082      }
#0083      default:
#0084       break;
#0085    }
#0086    return bResult;
#0087  }
#0088  //---------------------------------------------------------------------------
```

在 TOpenDialogEx::TaskModalDialog()函式的實作中(第 7 至 28 行)若只貿然改變原先 OPENFILENAME 結構上 lStructSize 的值，程式在未知記憶體上可能會產生未知的錯誤，因此我們建立了一個新的 OPENFILENAMEEX 的實體並且把原來的 OPENFILENAME 結構給拷貝過來，並把 lStructSize 設成新的 OPENFILENAMEEX 大小，這樣一來可以避免程式因爲存取未知記憶體時產生的錯誤，當然了把 OPENFILENAMEEX 結構交給 TOpenDialogEx::TaskModalDialog()函式處理完畢後，咱們還是得把 OPENFILENAMEEX 結構還原成 OPENFILENAME 並把 lStructSize 設回原始 OPENFILENAME 大小。

class 寫好了總得寫個程式測測看吧！咱們修改一開始用測試 TOpenDialog 類別使用法的程式，一樣是在 Button1 的 OnClick 事件中：

程式列表七、unit1.cpp

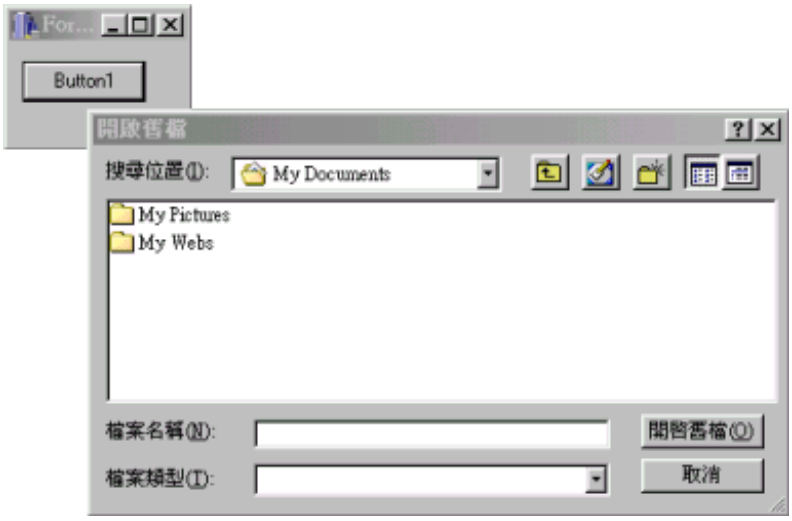
```
#0001. #include <vcl.h>
#0002. #pragma hdrstop
#0003.
#0004. #include "Unit1.h"
#0005. #include "TOpenDialogEx.h"
#0006. //---------------------------------------------------------------------------
#0007. #pragma package(smart_init)
#0008. #pragma resource "*.dfm"
#0009. TForm1 *Form1;
#0010. //---------------------------------------------------------------------------
#0011. __fastcall TForm1::TForm1(TComponent* Owner)
#0012. : TForm(Owner)
#0013. {
#0014. }
#0015. //---------------------------------------------------------------------------
#0016. void __fastcall TForm1::Button1Click(TObject *Sender)
#0017. {
#0018.
#0019.   TOpenDialogEx *pOD = new TOpenDialogEx(this);
#0020.   pOD->Execute();
#0021.   delete pOD;
#0022. }
#0023.
```

如此一來編譯後的程式在不同的平台上執行就有不同的執行效果了。(執行結果如圖四、圖五)

結語

爲了挖掘出 VCL 架構裡的奧祕，文章內列出了不少使用 Object Pascal 所撰寫的 VCL 原始碼，希望沒有把讀者們搞得暈頭轉向。在筆者使用 C++Builder 之前，對於 Object Pascal 是一竅不通，因此對於 Object Pascal 的學習一直抱著能不浪費時間學就不學的態度。但不久後，因爲工作的需要得逼自己開始鑽研 VCL 的內部構造，這才發現 VCL 的內部是如此的精妙，對於慣用 C/C++的我來說何嘗不是一個學習 Object Pascal 的機會呢？因此，在此呼籲各位讀者，若你真正想要把 C++Builder 與 VCL 架構給學好，該是自己開始研讀 VCL 的原始碼並學習 Object Pascal 的好時候了。

註一、VCL 的訊息分派可以參照筆者於 1998 年十一、十二月兩期 RUN!PC 上刊載的『親手打造 C++Builder 的 TRACE Window』一文，裡頭有詳細探討到 VCL 訊息分派，該文可於 http://insidebcb.copystar.com.tw 下載 PDF 版本。



圖四、範例程式在 Windows 98SE 中執行

圖五、範例程式在 Windows 98SE 中執行