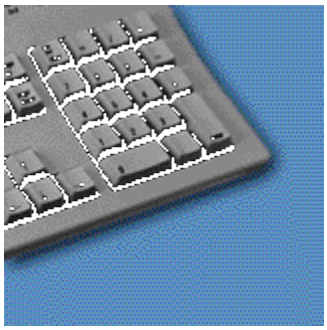


12. DDX의 소스분석

<장도비라>
이 장은 7장에서 작성한 Single 프로젝트의 대화상자 클래스인 CAboutDlg 부분의 코드를 분석하는 장이다. DDX(dialog data exchange)란 대화상자의 컨트롤과 대화상자의 래퍼 클래스가 가진 컨트롤에 대응된 멤버 함수가 서로의 값을 교환하는 방식을 말하는 것으로 MFC가 이를 구현한 방식에 대해서 살펴본다. 필자는 DDX를 설명하기 위해 CAboutDlg를 직접 사용하기 보다는 10장에서 구현한 RTTI 프로젝트에 대화상자를 추가할 것이고, 이를 단계별로 MFC의 것으로 변경해 나갈 것이다.
</장도비라>

DDX(dialog data exchange)란 대화상자의 컨트롤과 대화상자의 래퍼 클래스가 가진 컨트롤에 대응하는 멤버의 값을 서로 교환하는 방식이다. 살펴볼 내용은 아래와 같다.

- DDX의 원리
- CDataExchange의 동작 원리
- UpdateData()로 DDX를 동작시키는 원리
- DDX에 사용될 컨트롤과 멤버 변수를 클래스 위저드로 매핑하는 법

먼저, 메뉴에 반응하는 대화상자(dialog box)를 구현하기 위해 10장에서 완성된 RTTI 프로젝트 소스에 메뉴 리소스를 추가한다.

Generic 프로젝트의 작성

<절도비라>
대화상자의 원리를 이해하기 위해 Generic 프로젝트를 단계별로 작성한다. 대화상자에서도 메시지 맵을 사용할 것이고, 단계별 구현의 마지막에서는

MFC 대화상자의 자료 교환 방식인 DDX 매크로를 구현할 것이다.
</절도비라>

새로 만드는 프로젝트의 이름은 Generic이고□, 이 프로젝트에 아래의 리소

<여기서 잠깐>

기존의 프로젝트 이름을 변경하는 방법은 다음과 같다. 먼저 프로젝트 전체를 복사한다. 기존의 프로젝트 이름이 RTTI라 하자. 그러면, 텍스트 편집기를 이용하여 RTTI.dsw와 RTTI.dsp를 연다. RTTI문자열을 모두 찾아서 바꾸고자 하는 프로젝트 이름을 입력한다. 이때 RTTI.ico, RTTI.cpp등 프로젝트에 포함된 파일 이름이 RTTI를 포함하는 경우는 바꾸면 안된다. 텍스트 편집기를 종료하고, RTTI.dsw와 RTTI.dsp도 새이름으로 바꾼다. 혹은 www.codeproject.com에서 프로젝트 이름을 바꾸는 프로그램을 얻을 수 있다.
</여기서 잠깐>

스 스크립트 Generic.rc를 추가한다.

[예제 12.1] Generic.rc

```
#include "resource.h"

GENERIC                 ICON    DISCARDABLE     "generic.ico"

GENERIC MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Do it",                      IDM_DOIT
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       IDM_EXIT
    END
END
```

File→Do it에 대응하는 핸들러를 뷰에 설치한다. 뷰의 메시지 맵에 WM_COMMAND 핸들러를 추가한다.

```
class CView : public CWnd
{
public:
    DECLARE_DYNCREATE(CView)
    ...
```

```
    //{{AFX_MESSAGE_MAP
    ...
    void OnCommand(WPARAM, LPARAM);
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP()
};//class CView
```

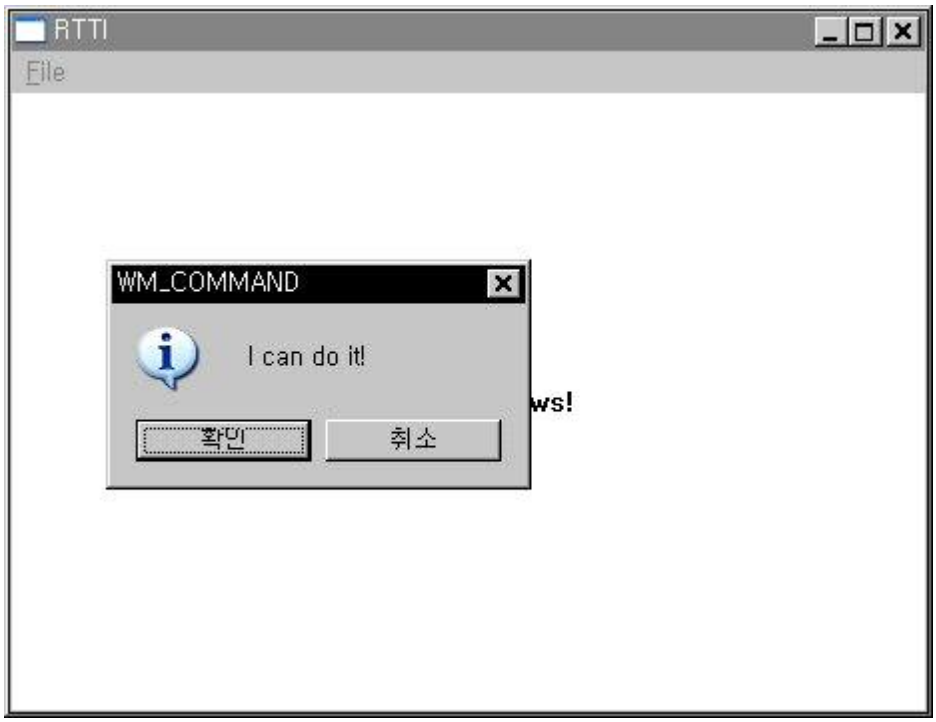
그리고 뷰의 구현 파일에 메시지 맵 엔트리를 추가하고,

```
BEGIN_MESSAGE_MAP(CView)
    { WM_PAINT,       CView::OnDraw },
    { WM_DESTROY,     CView::OnDestroy },
    { WM_LBUTTONDOWN, CView::OnLButtonDown },
    { WM_COMMAND,     CView::OnCommand },
END_MESSAGE_MAP()
```

OnCommand()를 구현한다.

```
void CView::OnCommand(WPARAM wParam, LPARAM lParam)
{
    switch(wParam)
    {
    case IDM_DOIT:
        MessageBox(GetSafeHwnd(),"I can do it!","WM_COMMAND",
            MB_OKCANCEL|MB_ICONINFORMATION);
        break;

    case IDM_EXIT:
        DestroyWindow(GetSafeHwnd());
        break;
    }//switch
}//CView::OnCommand()
```

메뉴가 발생시킨 WM_COMMAND메시지의 경우, wParam에 메뉴 ID가 넘어온다. 선택한 메뉴 항목이 File→Do it인 경우, MessageBox()를 이용하여 대화상자를 출력한다. 프로그램의 실행 결과는 아래 [그림 12.1]과 같다.



[그림 12.1] 첫 번째 Generic 프로젝트의 실행 화면: File→Do it을 선택하면 MessageBox()를 이용하여 대화상자를 출력한다.

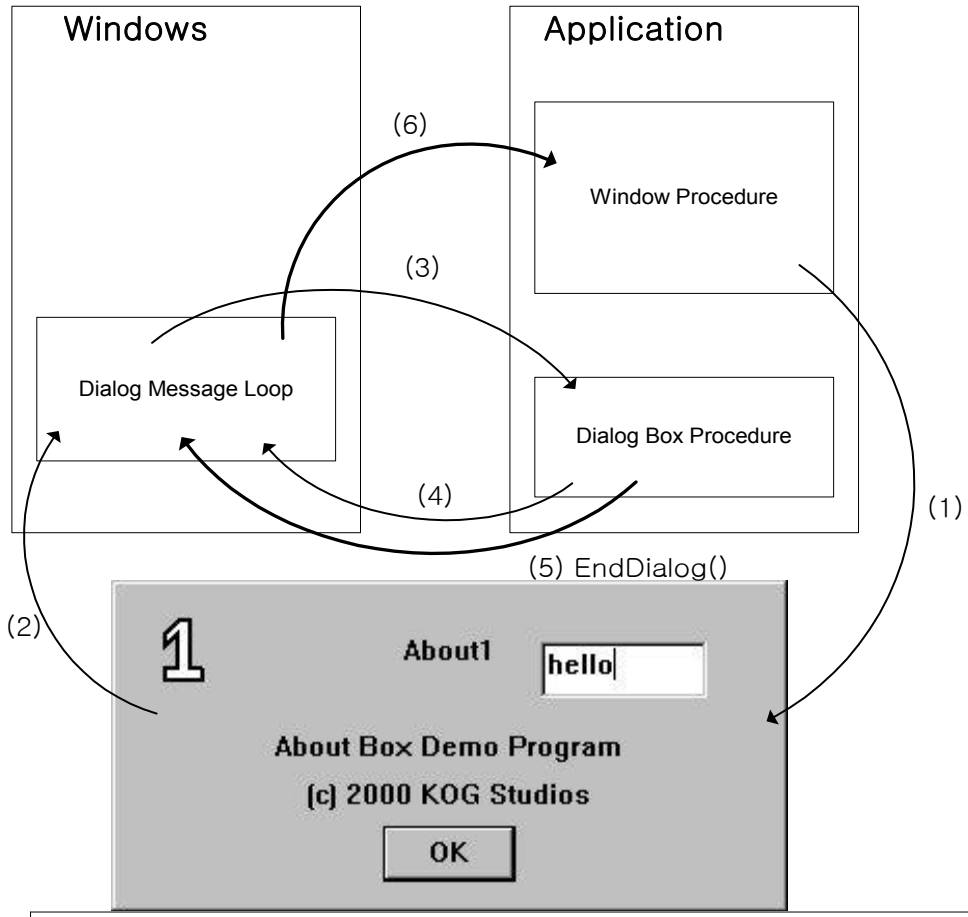
File→Exit을 선택하면 프로그램은 종료한다.

단계 1: 대화상자의 원리

클래스를 사용하지 않고 API함수를 직접 이용하여 **사용자 정의 대화상자(custom dialog box)**를 구현해 보자. 프로젝트에 추가할 대화상자의 모양은 아래 [그림 12.2]와 같다.

[그림 12.2] 사용자 정의 대화상자: 대화상자는 아이콘 그림, 세 개의 정적 텍스트 컨트롤, 한 개의 텍스트 박스와 한 개의 버튼을 포함한다.

모달(modal) 대화상자가 화면에 출력되었을 때, 많은 동작은 윈도우에 의해서 구현된다. 모달 대화상자를 화면에 출력하고 종료되기까지의 과정은 다음 [그림 12.3]과 같다.



[그림 12.3] 사용자 정의 대화상자의 동작: DialogBox()는 복잡한 내부 동작으로 구현된다.

1) 사용자는 DialogBox()를 호출하여, 대화상자를 출력한다. DialogBox()의 마지막 파라미터는 사용자 정의 대화상자 프로시저(custom dialog box procedure)의 시작 주소이다.

2) DialogBox()는 대화상자 윈도우를 만들고, 운영체제는 대화상자 메시지 루프(dialog message loop)에서 이벤트 처리를 시작한다.

3) 이벤트가 발생하면 메시지는 대화상자 프로시저로 전송된다.

4) 대화상자 프로시저는 필요한 메시지를 처리한다. 메시지를 처리한 경우, TRUE를 리턴하고 처리하지 않은 경우 FALSE를 리턴한다. FALSE를 리턴한 경우, 운영체제는 DialogPoc()을 호출하여 입력초점(input focus)의 이동 등 디폴트 처리를 수행한다.

5) 사용자는 대화상자를 종료하기 위해 EndDialog()를 호출한다. EndDialog()는 대화상자 메시지 루프를 탈출하라는 플래그를 설정하고, 리턴 값으로 두 번째 파라미터를 저장해 놓는다.

6) 종료 플래그가 설정되어 있으므로 메시지 루프는 종료되고, 저장된 값을 리턴한다.

이제 대화상자 스크립트를 Generic.rc파일에 추가한다. 내용을 아래에 리스트하였다. 굵게 표시된 부분이 추가된 리소스 스크립트이다.

[예제 12.2] 변경된 Generic.rc

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

About1      ICON    "about1.ico"
GENERIC     ICON    "generic.ico"

GENERIC MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Do it",                      IDM_DOIT
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       IDM_EXIT
    END
END

AboutBox DIALOG  20, 20, 160, 80
STYLE WS_POPUP | WS_DLGFRAME
{
     CTEXT "About1"                     -1,   0, 12, 160,  8
     ICON  "About1"                     -1,   8,  8,   0,  0
     CTEXT "About Box Demo Program"      -1,   0, 36, 160,  8
     CTEXT "(c) 2000 KOG Studios"       -1,   0, 48, 160,  8
     DEFPUSHBUTTON "OK"         IDOK,64, 60, 32, 14, WS_GROUP
     EDITTEXT IDC_EDIT1, 102, 14, 40, 14, ES_AUTOHSCROLL
}
```

대화상자의 텍스트 박스에 입력된 내용을 뷰에 저장하기 위해 뷰에 m_str[] 멤버를 하나 추가한다.

```
class CView : public CWnd
{
public:
```

```
    DECLARE_DYNCREATE(CView)

    char m_str[20];

    CView();
    ...
```

또한 대화상자에서 입력하는 동안의 내용을 저장하기 위해 뷰의 구현 파일에 하나의 전역변수 g_strEdit1[]을 선언한다. 입력된 내용을 취소할 수 있기 때문에 변수는 두개가 필요하다.

```
...
BEGIN_MESSAGE_MAP(CView)
    { WM_PAINT,       CView::OnDraw },
    { WM_DESTROY,     CView::OnDestroy },
    { WM_LBUTTONDOWN, CView::OnLButtonDown },
    { WM_COMMAND,     CView::OnCommand },
END_MESSAGE_MAP()

CWinApp app;

BOOL    CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;

char    g_strEdit1[20];

CView::CView()
{
    strcpy( m_str, "hello" );
}
...
```

이제 OnCommand() 핸들러는 다음과 같이 작성한다.

```
void CView::OnCommand(WPARAM wParam, LPARAM lParam)
{
    int iRet = 0;

    switch (wParam)
    {
    case IDM_DOIT:
        strcpy(g_strEdit1, m_str); // (1)
```

```
        iRet = DialogBox( app.GetInstanceHandle(), "AboutBox",
                          GetSafeHwnd(), AboutDlgProc ); // (2)

        if (iRet == 100) // (3)
        {
            strcpy(m_str, g_strEdit1);
        }//if
        break;

    case IDM_EXIT:
        DestroyWindow(GetSafeHwnd());
        break;
    }//switch
}//CView::OnCommand()
```

DialogBox()를 호출하기 전 가장 최근의 타당한 입력을 g_strEdit1[]에 복사한다(1). g_strEdit1[]의 내용이 대화상자의 텍스트 박스에 출력된다. 이제 DialogBox()를 호출하여 대화상자를 실행한다(2). 마지막 파라미터에 콜백함수(callback function)의 시작주소를 전달한다. 사용자가 OK를 선택하여 종료한 경우, 대화상자의 텍스트 박스의 내용을 뷰에 저장한다.

이제 AboutDlgProc()함수를 살펴보자.

```
BOOL CALLBACK AboutDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
                           LPARAM lParam)
{
    static HWND hEdit1;

    switch (iMsg) {
    case WM_INITDIALOG :
        hEdit1 = GetDlgItem(hDlg, IDC_EDIT1);
        SetDlgItemText(hDlg, IDC_EDIT1, g_strEdit1); // (1)
        return TRUE;

    case WM_COMMAND :
        switch (LOWORD (wParam))
        {
        case IDOK :
            GetDlgItemText(hDlg, IDC_EDIT1, g_strEdit1, 19); // (2)
            EndDialog(hDlg, 100); // (3)
              //                 ^ Note: this is the return value of
'DialogBox()'
```

```
            return TRUE;

        case IDCANCEL :
            EndDialog (hDlg, 0) ;
            return TRUE ;
        }
        break;
    }//switch
    return FALSE ; // (4)
}//AboutProc()
```

WM_INITDIALOG는 대화상자 컨트롤의 값을 초기화할 좋은 시점□이다.

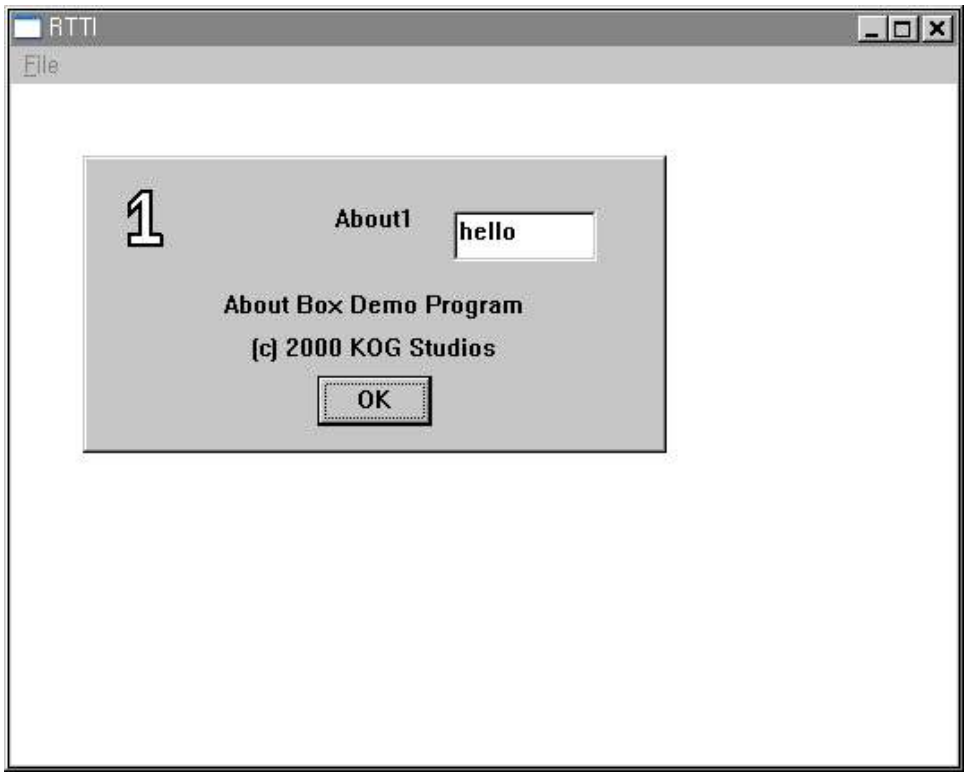
<여기서 잠깐>

WM_INITDIALOG가 발생했을 때, 대화상자는 화면에 표시되지 않았으므로, 데이터를 초기화하는 등의 작업은 가능하지만, 그리는 작업은 불가능하다.

</여기서 잠깐>

이 메시지에서 텍스트 박스의 내용을 g_strEdit1[]의 내용으로 초기화한다(1). 메시지를 처리하지 않은 경우는 FALSE를 리턴해야 한다(4). 사용자가 엔터키를 누르거나 OK버튼을 선택한 경우, 텍스트 박스에 입력된 내용을 g_strEdit1[]에 복사하고(2), EndDialog()를 호출하여 대화상자 메시지 루프를 종료할 것을 요청한다(3).

프로그램의 실행 결과는 아래 [그림 12.4]와 같다.



[그림 12.4] 대화상자의 출력: 대화상자가 출력되었을 때 텍스트 박스에 문자열을 입력한다. 문자열을 바꾸었더라도 사용자가 Esc키를 눌러 종료하면 다음 대화상자 출력에서 문자열의 내용은 바뀌지 않는다. 문자열의 내용을 바뀌게 하려면 반드시 OK버튼을 누르거나 엔터키로 대화상자를 종료해야 한다.

대화상자가 출력되었을 때 대화상자를 취소한 경우 바뀐 입력은 대화상자의 컨트롤에 반영되지 않아야 한다. 예를 들어 텍스트 박스에 world라고 입력하고 Esc를 누르면 다음 대화상자 출력은 이전 값인 hello를 보여준다. 이 동작을 위해 두 개의 변수를 사용했고, MFC는 이러한 동작을 DDX로 자동화한다.

단계 2: 메시지 맵의 사용

대화상자 프로시저도 메시지 맵을 이용해서 자동화하는 것이 가능하다. 하지만, 그전에 먼저 메시지 맵과 윈도우 프로시저에서 간과하고 넘어갔던 부분

을 다시 살펴볼 필요가 있다. 먼저 지금까지 사용했던 MessageMap 구조체를 다시 살펴보자.

```
typedef void (CView::*CViewFunPointer)(WPARAM, LPARAM);
typedef struct tagMessageMap
{
    UINT            iMsg;
    CViewFunPointer fp; // (1)
} MessageMap;
```

MessageMap의 두 번째 멤버는 뷰 클래스의 멤버 함수만을 가리킬 수 있다(1). 하지만, 윈도우 프로시저가 가리키는 대상 윈도우가 항상 뷰라고 가정할 수는 없다. 지금까지 사용했던 윈도우 프로시저를 보자.

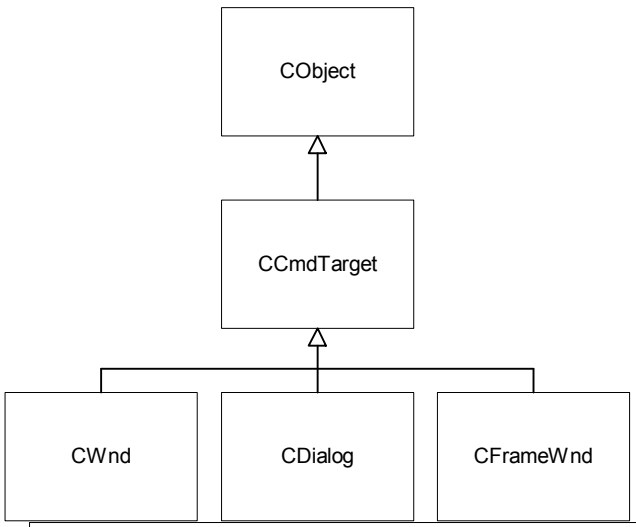
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg,
                         WPARAM wParam, LPARAM lParam)
{
    static CViewFunPointer  fpCViewGlobal = NULL;
    int                     i = 0;
    CView*                  p;

    while ( CView::messageMap[i].iMsg != 0 ) // (1)
    {
        if ( iMsg == CView::messageMap[i].iMsg )
        {
            fpCViewGlobal = CView::messageMap[i].fp;
            p = static_cast<CView*>(app.m_pMainWnd); // (2)
            (p->*fpCViewGlobal)(wParam, lParam); // (3)

            return 0;
        }//if
        ++i;
    }//while

    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc()
```

윈도우 프로시저는 항상 뷰의 메시지 맵 만을 검사하고(1), 응용 프로그램의 메인 윈도우를 뷰라고 가정하여(2), 뷰의 멤버 함수를 호출하고 있다. 하지만 윈도우 프로시저로 전달되는 윈도우 핸들이 가리키는 모든 대상이 뷰라고

가정할 수 없다. 사실 윈도우 프로시저의 윈도우 핸들을 전혀 이용하고 있지 않다!

하나의 윈도우 프로시저가 여러 대상 윈도우의 멤버 함수를 호출할 수 있는 해결책은 윈도우 프로시저가 받는 윈도우 핸들을 이용하여, 대상 윈도우 객체를 얻어 내는 것이다. 그리고 그 윈도우의 메시지 맵을 얻어 내고, 대상 클래스의 멤버 함수를 호출하도록 논리를 구성하는 것이다.

이 문제를 해결하기 위해 윈도우 메시지를 받을 수 있는 모든 클래스의 공통 베이스 클래스를 둘 필요가 있다. 이 클래스를 CCmdTarget이라 하자.



[그림 12.5] CCmdTarget 클래스: 윈도우 메시지를 받을 수 있는 모든 클래스는 CCmdTarget을 상속 받아서 구현한다.

CCmdTarget을 다음과 같이 간단하게 설계한다.

```
class CCmdTarget : public CObject
{
public:
    DECLARE_DYNCREATE(CCmdTarget)
    DECLARE_MESSAGE_MAP()
};//class CWnd
```

이제 CWnd는 CObject를 바로 상속받지 않고 CCmdTarget을 통해 상속 받는다.

```
class CWnd : public CCmdTarget
{
public:
    DECLARE_DYNCREATE(CWnd)
    ...
```

DECLARE_MESSAGE_MAP() 매크로에 메시지 맵 테이블을 얻어내는 가상 함수를 포함할 필요가 있다. 이 가상함수는 CCmdTarget의 하위 클래스가 무엇이든 메시지 맵 테이블을 얻어내는 역할을 한다. 수정된 매크로는 다음과 같다.

```
#define DECLARE_MESSAGE_MAP() \
    static MessageMap messageMap[];\
    virtual const MessageMap* GetMessageMap() const; // (1)

#define BEGIN_MESSAGE_MAP(class_name) \
    const MessageMap* class_name::GetMessageMap() const \
    { return class_name::messageMap; } \ // (2)
    MessageMap class_name::messageMap[] = {
```

DECLARE_MESSAGE_MAP()은 GetMessageMap()이라는 가상함수의 선언을 포함하고(1), BEGIN_MESSAGE_MAP()은 GetMessageMap()의 구현을 포함한다(2).

이제 메시지 맵을 다음과 같이 작성한다.

```
typedef void (CCmdTarget::*CCmdTargetFunPointer)(WPARAM, LPARAM);
typedef struct tagMessageMap
{
    UINT                    iMsg;
    CCmdTargetFunPointer    fp;
} MessageMap;
```

메시지 맵의 두 번째 멤버는 CCmdTarget 클래스의 멤버 함수를 가리킬 수 있다. 그래서 메시지 맵을 포함하는 모든 클래스는 반드시 CCmdTarget을 상속 받아 구현해야 한다.

이제 모든 윈도우에 공통으로 사용할 수 있는 윈도우 프로시저를 다음 [예제 12.3]과 같이 작성할 수 있다.

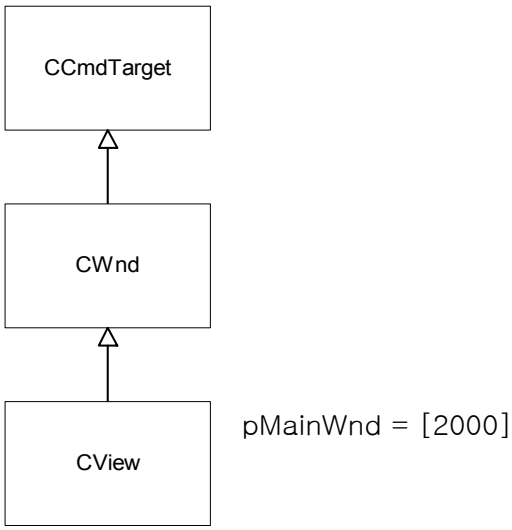
[예제 12.3] WndProc()

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam,
                         LPARAM lParam)
{
    int                     i = 0;
    CCmdTarget*             pMainWnd;
    const MessageMap*       pMessageMap;

    // get target window and it's message map
    pMainWnd    = CWnd::GetWndFromHandle(hwnd); // (1)
    pMessageMap = pMainWnd->GetMessageMap(); // (2)

    // iterate all message map entries
    while ( pMessageMap[i].iMsg != 0 ) // (3)
    {
        if ( iMsg == pMessageMap[i].iMsg ) // (3)
        {
            (pMainWnd->*pMessageMap[i].fp)(wParam, lParam); // (4)

            return 0;
        }//if
        ++i;
    }//while

    return DefWindowProc(hwnd, iMsg, wParam, lParam);
}//WndProc()
```

윈도우 프로시저는 먼저 특별한 함수를 호출하여 - 이 함수의 이름은 실제 MFC의 것과는 다르다 - 윈도우 프로시저에 전달된 윈도우 핸들 hwnd에 대응하는 윈도우 객체를 찾아 낸다(1). pMainWnd의 타입이 CCmdTarget*라는 것에 주의하자. 그리고 대상 윈도우의 메시지 맵을 얻어 낸다(2). GetMessageMap()이 가상함수이므로 가능하다. 이제 대상 윈도우 객체의 메시지 맵을 검사한다(3). 메시지 맵의 엔트리가 발견되면 해당 핸들러를 호출한다(4).

품을 수 있는 한 가지 의문은 pMainWnd는 CCmdTarget* 타입이고, 메시지 맵의 엔트리는 CView등 다른 타입인데 ->* 연산자를 이용하여 호출했을 때 바른 대상이 호출되는가? 하는 것이다. 아래의 설명을 읽지 말고 독자들 스스로 답을 찾아보기 바란다.

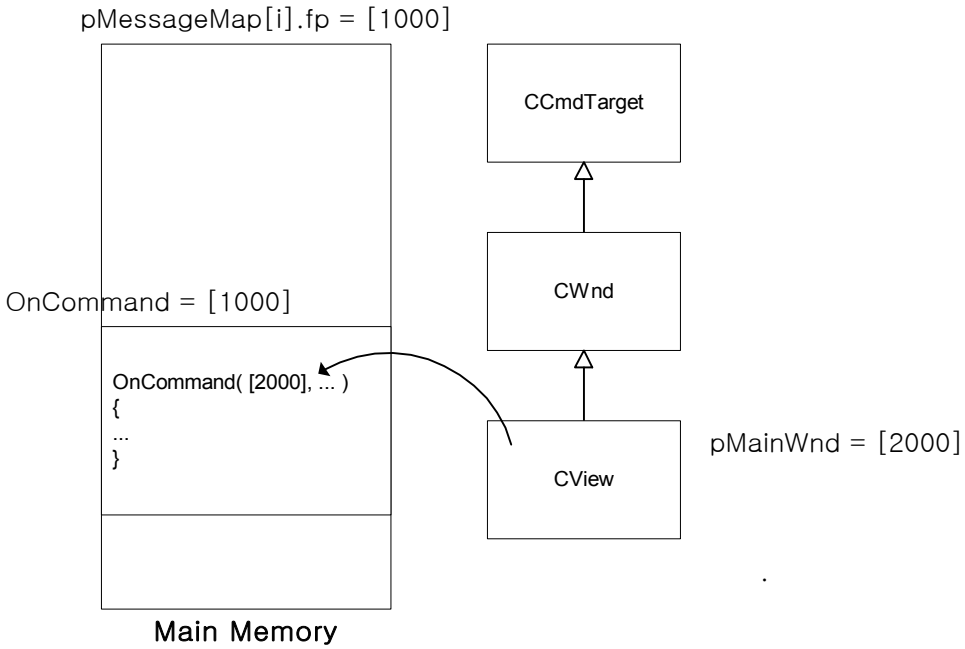
이에 대한 대답은 멤버 함수의 호출 메커니즘에 의해서 명확하다.



[그림 12.6] 객체의 시작 주소: 생성된 객체의 타입이 CView이고 그 객체의 시작 주소가 [2000]이라면 상위 클래스로의 업 캐스팅(up casting)은 서브타입의 원리를 만족하므로 안전하다. 또한, 값 [2000]이 변하는 것은 아니다.

생성된 객체의 주소는 캐스팅에 의해 변하지 않는다. 또한, 업 캐스팅(up casting)은 안전(safe)하다. 예를 들면 CView 객체를 만들었고 그 객체의 시작 주소가 [2000]이라면 CCmdTarget으로의 캐스팅은 안전하며 여전히 [2000]이다.

다음으로 살펴볼 내용은 멤버 함수의 호출 메커니즘이다.



pMainWnd->*pMessageMap[i].fp(wParam, lParam);

[그림 12.7] 멤버 함수의 호출: 멤버 함수의 호출은, 생성된 함수를 호출하는 첫 번째 파라미터로 객체의 시작 주소가 전달된다. 예를 들면, 메시지 맵의 엔트리가 [1000]이었고 이 함수가 OnCommand()라 하자. 그러면 pMainWnd->*pMessageMap[i].fp는 [1000]번지의 함수 OnCommand()를 호출하고 첫 번째 파라미터로 [2000]을 전달한다. 멤버 변수는 [2000]의 상대 주소로 번역되므로 안전하다.

일반 함수와 멤버 함수의 가장 큰 차이점은 멤버 함수는 멤버 변수를 객체의 시작 주소 this의 상대 주소로 번역한다는 점이다. 이러한 동작은 멤버 변수가 묵시적(implicit)으로 받는 첫 번째 파라미터인 객체의 시작 주소 this 때문에 가능하다. [1000]번지에 있는 함수가 CView의 멤버 함수 OnCommand()라 가정하자. 그러면 아래의 문장에서 OnCommand()는 모두 [1000]번지의 함수를 호출한다.

```
CView a;
CView b;
a.OnCommand();
b.OnCommand();
```

단 각각의 경우 첫 번째 파라미터로 a의 시작 주소와 b의 시작 주소가 전

달되는 것이다.

이제 메시지 맵의 함수 포인트 엔트리 pMessageMap[i].fp가 [1000]이었다고 하자. 그러면 아래의 문장은 [1000]번지의 함수 OnCommand()를 호출하면서 첫 번째 파라미터로 pMainWnd 즉 [2000]을 전달하는 것이다.

```
(pMainWnd->*pMessageMap[i].fp)(wParam, lParam);
```

그래서 윈도우 프로시저는 정확하게 동작한다.

대화상자 윈도우의 베이스 클래스로 사용될 CDialog를 [예제 12.4]처럼 설계한다.

[예제 12.4] class CDialog

```
#include "CObject.h"
#include "DataExchange.h"
#include "CWnd.h"

class CDialog : public CWnd // (1)
{
public:
    static HWND     m_hDlg;       // dialog window handle
    static CDialog* m_pDlgObject; // real dialog object pointer
    static CDialog* GetDialogFromHandle(HWND hDlg); // (2)

    CDialog(CDialog* pDlgObject);
    virtual ~CDialog();

    int UpdateData(int bSaveAndValidate = 1);
    int DoModal(); // (3)

protected:
    //virtual function
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
};//class CDialog
extern CWinApp app;

// Dialog Procedure -------------------------------------
int CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);
```

```
HWND     CDialog::m_hDlg         = NULL; // dialog window handle
CDialog* CDialog::m_pDlgObject   = NULL; // dialog object pointer

/*static*/ CDialog* CDialog::GetDialogFromHandle(HWND hDlg)
{
    m_hDlg = hDlg;
    return m_pDlgObject;
}//CDialog::GetDialogFromHandle()

CDialog::CDialog(CDialog* pDlgObject)
{
    m_pDlgObject = pDlgObject;     // pointer to a dialog object
}//CDialog::CDialog()

CDialog::~CDialog()
{
}//CDialog::~CDialog()

/*virtual*/ void CDialog::DoDataExchange(CDataExchange* pDX)
{
}//CDialog::DoDataExchange()

int CDialog::UpdateData(int bSaveAndValidate /*= 1*/)
{
    CDataExchange dx(bSaveAndValidate);

    DoDataExchange(&dx);
    return 1;
}//CDialog::UpdateData()

int CDialog::DoModal()
{
    return DialogBox( // (3)
        app.GetInstanceHandle(),        // application instance handle
        "AboutBox",                     // actually IDD will be used
        app.GetMainWnd()->GetSafeHwnd(),// parent window handle
        DlgProc );                      // dialog procedure
}//CDialog::DoModal()
```

CDialog 역시 윈도우이므로 CWnd를 상속받아 구현한다(1). 윈도우 핸들에

서 윈도우 객체를 얻어 내기 위해 자료구조 **맵(map)**□

<여기서 잠깐>

맵(map)이란 튜플(tuple)로 표현되는 쌍에서 한 쪽을 입력으로 받아 다른 한 쪽을 찾는 자료구조이다. 예를 들면 윈도우 객체의 시작 주소와 윈도우 핸들을 쌍으로 관리하는 맵에서 윈도우 핸들을 맵의 입력으로 주면 해당하는 윈도우 객체를 찾을 수 있다. 맵에서 검색 시간은 O(log*n*)이다.

</여기서 잠깐>

을 유지할 필요가 있다. 하지만, 코드를 간단히 하기 위해 대화상자 윈도우는 오직 한 개만 존재한다고 가정하자. 하지만 일관된 윈도우 프로시저 작성을 위해 다이얼로그 윈도우 핸들에서 다이얼로그 객체를 얻어 내는 함수를 GetDialogFromHandle()이라 하자(2). 하지만 이 함수는 오로지 한 객체만을 리턴한다. CDialog는 DialogBox()를 호출하는 DoModal() 멤버 함수를 가진다(3). 이제 대화 상자를 출력하기 위해 대화상자 객체를 만들고 DoModal()을 호출하면 된다.

대화상자와 일반적인 윈도우에 사용되는 공통된 윈도우 프로시저를 제작할 수 있다. 하지만, 코드를 간단하게 하기 위해 대화상자용 다이얼로그 프로시저를 별도로 만들자. [예제 12.5]의 DlgProc()은 CDialog 클래스의 구현 파일에 포함되어 있고 모든 대화상자의 다이얼로그 프로시저로 사용된다.

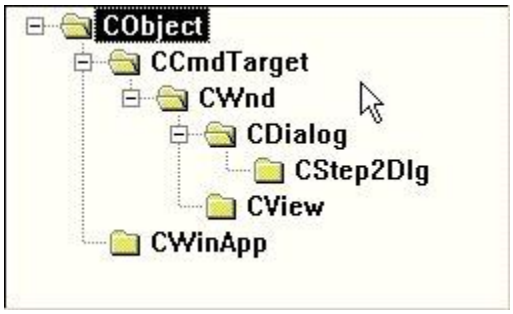
[예제 12.5] DlgProc()

```
int CALLBACK DlgProc(HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    int                     i = 0;
    CCmdTarget*             pDialogWnd;
    const MessageMap*       pMessageMap;

    pDialogWnd  = CDialog::GetDialogFromHandle(hDlg);
    pMessageMap = pDialogWnd->GetMessageMap();

    while ( 0 != pMessageMap[i].iMsg )
    {
        if ( LOWORD(iMsg) == pMessageMap[i].iMsg )
        {
            (pDialogWnd->*pMessageMap[i].fp)(wParam, lParam);
```

```
            return TRUE; // (1)
        }//if
        ++i;
    }//while

    return FALSE; // (1)
}//DlgProc()
```

일반적인 윈도우 프로시저와의 차이점은 메시지를 처리한 경우 TRUE를 그렇지 않은 경우 FALSE를 리턴 한다는 것이다(1). 그리고 디폴트 메시지 처리를 위한 DefWindowProc() 호출이 없다.

Generic 프로젝트의 클래스 계층도는 아래 [그림 12.8]과 같다. 사용자 정의 대화상자는 CDialog를 상속 받아 구현한다. 설계할 대화상자는 CStep2Dlg 클래스로 랩(wrap)한다.



[그림 12.8] Generic 프로젝트의 클래스 계층도: 모든 클래스의 공통 조상은 CObject이다. 메시지 핸들러를 가지는 모든 클래스는 CCmdTarget을 상속 받아 구현한다.

CStep2Dlg 클래스의 소스를 아래에 리스트하였다.

[예제 12.6] class CStep2Dlg.

```
#include "CObject.h"
#include "DataExchange.h"
#include "Dialog.h"
#include "Resource.h"
```

```
class CStep2Dlg : public CDialog // (1)
{
public:
    //{{AFX_DATA
    char m_strEdit1[20]; // (2)
    //}}AFX_DATA

public:
    CStep2Dlg(CObject* pParent = NULL);
    virtual ~CStep2Dlg();

    //{{AFX_VIRTUAL
    // virtual member functions
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

public:
    //{{AFX_MESSAGE_MAP
    void OnInitDialog(WPARAM wParam, LPARAM lParam); // (3)
    void OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP() // (3)
};//class CStep2Dlg

#pragma warning(disable: 4355)
  // 'this' used in base member initialization list

BEGIN_MESSAGE_MAP(CStep2Dlg) // (3)
    { WM_INITDIALOG, (CCmdTargetFunPointer)CStep2Dlg::OnInitDialog },
    { WM_COMMAND,    (CCmdTargetFunPointer)CStep2Dlg::OnCommand },
END_MESSAGE_MAP()

CStep2Dlg::CStep2Dlg(CObject* pParent /*= NULL*/) : CDialog(this)
{
    //{{AFX_DATA_INIT(CStep4Dlg)
    strcpy(m_strEdit1, "");
    //}}AFX_DATA_INIT
}//CStep2Dlg::CStep2Dlg()

CStep2Dlg::~CStep2Dlg()
{
```

```
}//CStep2Dlg::~CStep2Dlg()

void CStep2Dlg::DoDataExchange(CDataExchange* pDX)
{
}//void CStep2Dlg::DoDataExchange()

//-----------------------------------------------
// Name: OnInitDialog()
// Desc: this function may be virtual in MFC. Why?
void CStep2Dlg::OnInitDialog(WPARAM wParam, LPARAM lParam)
{
    //hEdit1 = GetDlgItem(hDlg, IDC_EDIT1);
    //SetFocus( hEdit1 );

    SetDlgItemText(m_hDlg, IDC_EDIT1, m_strEdit1); // (4)
}//CStep2Dlg::OnInitDialog()

void CStep2Dlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    switch ( LOWORD(wParam) )
    {
    case IDOK:
        GetDlgItemText(m_hDlg, IDC_EDIT1, m_strEdit1, 19); // (5)
        EndDialog(m_hDlg, IDOK);
        break;

    case IDCANCEL:
        EndDialog(m_hDlg, IDCANCEL) ;
        break;
    }//switch
}//CStep2Dlg::OnCommand()
```

CStep2Dlg는 CDialog를 상속받아 구현한다(1). 그리고 대화상자의 텍스트 박스 컨트롤의 내용을 담고 있을 멤버 변수를 선언한다(2). CStep2Dlg는 다른 일반적인 윈도우처럼 메시지 맵을 포함한다(3). WM_INITDIALOG 핸들러에서 텍스트 박스의 내용을 초기화하고(4), WM_COMMAND의 메시지 핸들러에서 OK버튼을 선택한 경우, 텍스트 박스의 내용을 멤버 변수로 가져오는 부분(5)에 주목하자.

DDX의 핵심은 텍스트 박스 같은 컨트롤과 이 컨트롤에 대응된 멤버 변수의 코드 생성을 자동화하는 것이다. 이의 핵심을 이루는 함수가 DoDataExchange()와 UpdateData(), 그리고 각 타입의 데이터 교환을 위한

DDX_로 시작하는 전역 함수들이다. 이것은 다음 단계에서 구현하기로 하고, 이제 대화상자를 생성하는 부분을 보자. 대화 상자의 생성은 뷰 클래스의 WM_COMMAND 핸들러에서 한다.

```
void CView::OnCommand(WPARAM wParam, LPARAM lParam)
{
    int         iRet = 0;
    CStep2Dlg   dlg; // (1)

    switch( LOWORD(wParam) )
    {
    case IDM_DOIT:
        strcpy(dlg.m_strEdit1, m_str); // (2)
        iRet = dlg.DoModal(); // (3)

        if (iRet == IDOK)
        {
            strcpy(m_str, dlg.m_strEdit1); // (4)
        }//if
        break;

    case IDM_EXIT:
        DestroyWindow(GetSafeHwnd());
        break;
    }//switch
}//CView::OnCommand()
```

대화상자 생성을 위해 객체를 만든다(1). 그리고 대화상자의 최근 값을 대화 상자 객체의 멤버로 복사한다(2). 이제 DoModal()을 호출하여 대화상자를 출력한다(3). 대화상자가 IDOK를 리턴한 경우, 대화상자의 입력값을 뷰에서 사용하기 위해 뷰의 멤버로 복사한다(4).

아래에 CView의 전체 소스를 리스트하였다.

[예제 12.7] CView.h

```
#include "stdafx.h"
#include "CWnd.h"

#ifndef _CView_
#define _CView_
```

```
class CView : public CWnd
{
public:
    DECLARE_DYNCREATE(CView)

    char m_str[20];

    CView();

public:
    //{{AFX_VIRTUAL
    // you may override some virtual functions
    // you know, there are two types of message map
    //
    //virtual void PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

    //{{AFX_MESSAGE_MAP
    void OnDraw(WPARAM, LPARAM);
    void OnDestroy(WPARAM, LPARAM);
    void OnLButtonDown(WPARAM, LPARAM);
    void OnCommand(WPARAM, LPARAM);
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP()
};//class CView

#endif // #ifndef _CView_
```

[예제 12.8] CView.cpp

```
#include "CWinApp.h"
#include "CView.h"
#include "resource.h"
#include "Step2Dlg.h"

IMPLEMENT_DYNCREATE(CView)

BEGIN_MESSAGE_MAP(CView)
    { WM_PAINT,       (CCmdTargetFunPointer)CView::OnDraw },
```

```
    { WM_DESTROY,     (CCmdTargetFunPointer)CView::OnDestroy },
    { WM_LBUTTONDOWN, (CCmdTargetFunPointer)CView::OnLButtonDown },
    { WM_COMMAND,     (CCmdTargetFunPointer)CView::OnCommand },
END_MESSAGE_MAP()

CWinApp app;

CView::CView()
{
    strcpy( m_str, "hello" );
}//CView::CView()

//CView Event handler-----------------------------------------
void CView::OnDraw(WPARAM, LPARAM)
{
    HDC             hdc;
    PAINTSTRUCT     ps;
    RECT            rect;

    hdc = BeginPaint( GetSafeHwnd(), &ps );
    GetClientRect( GetSafeHwnd(), &rect );
    DrawText( hdc, "Hello, Windows!", -1, &rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER );
    EndPaint( GetSafeHwnd(), &ps );
}//CView::OnDraw()

void CView::OnDestroy(WPARAM, LPARAM)
{
    PostQuitMessage(0);
}//CView::OnDestroy()

void CView::OnLButtonDown(WPARAM, LPARAM)
{
    HDC hdc;

    hdc = GetDC( GetSafeHwnd() );
    Ellipse( hdc, 0, 0, 300, 300 );
    ReleaseDC( GetSafeHwnd(), hdc );
}//CView::OnLButtonDown()

void CView::OnCommand(WPARAM wParam, LPARAM lParam)
{
    int         iRet = 0;
```

```
    CStep2Dlg   dlg;

    switch( LOWORD(wParam) )
    {
    case IDM_DOIT:
        strcpy(dlg.m_strEdit1, m_str);
        iRet = dlg.DoModal();

        if (iRet == IDOK)
        {
            strcpy(m_str, dlg.m_strEdit1);
        }//if
        break;

    case IDM_EXIT:
        DestroyWindow(GetSafeHwnd());
        break;
    }//switch
}//CView::OnCommand()
```

단계 3: DDX 매크로

다이얼로그 래퍼 클래스의 멤버 변수 값을 다이얼로그의 컨트롤로 전송하기 위해 SetDlgItemText(), 컨트롤의 값을 멤버 변수로 가져오기 위해 GetDlgItemText()를 사용했다. 이 두 가지를 동시에 하는 함수 UpdateData()를 설계할 수 있고, 가변적인 컨트롤의 값을 결정하기 위해 가상함수 DoDataExchange()를 설계할 수 있다.

DDX의 원리□는 다음과 같다.

<여기서잠깐>
이 장의 소스 분석은 DDV(dialog data validation)는 포함하지 않는다. DDV는 컨트롤에 입력되는 내용의 타당성을 검증하는 방식으로, DDX의 원리와 크게 다르지 않다고 판단했기 때문이다.
</여기서잠깐>

1) 사용자는 다이얼로그 데이터 교환을 위해 UpdateData()를 호출한다.
2) 다이얼로그 클래스의 베이스 클래스인 CDialog의 UpdateData()는 가상 함수 DoDataExchange()를 호출한다.
3) 사용자는 교환을 원하는 데이터를, 상속 받은 DoDataExchange()함수에 구현한다.

우리는 데이터 교환의 전송 방향을 결정하는 클래스 CDataExchange를 설계한다.

```
class CDataExchange
{
// Attributes
public:
    BOOL m_bSaveAndValidate;   // TRUE => save and validate data
    CWnd* m_pDlgWnd;           // container usually a dialog

    /*
// Operations (for implementors of DDX and DDV procs)
    HWND PrepareCtrl(int nIDC);     // return HWND of control
    HWND PrepareEditCtrl(int nIDC); // return HWND of control
    void Fail();                    // will throw exception

#ifndef _AFX_NO_OCC_SUPPORT
    CWnd* PrepareOleCtrl(int nIDC); // for OLE controls in dialog
#endif
    */
// Implementation
    CDataExchange(CWnd* pDlgWnd, BOOL bSaveAndValidate);
    /*
    HWND m_hWndLastControl;    // last control used (for validation)
    BOOL m_bEditLastControl;   // last control was an edit item
    */
};//class CDataExchange
```

이 클래스의 멤버 변수 m_bSaveAndValidate가 FALSE이면, 멤버 변수의 값을 컨트롤로 전송하는 것의 의미하고, TRUE이면 컨트롤의 값을 멤버 변수로 전송하는 것을 의미한다. m_pDlgWnd는 데이터 교환을 진행하는 윈도우 객체를 가리킨다

CDialog의 UpdateData()를 다음과 같이 구현한다.

```
int CDialog::UpdateData(int bSaveAndValidate /*= 1*/)
{
    CDataExchange dx(this, bSaveAndValidate);
```

```
    DoDataExchange(&dx);
    return 1;
}//CDialog::UpdateData()
```

UpdateData()는 CDataExchange 객체를 만들고 가상 함수 DoDataExchange()를 호출하는 간단한 일을 한다.

그리고 텍스트 박스의 데이터 교환을 위한

```
void DDX_Text(CDataExchange* pDX, UINT id, char* str);
```

함수를 CDialog에 추가한다. 실제로 MFC(SP5)는 모든 데이터 타입에 대해, DDX_함수를 제공한다. DDX_Text()는 다음과 같이 구현한다.

```
void DDX_Text(CDataExchange* pDX, UINT id, char* str)
{
    if ( pDX->m_bSaveAndValidate ) // (1)
    {
        GetDlgItemText(pDX->m_pDlgWnd->GetSafeHwnd(), id, str, 19);
    }
    else // (2)
    {
        SetDlgItemText(pDX->m_pDlgWnd->GetSafeHwnd(), id, str);
    }//if.. else..
}//DDX_Text()
```

DDX_Text()는 파라미터로 전달되는 CDataExchange 객체의 상태에 따라, 컨트롤로 값을 전송하거나(2), 컨트롤로부터 값을 읽는다(1).

이제 CDialog를 상속 받은 CStep2Dlg의 DoDataExchange()에서는 컨트롤에 매핑된 모든 멤버 변수에 대해 DDX_함수를 호출한다.

```
void CStep2Dlg::DoDataExchange(CDataExchange* pDX)
{
    //CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CStep2Dlg)
    DDX_Text(pDX, IDC_EDIT1, m_strEdit1);
    //}}AFX_DATA_MAP
}//void CStep2Dlg::DoDataExchange()
```

예에서는 컨트롤에 매핑된 변수가 텍스트 박스에 대응하는 m_strEdit1밖에 없다. CStep2Dlg 클래스의 소스를 [예제 12.9]에 리스트하였다.

[예제 12.9] CStep2Dlg.h

```
#if !defined(_CSTEP2DLG_DEFINED_)
#define _CSTEP2DLG_DEFINED_

#include "CObject.h"
#include "DataExchange.h"
#include "Dialog.h"
#include "Resource.h"

class CStep2Dlg : public CDialog
{
public:
    //{{AFX_DATA
    char m_strEdit1[20]; // (1)
    //}}AFX_DATA

public:
    CStep2Dlg(CObject* pParent = NULL);
    virtual ~CStep2Dlg();

    //{{AFX_VIRTUAL
    // virtual member functions
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

public:
    //{{AFX_MESSAGE_MAP
    void OnInitDialog(WPARAM wParam, LPARAM lParam);
    void OnCommand(WPARAM wParam, LPARAM lParam);
    void OnLButtonDown(WPARAM wParam, LPARAM lParam);
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP()
};//class CStep2Dlg

#endif // !defined(_CSTEP2DLG_DEFINED_)
```

[예제 12.10] CStep2Dlg.cpp

```
#include "stdafx.h"
#include <string.h>
#include "Step2Dlg.h"

#pragma warning(disable: 4355)
  // 'this' used in base member initialization list

BEGIN_MESSAGE_MAP(CStep2Dlg)
    { WM_INITDIALOG,  (CCmdTargetFunPointer)CStep2Dlg::OnInitDialog },
    { WM_COMMAND,     (CCmdTargetFunPointer)CStep2Dlg::OnCommand },
    { WM_LBUTTONDOWN, (CCmdTargetFunPointer)CStep2Dlg::OnLButtonDown },
END_MESSAGE_MAP()

CStep2Dlg::CStep2Dlg(CObject* pParent /*= NULL*/) : CDialog(this)
{
    //{{AFX_DATA_INIT(CStep4Dlg)
    strcpy(m_strEdit1, ""); // (2)
    //}}AFX_DATA_INIT
}//CStep2Dlg::CStep2Dlg()

CStep2Dlg::~CStep2Dlg()
{
}//CStep2Dlg::~CStep2Dlg()

void CStep2Dlg::DoDataExchange(CDataExchange* pDX)
{
    //CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CStep2Dlg)
    DDX_Text(pDX, IDC_EDIT1, m_strEdit1); // (3)
    //}}AFX_DATA_MAP
}//void CStep2Dlg::DoDataExchange()

//-----------------------------------------------
// Name: OnInitDialog()
// Desc: this function may be virtual in MFC. Why?
void CStep2Dlg::OnInitDialog(WPARAM wParam, LPARAM lParam)
{
    SetFocus( GetDlgItem(m_hDlg, IDC_EDIT1) );
```

```
//{{ seojt: 2003-11-12 오전 11:05:26
    //SetDlgItemText(m_hDlg, IDC_EDIT1, m_strEdit1);
    UpdateData(FALSE); // (4)
//}} seojt
}//CStep2Dlg::OnInitDialog()

void CStep2Dlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    switch ( LOWORD(wParam) )
    {
    case IDOK:
//{{ seojt: 2003-11-12 오전 11:05:22
        //GetDlgItemText(m_hDlg, IDC_EDIT1, m_strEdit1, 19);
        UpdateData(TRUE); // (4)
//}} seojt
        EndDialog(m_hDlg, IDOK);
        break;

    case IDCANCEL:
        EndDialog(m_hDlg, IDCANCEL) ;
        break;
    }//switch
}//CStep2Dlg::OnCommand()

//{{ seojt: 2003-11-12 오전 11:06:04
void CStep2Dlg::OnLButtonDown(WPARAM wParam, LPARAM lParam)
{
    HWND hwnd = GetDlgItem(m_hDlg, IDOK);

    UpdateData(TRUE); // (4)
    SetWindowText(hwnd, m_strEdit1);
}//CStep2Dlg::OnLButtonDown()
//}} seojt
```

MFC의 클래스 위저드(class wizard)는 컨트롤에 멤버 변수를 매핑할 때마다 세 곳의 코드를 변경한다. 하나는 클래스에 해당 멤버 변수를 선언하는 것이고(1), 둘째는 생성자에 멤버 변수를 초기화하는 것이고(2), 세 번째는 DoDataExchange()에 함수 호출을 추가하는 것이다(3). 이 부분은 클래스 위저드에 의해 유지되어야 하므로, 특별한 주석에 의해 감싸진다.

이제 멤버 변수의 값을 컨트롤로 전송해야 하는 경우에는

```
UpdateData(FALSE)
```

를 호출하고, 컨트롤의 값을 멤버 변수로 읽어 와야 하는 경우에는

```
UpdateData(TRUE)
```

를 호출하면 된다.

이제 DDX의 원리를 모두 이해했다. 이제 MFC에서 동작을 확인해 보자.

MFC 버전의 작성

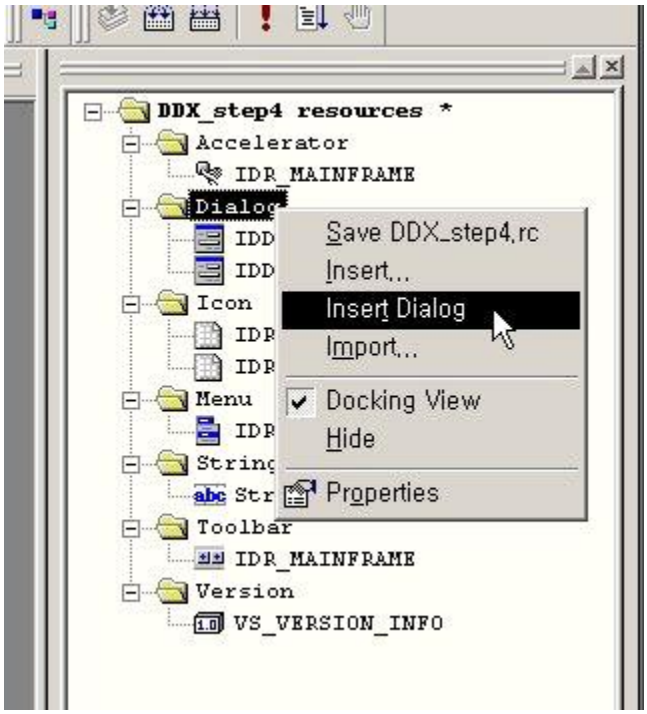
<절도비라>

이 절에서는 앞 절에서 작성한 Generic 프로젝트의 MFC 버전을 작성해보자. 이를 위해서 MFC 싱글 다큐먼트 프로젝트를 만들고 대화상자를 추가한 다음 DDX 코드를 추가해 볼 것이다.

</절도비라>

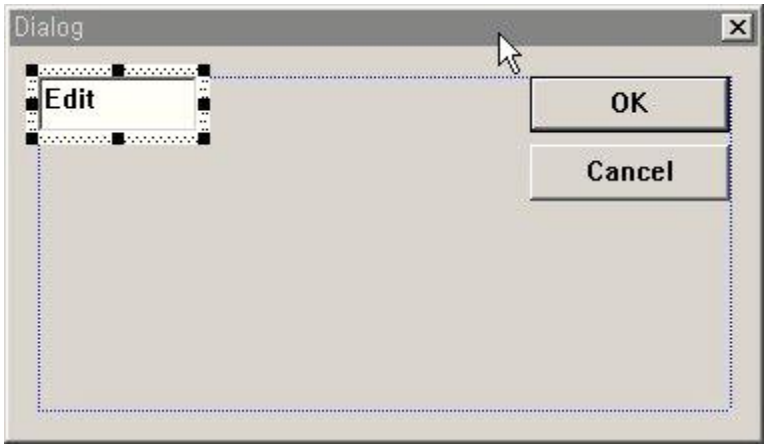
프로젝트 이름을 DDX_step4로 하여 MFC 싱글 다큐먼트 응용 프로그램을 만든다.

리소스 뷰의 Dialog에서 오른쪽 버튼을 눌러 Insert Dialog를 선택해서 다이얼로그를 리소스에 추가한다.



[그림 12.9] 다이얼로그의 추가: Insert Dialog로 IDD_DIALOG1을 리소스에 추가한다.

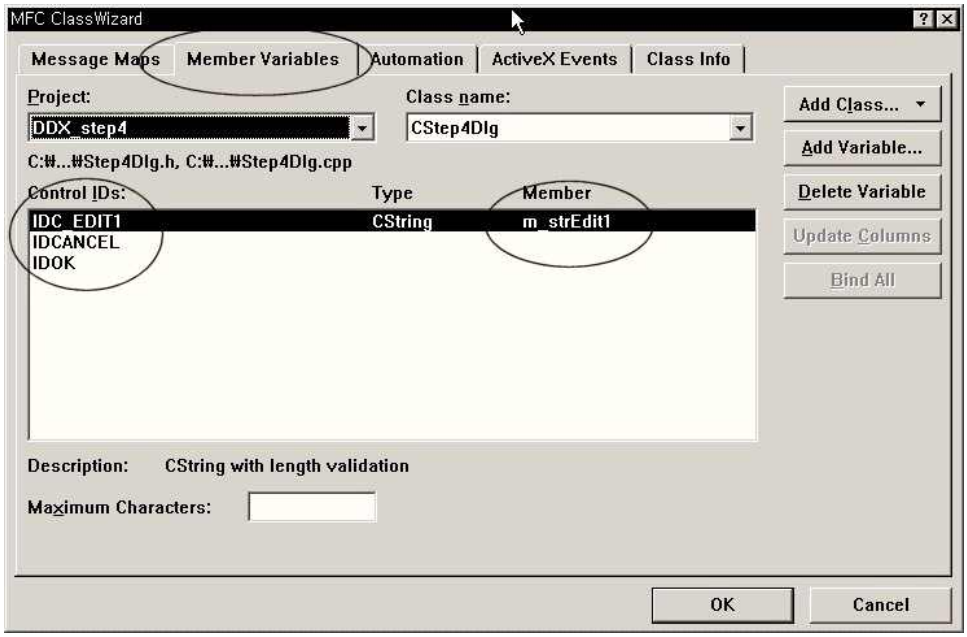
그리고 다이얼로그 편집기에서 IDC_EDIT1 텍스트 박스를 배치한다.



[그림 12.10] 텍스트 박스의 배치: 대화상자에 IDC_EDIT1을 ID로 가지는 텍스트

박슬르 배치한다.

이제 클래스 위저드를 실행하여, 배치된 컨트롤에 대응하는 멤버 변수를 추가한다. 래퍼 클래스의 이름을 CStep4Dlg로 하고, 멤버 변수를 m_strEdit1로 설정하자.



[그림 12.11] 클래스 위저드: 클래스 위저드를 실행하여 컨트롤에 대응하는 멤버 변수를 맵할 수 있다.

위와 같은 과정을 거치면 클래스 위저드는 소스의 세 군데를 변경한다. 이제 CStep4Dlg의 클래스 헤더에 m_strEdit1이 추가된 것을 확인할 수 있다.

```
class CStep4Dlg : public CDialog
{
// Construction
public:
    CStep4Dlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CStep4Dlg)
    enum { IDD = IDD_DIALOG1 }; // (1)
```

```
    CString m_strEdit1;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CStep4Dlg)
    protected:
    virtual  void  DoDataExchange(CDataExchange*  pDX);       //  DDX/DDV
support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CStep4Dlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

또한 CStep4Dlg의 구현 파일에서 추가된 두 곳의 소스를 확인할 수 있다.

```
// Step4Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "DDX_step4.h"
#include "Step4Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CStep4Dlg dialog
```

```
CStep4Dlg::CStep4Dlg(CWnd* pParent /*=NULL*/)
    : CDialog(CStep4Dlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CStep4Dlg)
    m_strEdit1 = _T(""); // (2)
    //}}AFX_DATA_INIT
}


void CStep4Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CStep4Dlg)
    DDX_Text(pDX, IDC_EDIT1, m_strEdit1);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CStep4Dlg, CDialog)
    //{{AFX_MSG_MAP(CStep4Dlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
/////
// CStep4Dlg message handlers
```

아래의 문제는 독자들 스스로 답해보기 바란다.

"(1) 클래스 헤더의 enum으로 선언된 다이얼로그 ID는 어떤 역할을 할까?"

"(2) 또한 문자열을 감싸는 _T("")는 왜 필요한가?"

뷰 클래스에 다음과 같은 테스트 루틴을 작성해 볼 수 있다.

```
void CDDX_step4View::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    int       iRet;
    CStep4Dlg dlg;
```

```
    dlg.m_strEdit1 = m_str;
    iRet = dlg.DoModal();
    if (iRet == IDOK) {
        strcpy(m_str, dlg.m_strEdit1);
    }//if

    CView::OnLButtonDown(nFlags, point);
}
```

응용 프로그램의 뷰 영역 - 클라이언트 영역 -에서 왼쪽 마우스 버튼을 누르면 대화상자를 출력한다. 텍스트 박스에 내용을 입력하고 OK버튼을 누른 경우, 입력한 내용은 대화상자를 다시 호출했을 때 나타난다.

MFC코드의 확인

<절도비라>

이 절에서는 대화상자의 DDX에 사용된 MFC의 실제 코드를 확인하고 구조를 다시 살펴본다. 실제 MFC 코드를 살펴보면서 개념을 확실하게 정립하고 이해하도록 하자.

</절도비라>

실제 MFC 코드를 확인해 보자. CDialog의 DoModal()을 DLGCORE.CPP의 493번 줄에서 확인할 수 있다.

```
int CDialog::DoModal()
{
    // can be constructed with a resource template or InitModalIndirect
    ASSERT(m_lpszTemplateName != NULL || m_hDialogTemplate != NULL ||
        m_lpDialogTemplate != NULL);

    // load resource as necessary
    LPCDLGTEMPLATE lpDialogTemplate = m_lpDialogTemplate;
    HGLOBAL hDialogTemplate = m_hDialogTemplate;
    HINSTANCE hInst = AfxGetResourceHandle();
    if (m_lpszTemplateName != NULL)
    {
```

```
        hInst = AfxFindResourceHandle(m_lpszTemplateName, RT_DIALOG);
        HRSRC hResource = ::FindResource(hInst, m_lpszTemplateName,
                                         RT_DIALOG);
        hDialogTemplate = LoadResource(hInst, hResource);
    }
    if (hDialogTemplate != NULL)
        lpDialogTemplate                                              =
(LPCDLGTEMPLATE)LockResource(hDialogTemplate);

    // return -1 in case of failure to load the dialog template resource
    if (lpDialogTemplate == NULL)
        return -1;

    // disable parent (before creating dialog)
    HWND hWndParent = PreModal();
    AfxUnhookWindowCreate();
    BOOL bEnableParent = FALSE;
    if (hWndParent != NULL && ::IsWindowEnabled(hWndParent))
    {
        ::EnableWindow(hWndParent, FALSE);
        bEnableParent = TRUE;
    }

    TRY
    {
        // create modeless dialog
        AfxHookWindowCreate(this);
        if (CreateDlgIndirect(lpDialogTemplate, // (1)
                        CWnd::FromHandle(hWndParent), hInst))
        {
            if (m_nFlags & WF_CONTINUEMODAL)
            {
                // enter modal loop
                DWORD dwFlags = MLF_SHOWONIDLE;
                if (GetStyle() & DS_NOIDLEMSG)
                    dwFlags |= MLF_NOIDLEMSG;
                VERIFY(RunModalLoop(dwFlags) == m_nModalResult);//(2)
            }

            // hide the window before enabling the parent, etc.
            if (m_hWnd != NULL)
                SetWindowPos(NULL, 0, 0, 0, 0, SWP_HIDEWINDOW|
                    SWP_NOSIZE|SWP_NOMOVE|SWP_NOACTIVATE|SWP_NOZORDER
```

```
        }
    }
    CATCH_ALL(e)
    {
        DELETE_EXCEPTION(e);
        m_nModalResult = -1;
    }
    END_CATCH_ALL

    if (bEnableParent)
        ::EnableWindow(hWndParent, TRUE);
    if (hWndParent != NULL && ::GetActiveWindow() == m_hWnd)
        ::SetActiveWindow(hWndParent);

    // destroy modal window
    DestroyWindow(); // (3)
    PostModal();

    // unlock/free resources as necessary
    if (m_lpszTemplateName != NULL || m_hDialogTemplate != NULL)
        UnlockResource(hDialogTemplate);
    if (m_lpszTemplateName != NULL)
        FreeResource(hDialogTemplate);

    return m_nModalResult;
}
```

DoModal()은 CreateDlgIndirect()를 호출하여 대화상자를 만들고(1), 메시지 루프를 실행한다(2). 모달 다이얼로그이기 때문에 메시지 루프에서 대기 상태가 되는 것이 가능하다. 메시지 루프를 탈출한 경우, DestroyWindow()를 호출하여 대화상자 윈도우를 파괴한다(3).

UpdateData()는 WINCORE.CPP의 3093번 줄에서 확인할 수 있다.

```
BOOL CWnd::UpdateData(BOOL bSaveAndValidate)
{
    ASSERT(::IsWindow(m_hWnd)); // calling UpdateData before DoModal?

    CDataExchange dx(this, bSaveAndValidate); // (1)

    // prevent control notifications from being dispatched during
```

```
UpdateData
    _AFX_THREAD_STATE* pThreadState = AfxGetThreadState();
    HWND hWndOldLockout = pThreadState->m_hLockoutNotifyWindow;
    ASSERT(hWndOldLockout != m_hWnd);   // must not recurse
    pThreadState->m_hLockoutNotifyWindow = m_hWnd;

    BOOL bOK = FALSE;       // assume failure
    TRY
    {
        DoDataExchange(&dx); // (2)
        bOK = TRUE;         // it worked
    }
    CATCH(CUserException, e)
    {
        // validation failed - user already alerted, fall through
        ASSERT(!bOK);
        // Note: DELETE_EXCEPTION_(e) not required
    }
    AND_CATCH_ALL(e)
    {
        // validation failed due to OOM or other resource failure
        e->ReportError(MB_ICONEXCLAMATION, AFX_IDP_INTERNAL_FAILURE);
        ASSERT(!bOK);
        DELETE_EXCEPTION(e);
    }
    END_CATCH_ALL

    pThreadState->m_hLockoutNotifyWindow = hWndOldLockout;
    return bOK;
}
```

UpdateData()는 CDataExchange 객체를 만들고, DoDataExchange()를 호출한다.

DDX_, DDV_로 시작하는 함수는 AFXDD_.H에서 확인할 수 있다.

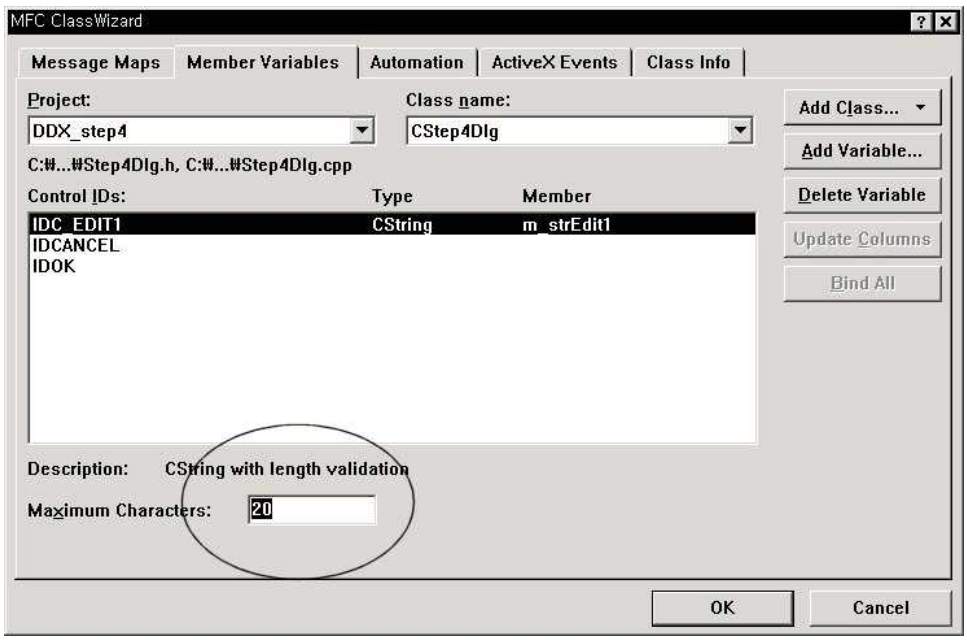
```
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1998 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
```

```
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// Do not include this file directly (included by AFXWIN.H)

/////////////////////////////////////////////////
// Standard Dialog Data Exchange routines

class COleCurrency;    // forward reference (see afxdisp.h)
class COleDateTime;    // forward reference (see afxdisp.h)

// simple text operations
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, BYTE& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, short& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, int& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, UINT& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, long& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, DWORD& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, CString& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, LPTSTR value,
                     int nMaxLen);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, float& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, double& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, COleCurrency& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, COleDateTime& value);

// special control types
void AFXAPI DDX_Check(CDataExchange* pDX, int nIDC, int& value);
void AFXAPI DDX_Radio(CDataExchange* pDX, int nIDC, int& value);
void AFXAPI DDX_LBString(CDataExchange* pDX, int nIDC, CString& value);
void AFXAPI DDX_CBString(CDataExchange* pDX, int nIDC, CString& value);
void AFXAPI DDX_LBIndex(CDataExchange* pDX, int nIDC, int& index);
void AFXAPI DDX_CBIndex(CDataExchange* pDX, int nIDC, int& index);
void AFXAPI DDX_LBStringExact(CDataExchange* pDX, int nIDC,
                              CString& value);
void AFXAPI DDX_CBStringExact(CDataExchange* pDX, int nIDC,
                              CString& value);
void AFXAPI DDX_Scroll(CDataExchange* pDX, int nIDC, int& value);
void AFXAPI DDX_Slider(CDataExchange* pDX, int nIDC, int& value);

void AFXAPI DDX_MonthCalCtrl(CDataExchange* pDX, int nIDC,
                             CTime& value);
void AFXAPI DDX_MonthCalCtrl(CDataExchange* pDX, int nIDC,
```

```
                              COleDateTime& value);
void AFXAPI DDX_DateTimeCtrl(CDataExchange* pDX, int nIDC,
                              CTime& value);
void AFXAPI DDX_DateTimeCtrl(CDataExchange* pDX, int nIDC,
                              COleDateTime& value);

// for getting access to the actual controls
void AFXAPI DDX_Control(CDataExchange* pDX, int nIDC, CWnd& rControl);

/////////////////////////////////////////////
// Standard Dialog Data Validation routines

// range - value must be >= minVal and <= maxVal
// NOTE: you will require casts for 'minVal' and 'maxVal' to use the
//   UINT, DWORD or float types
void AFXAPI DDV_MinMaxByte(CDataExchange* pDX, BYTE value, BYTE minVal,
                           BYTE maxVal);
void AFXAPI DDV_MinMaxShort(CDataExchange* pDX, short value,
                            short minVal, short maxVal);
void AFXAPI DDV_MinMaxInt(CDataExchange* pDX, int value, int minVal,
                          int maxVal);
void AFXAPI DDV_MinMaxLong(CDataExchange* pDX, long value, long minVal,
                           long maxVal);
void AFXAPI DDV_MinMaxUInt(CDataExchange* pDX, UINT value, UINT minVal,
                           UINT maxVal);
void AFXAPI DDV_MinMaxDWord(CDataExchange* pDX, DWORD value,
                            DWORD minVal, DWORD maxVal);
void AFXAPI DDV_MinMaxFloat(CDataExchange* pDX, float const& value,
                            float minVal, float maxVal);
void AFXAPI DDV_MinMaxDouble(CDataExchange* pDX, double const& value,
                             double minVal, double maxVal);

// special control types
void AFXAPI DDV_MinMaxSlider(CDataExchange* pDX, DWORD value,
                             DWORD minVal, DWORD maxVal);
void AFXAPI DDV_MinMaxDateTime(CDataExchange* pDX, CTime& refValue,
                               const CTime* refMinRange,
                               const CTime* refMaxRange);
void AFXAPI DDV_MinMaxDateTime(CDataExchange* pDX,
                               COleDateTime& refValue,
                               const COleDateTime* refMinRange,
                               const COleDateTime* refMaxRange);
void AFXAPI DDV_MinMaxMonth(CDataExchange* pDX, CTime& refValue,
```

```
                              const CTime* pMinRange,
                              const CTime* pMaxRange);
void AFXAPI DDV_MinMaxMonth(CDataExchange* pDX, COleDateTime& refValue,
                              const COleDateTime* refMinRange,
                              const COleDateTime* refMaxRange);


// number of characters
void AFXAPI DDV_MaxChars(CDataExchange* pDX, CString const& value,
                          int nChars);

/////////////////////////////////////////////////////////////////////////
```

DDV(dialog data validation) 함수는 대화상자 컨트롤의 값이 타당한지를 검사(validation)하는 함수이다. 각 컨트롤에 대한 제한 값을 입력한 경우 각 입력이 이 제한 값을 어기지 않도록 하는 코드가 DoDataExchange()에 추가된다. 예를 들면 에디트 컨트롤의 입력문자열의 길이가 20을 넘어가지 않도록 원한다면 아래 그림처럼 DDV값을 지정할 수 있다.



[그림 12.12] DDV: 에디터 컨트롤의 입력길이가 20을 넘지 않도록 지정하면 DoDataExchange()에 DDV 매크로가 추가된다.

요약

이 장에서 대화상자를 포함하는 Generic 프로젝트를 만들고 대화상자의 동작 원리를 살펴보았다. 그리고 MFC가 대화상자마다 래퍼 클래스를 만들고 래퍼 클래스의 멤버와 대화 상자의 컨트롤 사이에 데이터를 교환하는 방식인 DDX의 원리에 대해 살펴보았다.

- **DDX**란 대화상자의 컨트롤과 대화상자의 래퍼 클래스가 가진 컨트롤에 대응하는 멤버의 값을 서로 교환하는 방식이다.
- **DDV**는 대화상자 컨트롤의 값이 타당한지를 검사하는 방법이다.
- 대화상자는 자신을 위한 **대화상자 윈도우 프로시저**와 별도의 **대화상자 메시지 루프**를 가진다.
- 대화상자 데이터 전송의 방향을 결정하기 위해 **CDataExchange** 클래스를 사용한다.
- **UpdateData(FALSE)**는 컨트롤로 데이터를 전송하며, UpdateData(TRUE)는 컨트롤의 데이터를 멤버로 전송한다.
- **클래스 위저드**를 이용하면 대화상자에 배치된 컨트롤과 대응하는 래퍼 클래스의 멤버 변수를 매핑할 수 있다.

[문서의 끝]