iPush® Embedded 教育訓練教材－ Remote Administration Framework

著 作 人：艾揚科技股份有限公司

(ICE Technology Corporation)

文件編號：TEmbedded-10-002-tw

版　　次：V1.0

出版時間：2004-06-28

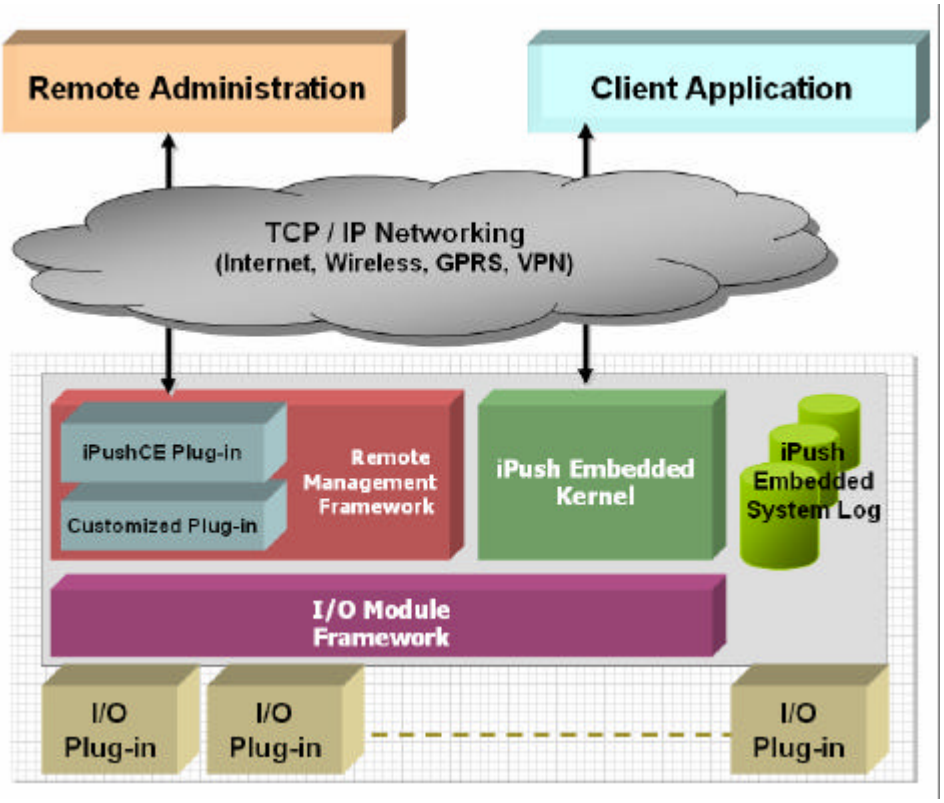
Remote Administrator Framework Programming

Remote Administration Framework 架構介紹

Remote Administration Framework 的設計目標，在於提供 PC-based 嵌入式控制器，一個方便維護、方便增加功能的遠端管理架構。讓使用者可以利用新增 DLL 的方式，來新增及維護遠端管理的功能。

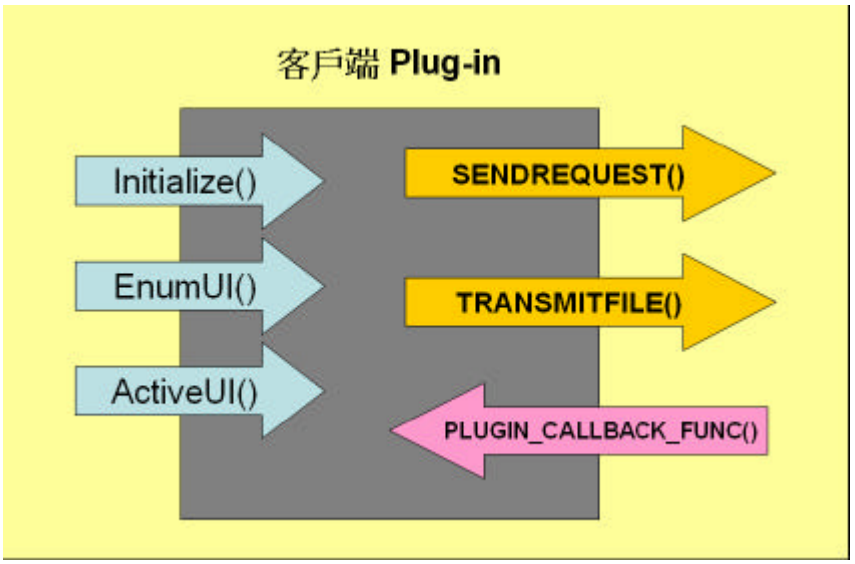
遠端呼叫架構介紹

Remote Administration Framework 是透過在客戶端及伺服端，分別新增動態連結資料庫檔案 (DLL) 的方式，來增加遠端管理的功能。



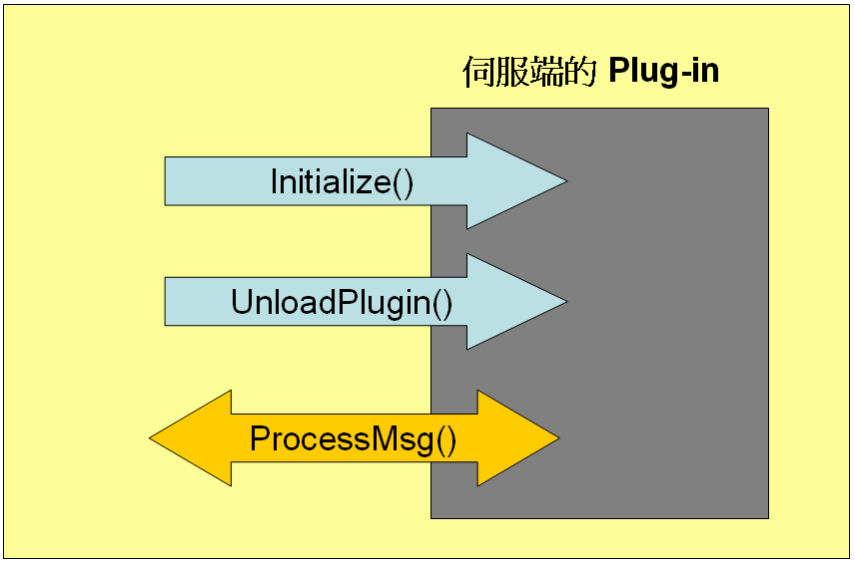
圖一、iPush Embedded 系統架構，伺服端可用增加 DLL 的方式來加強功能

若要更新或是新增客戶端的功能，可以利用撰寫新的 DLL 來完成。由於 TCP/IP 的連線部分已經交由 Remote Administration Framework 處理，所以使用者可以專心的把時間花在實際要處理的問題上，而不需要擔心連線的細節部分。



圖二、Remote Administration Framework **於客戶端需要實作的** API **介面**

在客戶端的圖例中，藍色部分是提供 Remote Administration **呼叫的介面，並透過** Send Request **以及** Transmit file **對遠端傳送命令及檔案，然後經由** Plug-in **的回呼函數** (Callback funcation) **取得結果**。。



圖三、Remote Administration Framework **於伺服端需要實作的** API **介面**

在伺服端的圖例中，藍色部分是提供 Remote Administration **呼叫的介面，並透過** ProcessMsg **接**

收遠端命令、檔案並回傳執行結果。我們將會在稍後的章節中，說明如何利用這個架構，從遠端執行伺服端的 Windows 系統函式，並回傳結果。

工具準備

雖然客戶端及 WinCon-8000 監控端都是在 Windows 平台上執行。但由於 PC 端及 WinCon-8000 端分別使用 x86 及 StrongARM 的硬體架構，所以我們必須使用不同的編譯工具來進行程式設計的工作。

在客戶端：

- 使用 Microsoft 公司的 Visual C++ 6 或 Visual C++.NET 來作為程式開發編譯的環境

而伺服端：

- 使用 Microsoft eMbedded Visual C++ 4.0 版
- Service Pack 2 和 QFE 更正檔，或是 Service Pack 3
- 安裝 StrongARM 的 Platform SDK 來進行程式的開發工作，安裝 eMbedded C++ 4.0 版時，預設就會安裝 WinCon-8000 所需的 Platform SDK

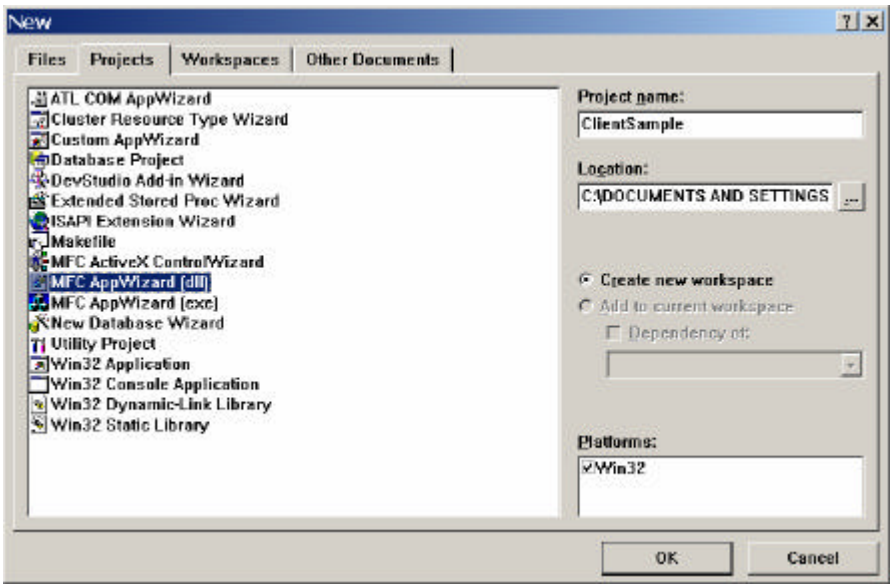
上面提到的軟體，都可以在 http://msdn.microsoft.com/downloads/ 上找到，檔名分別是 eVC4.exe、eVC4SP3.exe，可以利用輸入 eMbedded Visual C++ 來搜尋相關的結果。

建立新的遠端管理功能範例

SECTION 01：建立客戶端 Visual C++ 對話方塊專案

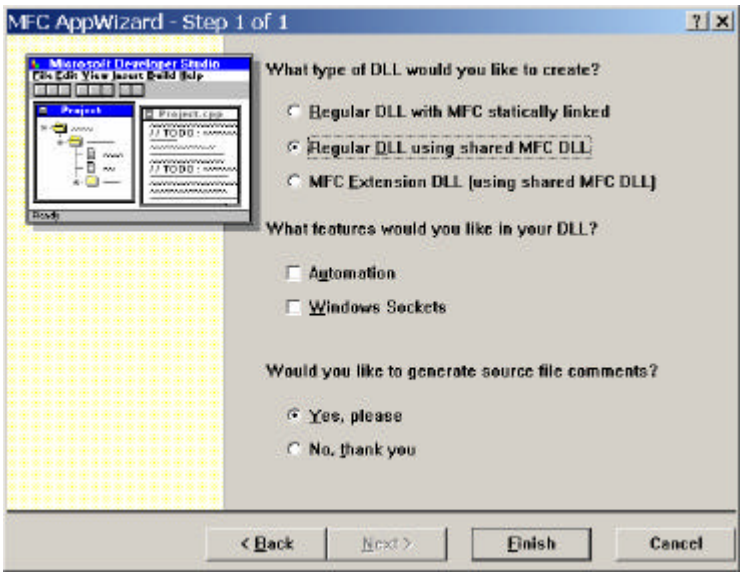
步驟一：要建立客戶端 DLL **的專案，請先開啟** Visual C++ 6.0。

步驟二：選擇「File**」下的「**New**」建立新的專案。選擇「**MFC AppWizard (dll)**」然後將檔名命名為「**Client Sample**」。**

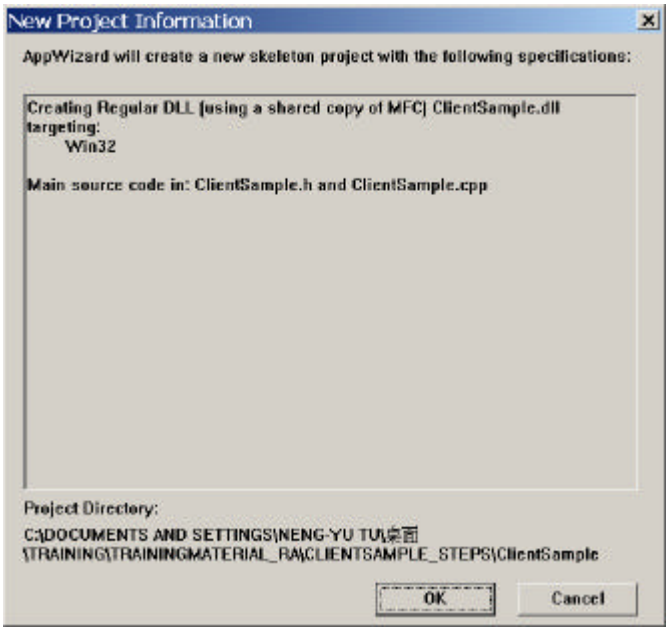


圖四、選擇建立 MFC AppWizard (dll)

步驟三：選擇「MFC AppWizard (dll)**」後，選擇「**Regular DLL using shared MFC DLL**」後，保持其他的預設選項，按下「**Finish**」鈕後，按下「**OK**」鈕完成專案的設定工作。**



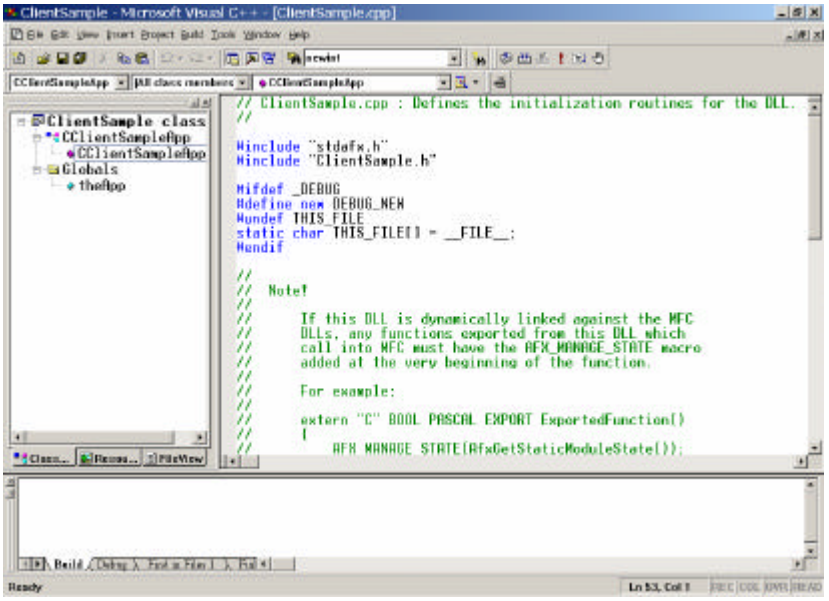
圖五、選擇建立 simple DLL 的專案



圖六、按下「OK」鈕完成設定

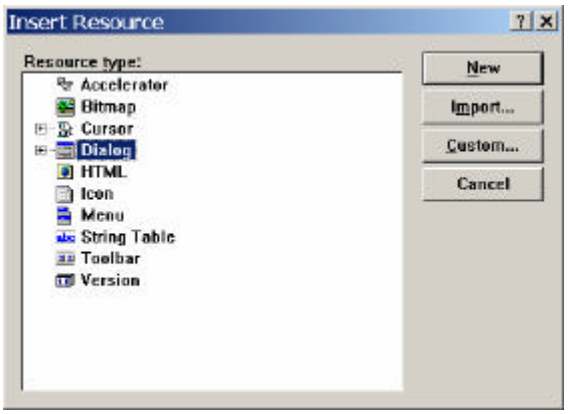
SECTION 02：建立客戶端的對話方塊介面

在建立完成專案的內容後，接下來我們先進入介面設計的工作。由於在這個範例中，我們必須使用一個對話方塊來當成資料輸入，以及顯示遠端回應訊息的工具。我們會在 ResourceView 標籤中，加入一個對話方塊資源，以及兩個 Edit Box 及對應的 Label 和 Button 資源。



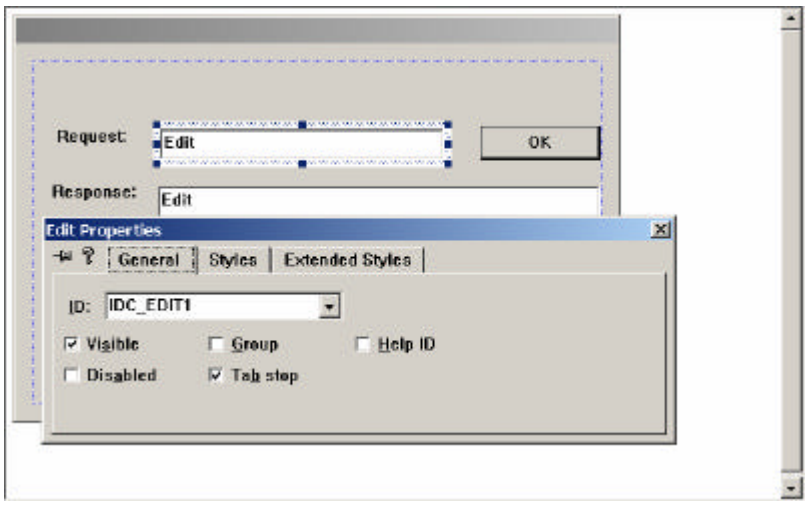
圖七、精靈會幫我們加入一些基本的程式碼及註解

步驟一：在這個程式資源中，加入一個對話方塊，以提供我們輸入命令以及顯示回應訊息使用，請選擇功能表中「Insert」下的「Resource」，插入一個對話方塊 (Dialog) 資源。



圖八、選擇對話方塊 (Dialog) 資源後，按下 「New」鈕

步驟二：在新增的對話方塊中，加入兩個 Edit Box、兩個 Label 以及一個 Button 的標籤 (Tab)。加入後，在元件上方按下滑鼠右鍵，設定名稱屬性。

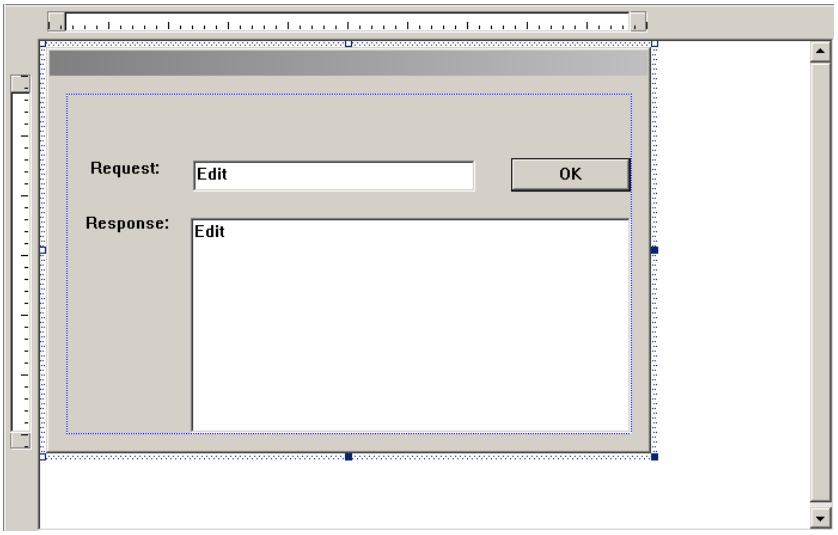


圖九、設定對話方塊標籤的名稱屬性

對話方塊及上面的元件屬性及功能如下表：

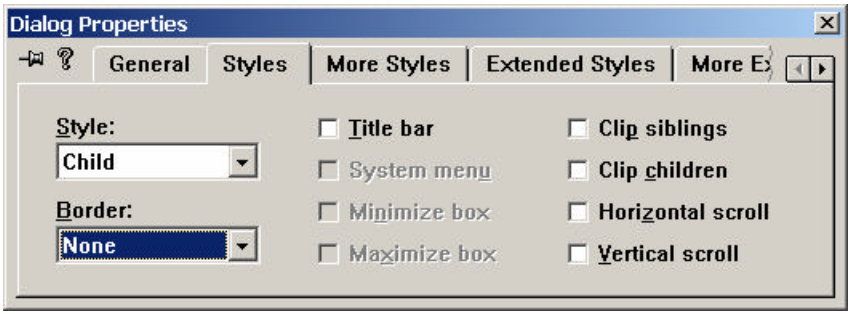
元件	命名	功用
Edit Box	IDC_EDIT_REQUEST	提供輸入遠端指令的視窗
Edit Box	IDC_EDIT_RESPONSE	提供遠端命令回應顯示的視窗
Label	IDC_STATIC	提示 Request: 文字方塊位置
Label	IDC_STATIC	提示 Response: 文字方塊位置
Button	IDC_BTN_SEND	傳送命令給遠端伺服器
對話方塊	IDD_DIALOG1	對話方塊

完成後如下圖：

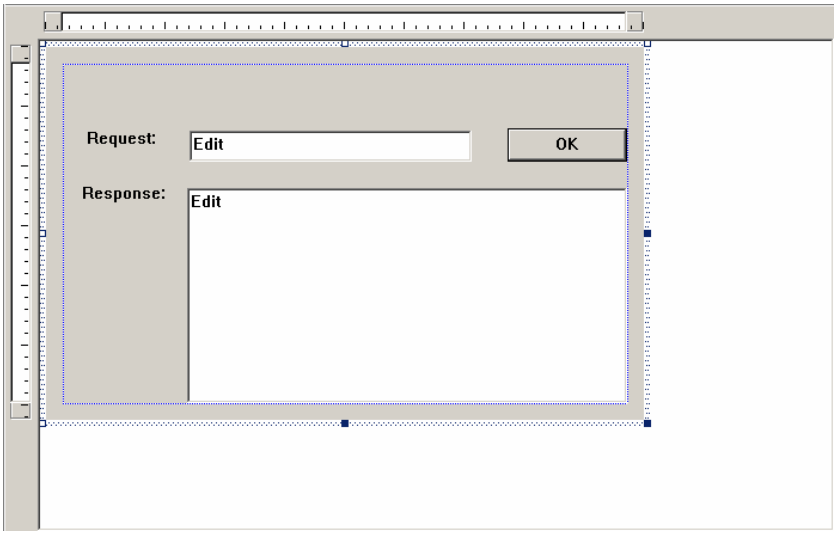


圖十、完成後的對話方塊

步驟三：由於這個對話方塊將以子頁籤的形式加入到 Remote Administration 的 Framework 中，所以還必須將這個對話方塊的「Style」標籤的「Style」屬性設成「Child」，而「Border」屬性設為「None」。



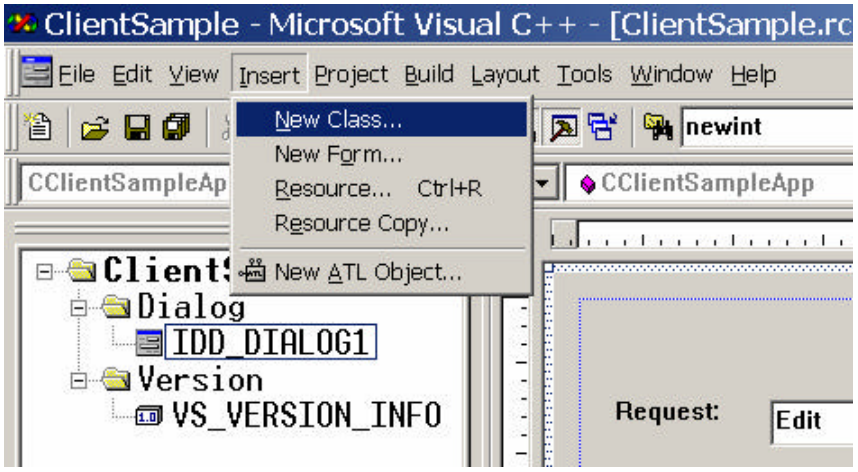
圖十一、設定 Style 屬性以及 Border 屬性



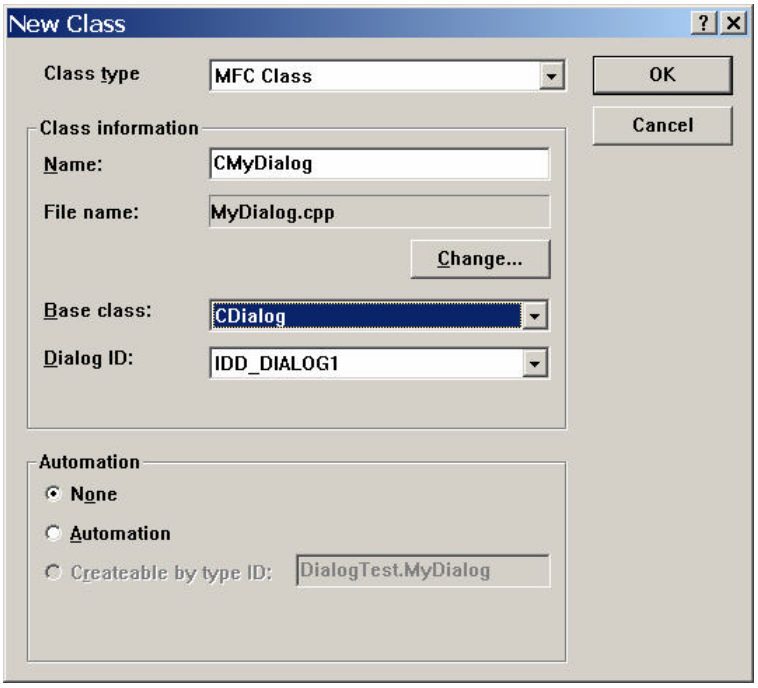
圖十二、最後顯示的無外框對話方塊資料

SECTION 03：加入對話方塊物件類別資料

在建立好對話方塊後，接下來就是將對話方塊的類別資料加到類別資料中。請按，選擇「New Class」，然後這目前的對話方塊，設定成繼承 CDialog 類別。



圖十三、選擇建立一個 New Class



圖十四、新的 Class **命名為** CMyDialog

SECTION 04：實作 Client Sample DLL 被呼叫的介面

接下來，我們要在 ClientSample.cpp **中，實作** 3 **個介面供** Remote Administration Framework **來呼叫，以取得。這三個介面分別是：**

- Initialize：**用來將** Plug in **的** ID，**以及** SendRequest、TransmitFile **的回應型態傳給**

DLL。

- EnumUI：**提供視窗數量，以及視窗的指標位置**
- ActivateUI：**供啟動視窗標籤**

步驟一：我們先在 Client Sample **中，加入以下程式碼：**

```
#CODE

#include "MyDialog.h"

extern "C"
{
//匯出三個 DLL 的介面供 Radm.exe 呼叫
    __declspec(dllexport) bool EnumUI(int* pCount,long* puid);
     __declspec(dllexport) HWND ActivateUI(long uid,HWND hParent);
     __declspec(dllexport) void Initialize(long id,SENDREQUEST,TRANSMITFILE);
}

//宣告三個 Public 物件，分別供傳遞 Plug-ins 的 ID 及傳遞命令或檔案使用
long                g_nPluginID;
SENDREQUEST     g_SendRequest;
TRANSMITFILE    g_TransmitFile;

#CODE
```

步驟二：宣告完就是實作的部分了，我們實作了 Initialize、EnumUI、**以及** ActivateUI **三部分供** Remote Administrator **呼叫使用。**

```
#CODE

//Initialize 實作
void Initialize(long id,SENDREQUEST SendRequest,TRANSMITFILE TransmitFile)
{
    g_nPluginID      = id;
    g_SendRequest   = SendRequest;
    g_TransmitFile = TransmitFile;
}
```

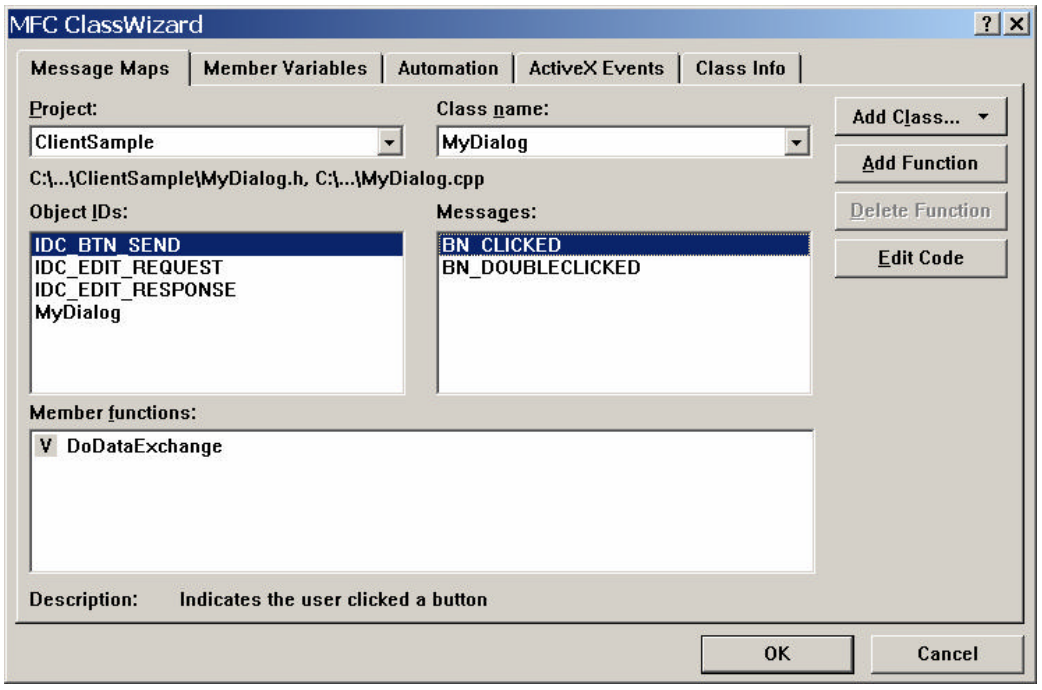
```
//EnumUI 實作，根據 puid 的值是不是 NULL，來做進一步的判斷

bool EnumUI(int* pCount,long* puid)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

   if(puid == NULL)
    {
        if(pCount == NULL)
              return false;
         else
             *pCount = 1;
    }
    else
    {
        puid[0] = 0;
    }
   return true;
}

/*取得 DLL 狀態的程式，然後根據對話方塊資源，取得對應的對話方塊 Handle 供呼叫*/

HWND ActivateUI(long uid,HWND hParent)
{
       AFX_MANAGE_STATE(AfxGetStaticModuleState());
       CDialog* pDlg = new MyDialog;
       if(pDlg->Create(MAKEINTRESOURCE(IDD_DIALOG1),CWnd::FromHandle(hParent)))
            return pDlg->m_hWnd;
else
{
            delete pDlg;
            return NULL;
}
}

#CODE
```

SECTION 05：傳送命令的處理方式

宣告完三個需要提供給 Remote Administration Framework 的基本 DLL 介面後，接下來就是實作：

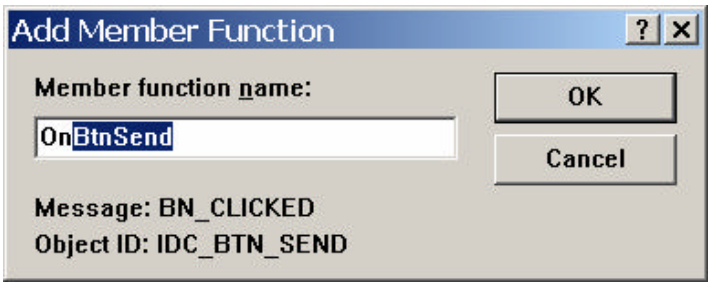
1. 如何從對話方塊送出命令
2. 以及由對話方塊接收回傳資料的方法

步驟一：我們在這邊使用 Class Wizard ([View] 下的 [Class Wizard] 或直接按下 「Ctrl」+「W」鍵啟動精靈。幫助完成 [Send] 按鈕按下後，將 Request 的訊息送給遠端，以及從遠端接收回應的程式碼的工作。

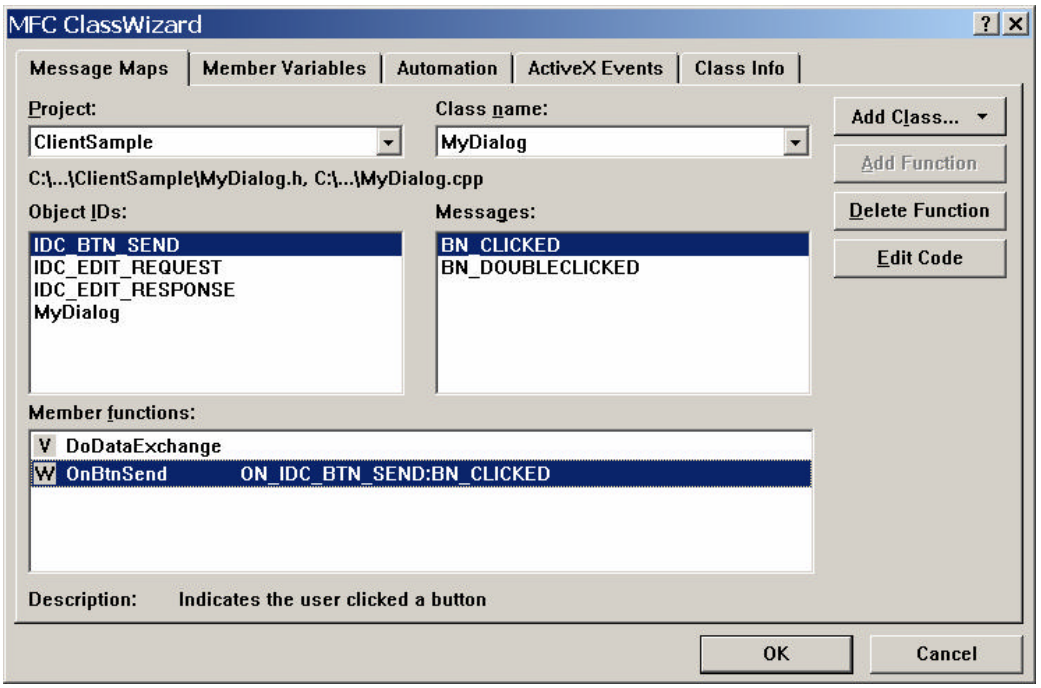


圖十五、處理按鈕按下後的事件

步驟二：在 Class name 選擇 MyDialog，接著在 Object IDs 選擇 IDC_BTN_SEND，然後選擇 Messages 的 BN_CLICKD 事件，最後按下 [Add Function] 鈕。



圖十六、處理 OnBtnSend 函式

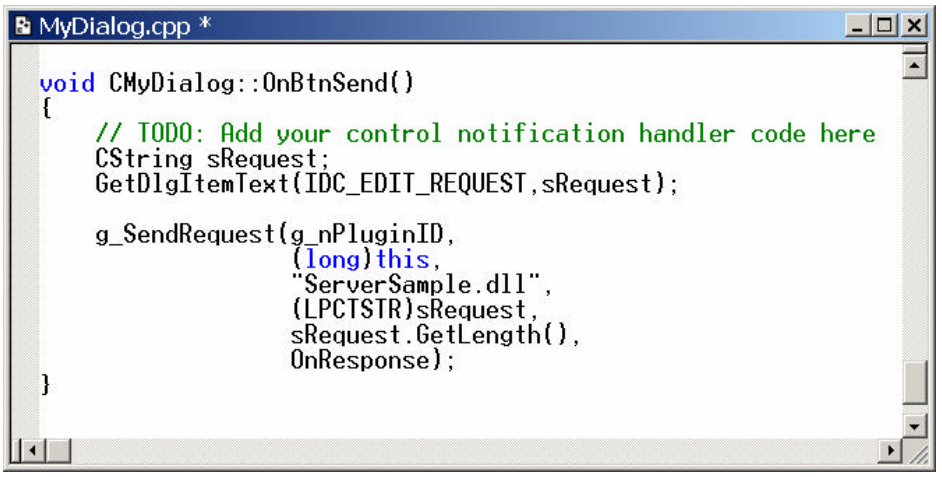


圖十七、確認增加 OnBtnSend 函式

步驟三：按下 [Edit Code] **鍵，編輯程式碼，請在** OnBtnSend() **函式中，加入以下的程式碼：**

```
#CODE

//設定 sRequest 當成
    CString sRequest;
    GetDlgItemText(IDC_EDIT_REQUEST,sRequest);

    g_SendRequest(g_nPluginID,
                  (long)this,
                  "ServerSample.dll",
                  (LPCTSTR)sRequest,
```

```
                    sRequest.GetLength(),
                    OnResponse);
}

#CODE
```



圖十八、OnBtnSend() 內容

在這一段程式碼中，我們將對話方塊中的命令字元，傳入 sRequest **中，然後在** g_SendRequest **函式中，分別傳入目前的：**

- g_nPluginID：**目前分配到的** Plug-ins ID
- (long)this：**對話方塊指標**
- **遠端呼叫的對應處理** dll **名稱，在此稱為** ServerSample.dll
- (LPCTSTR)sRequest：**命令文字**
- sRequest.GetLength()：**命令的長度**
- OnResponse：**回應的函式**

其中，OnResponse **函式是提供給遠端處理資料完後，供遠端回呼** (Callback) **使用。我們實作成為** OnResponse **程式。**

步驟四：最後，請在 MyDialog.h **的** Protected **宣告中，加入釋放視窗資源的宣告：**

```
virtual void PostNcDestroy();
```

然後在 MyDialog.cpp **中，加入實作釋放視窗資源的方法：**

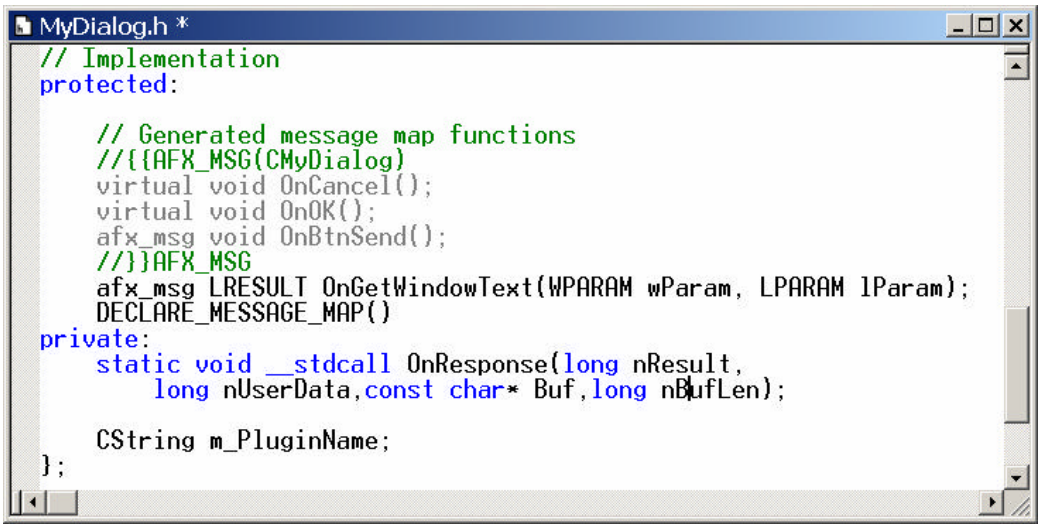
```
#CODE

void CMyDialog::PostNcDestroy()
{
	// TODO: Add your specialized code here and/or call the base class

	CDialog::PostNcDestroy();

	delete this;
}

#CODE
```

SECTION 06：Callback 的函式宣告與實做

在 OnResponse 中，我們提供傳入四個函數，其中第 3 個及第 4 個函數是提供遠端函式回傳資料。而第一個 nResult 則會傳回大於 0 的值。實際定義如下，關於函數設定的詳細資料，請參閱 ”Remote Administration Programming Guide” 的說明。

步驟一：請在 MyDialog 的 Class 中，加入一行宣告定義：

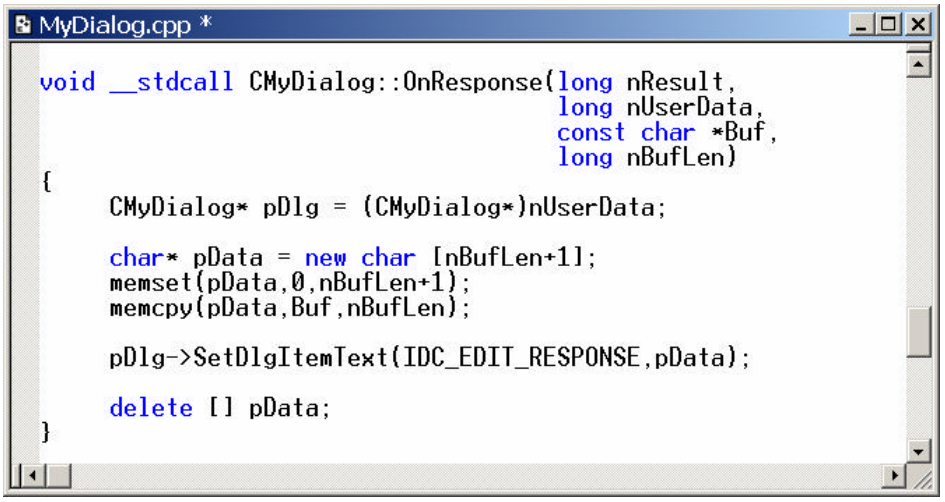
```
#CODE

static void __stdcall OnResponse (long nResult,long nUserData,const char* Buf,long nBufLen);

#CODE
```



圖十九、在 MyDialog.h 中，先加入 OnResponse 的宣告定義

步驟二：在 MyDialog.cpp 的實作程式碼中，將收到的結果，顯示在對話方塊中。

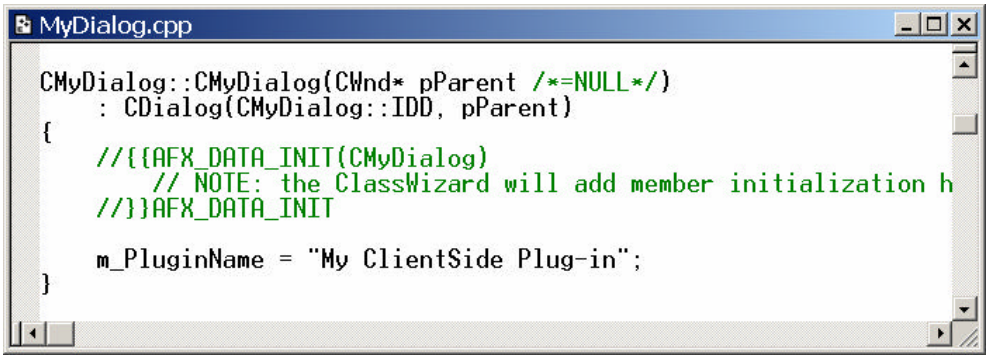
```
#CODE

void __stdcall CMyDialog::OnResponse(long nResult,
                                     long nUserData,
                                     const char *Buf,
                                     long nBufLen)
{
    CMyDialog* pDlg = (CMyDialog*)nUserData;

     char* pData = new char [nBufLen+1];
     memset(pData,0,nBufLen+1);
     memcpy(pData,Buf,nBufLen);

     pDlg->SetDlgItemText(IDC_EDIT_RESPONSE,pData);

     delete [] pData;
}

#CODE
```



圖二十、在 MyDialog.CPP **中，實作的** OnResponse **的程式碼**

在這邊我們用了 memset **及** memcpy **等** Windows **函式來設定對話方塊內** pData **指向的資料。然後利用** CDialog **的** SetDlgItemText **方法，將資料顯示在** IDC_EDIT_RESPONSE **上。**

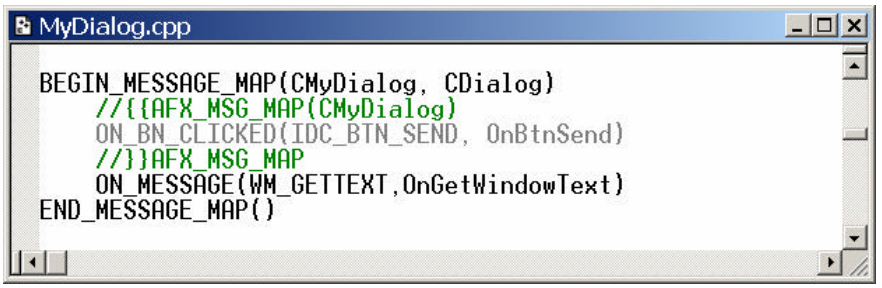
步驟三：視窗上設定 Plug-in **的標籤文字部分，以便在** MyDialog **初始化時，設定好對話方塊** m_PluginName **的名稱：**



圖二十一、加入 m_PluginName **的名稱**

為了讓程式能夠顯示 Plug-ins **的標籤，** Remote Administrator **會觸發一個** WM_GETTEXT **的視窗訊息。**

步驟四：在 MyDialog **的** Message Map **中，加入對應的處理函式** OnGetWindowText**。加入** WM_GETTEXT **的函式內容。**

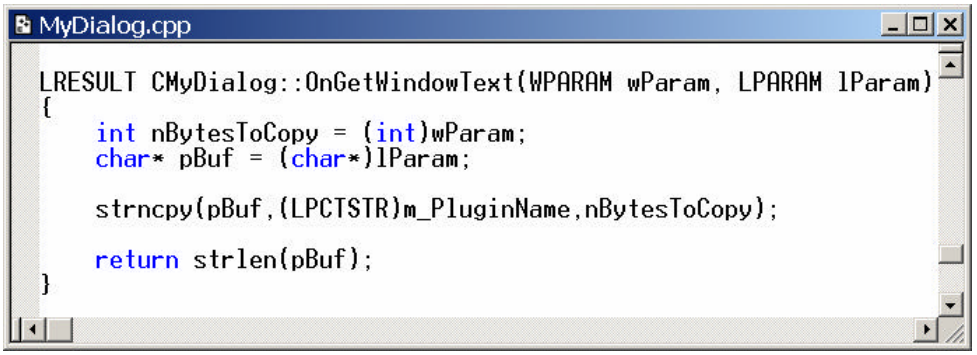


圖二十二、在 MyDialog 的 Message Map 中，加入 WM_GETTEXT 的處理函式

而 OnGetWindowText 的實作程式碼如下：

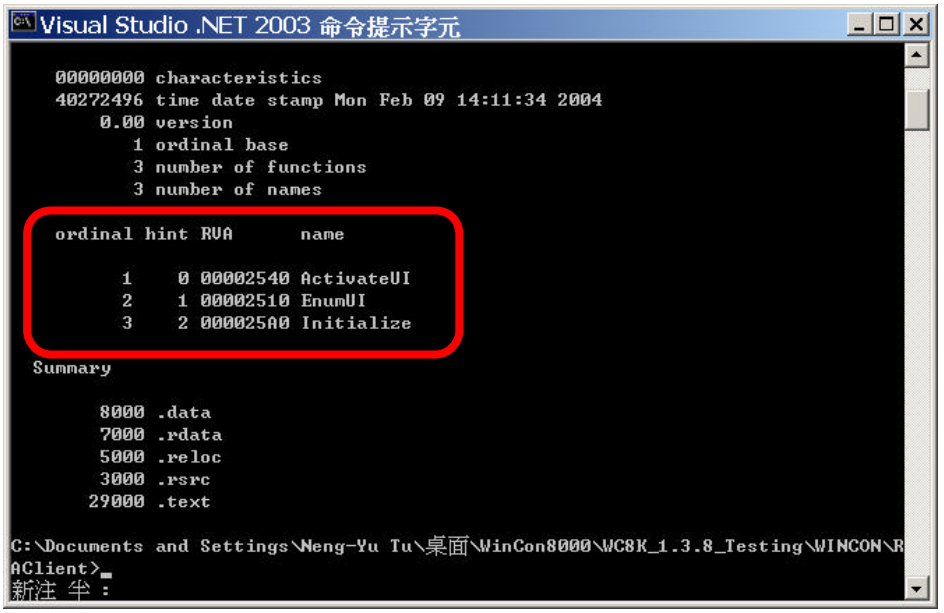
```
LRESULT CMyDialog::OnGetWindowText (WPARAM wParam, LPARAM lParam)
{
    int nBytesToCopy = (int)wParam;
     char* pBuf = (char*)lParam;

     strncpy(pBuf,(LPCTSTR)m_PluginName,nBytesToCopy);

     return strlen(pBuf);
}
```

此時 Plug-in 的名稱將會被送到 pBuf 中，然後會傳回 Plug-in 的名稱長度。



圖二十三、實作 OnGetWindoeText 函式的內容

編譯完成後，會得一個 ClientSample.dll 程式檔，我們可以使用 Dumpbin /Exports ClientSample.dll 看看 EnumUI、ActivateUI、Initialize 三個介面是否已經成功的被匯出。Dumpbin 程式在 VC++ 6.0 的 C:\Program Files\Microsoft Visual Studio\VC98\Bin 目錄，或是在 VC++ .NET 的 C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin 目錄下。



圖二十四、使用 Dumpbin / Exports clientsample.dll **的結果**

SECTION 07：伺服端的 DLL 設計 (WinCon-8000)

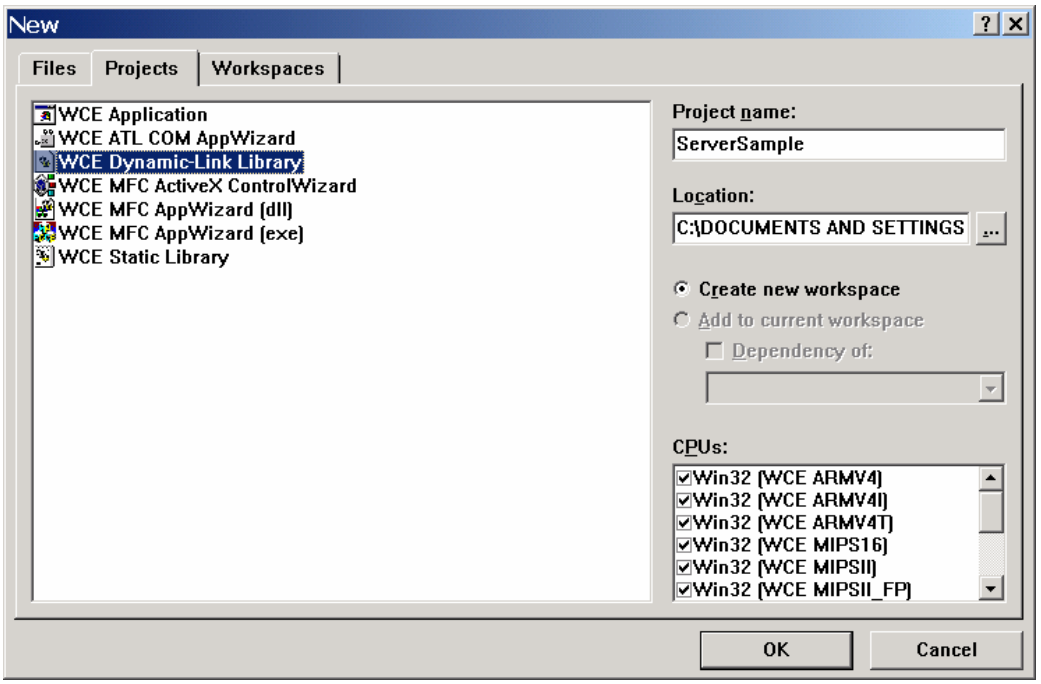
在 Windows CE **上的程式設計因為不用處理使用者介面的問題，所以只要實作接收到程式的處理部分就可以了。**

在進入程式設計之前，請先安裝：

- **微軟的** eMbedded Visual C++ 4.0
- eMbedded Visual C++ 4.0 Services Pack 2 **或是之後的版本**
- **對應** ARM **的** Platform SDK，**以編譯在** WinCon-8000 **上的動態連結** DLL **檔**

步驟一：開啟及設定 eMbedded Visual C++ **的環境**

在開啟 eMbedded Visual C++ **環境後，請選擇** WCE Dynamic-Link Library **後，選擇建立一個空白的** ServerSample **專案。**



圖二十五、建立一個 Server Sample 專案

SECTION 08：實作伺服端的回應訊息

步驟一：建立好 Server Sample 專案後，首先請在 ServerSample.cpp 的最前面，引入 Remote Administration Framwork 所需的 RAdmPlugin.h 標頭檔敘述：

```
#include "RAdmPlugin.h"
```

Remote Administration Framwork 的 Server 端需要開放三個介面供遠端呼叫，這三個介面分別是：

- Initialize( )：在這裡執行被呼叫 DLL 的初始化工作，同時可藉由兩個函數傳回 DLL 的版本資訊。
- ProcessMsg( )：處理接收訊息的實際處理函式位置，接收 PLUGIN_MESSAGE 結構 (struct) 的指標。
- UnloadPlugIn( )：在這裡可加入當 Plug-ins 被 Unload 的要執行的函式，例如像是釋放系統資源等。

步驟二：請加入三行 Server 端開放的介面：

```
#CODE

extern "C"
{

    __declspec(dllexport) void Initialize(long* major,long* minor);
    __declspec(dllexport) bool ProcessMsg(PLUGIN_MESSAGE* msg);
    __declspec(dllexport) void UnloadPlugIn();

};

#CODE
```

由於 eMbedded Visual C++ 會自動加入 Windows DLL 所需的 DllMain 進入點，本範例中我們不對 DllMain 做任何處理。

步驟三：設定 Initialize 函數

在 Initialize 函數中，可以提供 DLL 函式必要的初始值函數，在這邊我們僅回傳 DLL 的版本訊息。

```
#CODE

void Initialize(long* major, long* minor)
{
   if(major != NULL && minor != NULL)
     {
         *major = 1;
          *minor = 0;
     }
}

#CODE
```

步驟四：處理收到的訊息

我們會從 ProcessMsg( ) **收到** PLUGIN_MESSAGE **的** struct **指標，整個** struct **的結構如下。**

```
struct PLUGIN_MESSAGE
{
long ConnID;
long TotalBytes;
long StgHandle;
WRITEDATA       WriteData;
TRANSMITFILE TransmitFile;
};
```

其中第 1 **個引數是這個** DLL **的** ID**、第** 2 **個引數是傳入訊息引數的長度、第** 3 **個引數是訊息暫存檔的檔案** Handle。

而第 4 **個和第** 5 **個引數則是提供來讓資料傳回** Radm.dll **的** Callback **函數，當資料處理完後，可以將回應利用這兩個函數傳回。**

步驟五：在這邊我們接收一個 gettime **訊息，然後將訊息利用** WriteData **方法將資料回傳給** Radm.dll **後，傳回給客戶端。**

在下面的程式碼中，我們執行了下面幾個工作：

- **從傳入的** msg **取得接收指令暫存檔的** Handle
- **利用** ReadFile **將資料讀出做判斷處理**
- **取得系統時間**
- **將系統時間利用** WriteData **方法回傳**

```
#CODE

bool ProcessMsg(PLUGIN_MESSAGE* msg)
{
    DWORD    ByteReads = 0;
//取得指令暫存檔的 Handle
    HANDLE hFile = (HANDLE)msg->StgHandle;
    long nBufSize = GetFileSize(hFile,0);
```

```
        if(nBufSize == 0)
            return false;
    //取得指令的大小，並配置記憶體區塊
        char* pszCmd = new char [nBufSize + 1];
        memset (pszCmd,0,nBufSize + 1);
        bool nResult = false;
    //讀取指令暫存檔的內容
        if(ReadFile(hFile,pszCmd,nBufSize,&ByteReads,NULL))
        {
            _strlwr(pszCmd);
            char szMsg[256];
    //取得系統時間
            if(strcmp(pszCmd,"gettime") == 0)
            {
    //使用系統時間物件
                SYSTEMTIME  tmNow;
                GetLocalTime(&tmNow);
                sprintf(szMsg,"%d/%d/%d %d:%d:%d",
                        tmNow.wYear,
                        tmNow.wMonth,
                        tmNow.wDay,
                        tmNow.wHour,
                        tmNow.wMinute,
                        tmNow.wSecond);
            }
            else
                strcpy(szMsg,"Command not supported");
    //利用 WriteData 及先前得到的 ConnID 將資料 Callback 回 Radm.dll
            msg->WriteData(msg->ConnID,
                           szMsg,
                           strlen(szMsg),
                           NULL);
        }
        else
            nResult = false;

        delete [] pszCmd;
```

```
    return nResult;
}

#CODE
```

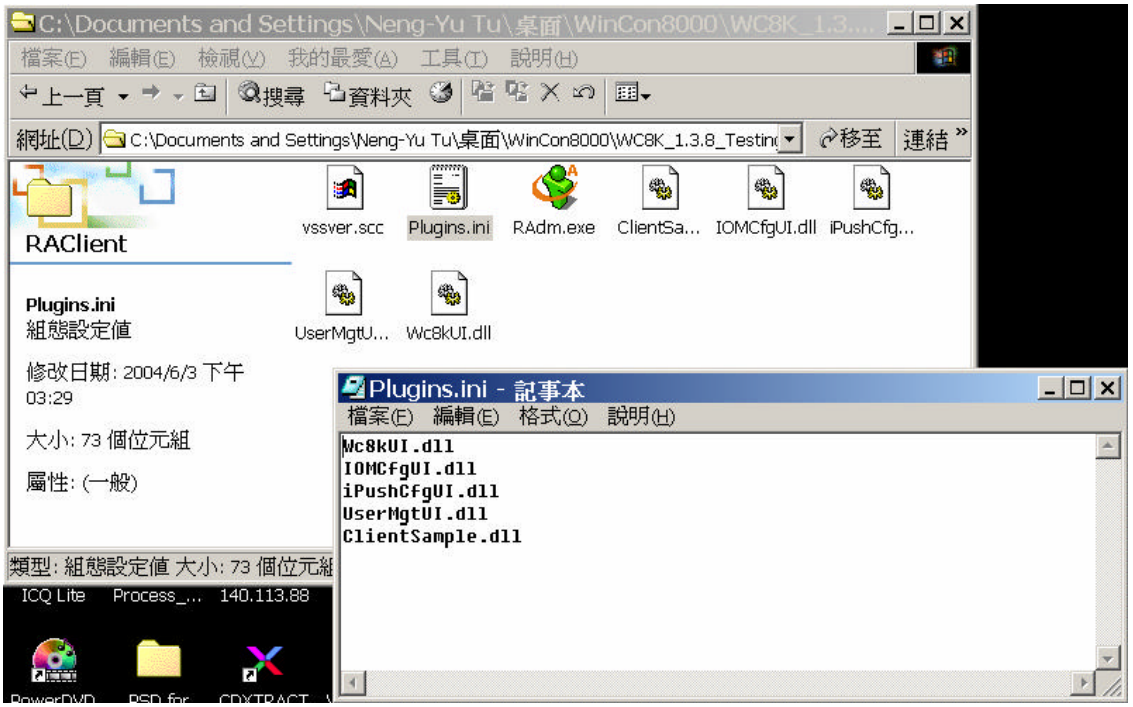
步驟六：請先確定您已經選取了 ARMV4 **及** Release **的編輯選項，然後請接著選取工具列上的**「Build」**下的**「Rebuild All」**來建立** DLL**。完成後，請檢視程式放置目錄的** Release **資料夾下，是否多了一個** ServerSample.dll **動態連結函式檔。**

SECTION 09：部署與測試功能

我們必須將 ClientSample.dll **及** ServerSample.dll **分別部署到** PC **端及伺服端。**

步驟一：PC **端的部署**

首先我們必須將 ClientSample.dll **複製到** PC **上的** RAClient **資料夾中，複製完成後，我們還必須修改** plugins.ini **檔，將新的** Plug-in **名稱加入檔案的內容中。**



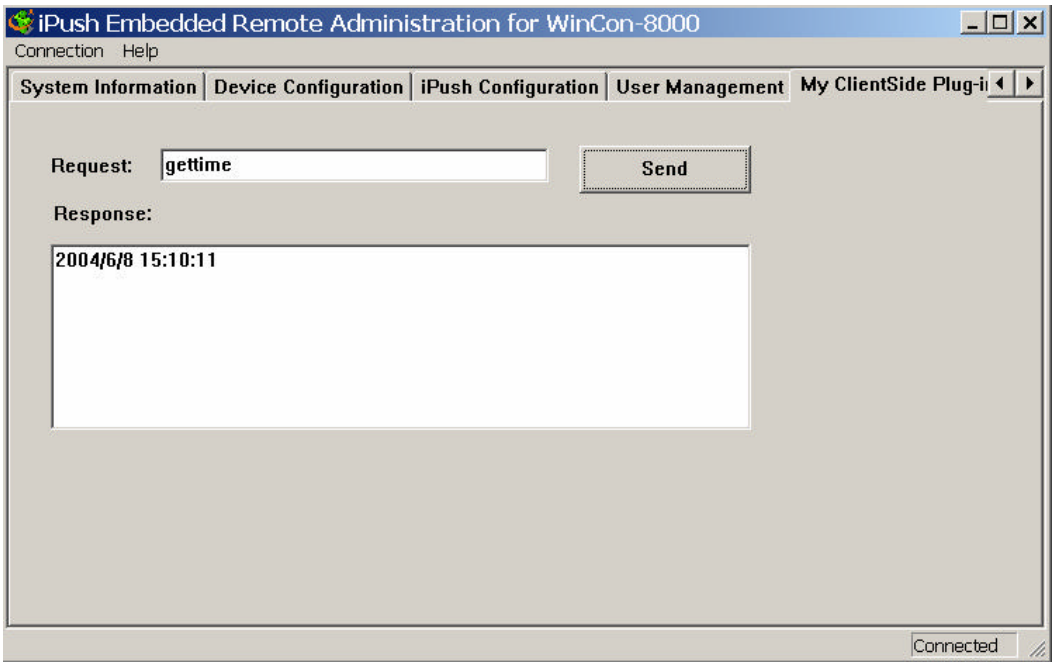
圖二十六、加入 ClientSample.dll **一行敘述**

步驟二：伺服端的部署

伺服端的部署比較單純，只要將 ServerSample.dll，**加入到** iPush Embedded **安裝目錄下的** RAdm **資料夾中就可以了。**

步驟三：測試並執行

最後請從客戶端執行 RAdm.exe，**並且以預設的使用者** wc8k **及密碼** wc8kadm **進入遠端管理程式。進入後，切換到** My Client Side Sample **標籤，在** Request**：旁的對話方塊中，輸入「**gettime」**後，按下「**Send」**鈕：**



圖二十七、輸入「gettime**」命令後，按下「**Send**」鈕的顯示結果**

我們最後可以在「Response**：」的對話方塊中，看到遠端** WinCon-8000 **上的系統時間。**