

WEEK

1

DAY 5

다이얼로그 박스를 통해 사용자 입력을 받아내기

여러분이 사용하게 되는 대부분의 애플리케이션은 사용자로 하여금 애플리케이션의 작동 방식을 설정하게 하거나, 종료시 작업한 것을 저장할 것인가를 묻는 등이 필요한 경우에 많은 문제에 봉착한다. 이러한 대부분의 상황에서는 우리가 지금까지 사용했던 스타일의 윈도우를 새로 띄워 사용자에게 메시지를 전달하거나 입력을 받아내는 것이 보통인데, 이러한 윈도우를 다이얼로그 윈도우라고 한다.

다이얼로그 윈도우는 대개 한 개 이상의 컨트롤과 여러분들이 넣어주었으면 하는 정보를 설명하는 텍스트를 가지고 있으며, 워드프로세서나 프로그래밍 에디터에서처럼 무식하게 큰 빈 공간이 없다. 4장까지

공부하면서 만들었었던 모든 애플리케이션이 다이얼로그 윈도우를 기반으로 만들어진 것이며, 앞으로 몇 장 더 이러한 스타일로 만들게 될 것이다.

하지만, 지금까지 만든 애플리케이션은 다이얼로그 윈도우가 하나밖에 없는 것이었지만, 이번 장부터는 다르다.

- ◆ 다이얼로그 윈도우를 좀더 유연하게 사용하기
- ◆ 다른 다이얼로그 윈도우를 띄우고 이 윈도우를 통해 입력된 정보를 애플리케이션의 메인 윈도우로 되돌리기
- ◆ 메시지 박스와 같은 표준 다이얼로그와 지금까지 만들어왔었던 다이얼로그(커스텀 다이얼로그)를 동시에 사용하기

내장된 다이얼로그 윈도우를 먼저 사용해 보자

윈도우 운영체제에 이미 상당량의 내장 다이얼로그 윈도우가 준비되어 있다. 이 중에서 메시지 박스(MessageBox)라고 불리는 단순한 윈도우는 사용자에게 메시지를 보여주기 위한 용도로 쓰이며, 클릭할 수 있는 버튼을 3개까지 붙일 수 있다. 좀더 복잡한 용도로 사용되는 내장 다이얼로그 윈도우에는 파일 열기(File Open), 파일 저장(File Save), 인쇄(Print) 다이얼로그 등이 포함되는데, 많은 애플리케이션에서 동일한 목적으로 사용될 목적으로 만들어졌다는 뜻에서 공통(Common) 다이얼로그라고 불리며, 여러 가지의 C++ 클래스로 포장되어 인스턴스처럼 사용된다.

■ 메시지 박스 사용하기

지나온 장에서 배웠듯이, 메시지 박스의 사용은 함수 호출 한번(메시지 텍스트를 인수로 한)으로 끝난다. 사용자에게 보여줄 메시지와 아이콘, 그리고 “잘 읽었다. 이제 닫아라”란 뜻의 버튼을 가진 다이얼로그 박스가 하나 나타날 것이다. 다른 윈도우 소프트웨어를 사용하면서 이미 알고 있겠지만, 메시지 박스에 붙은 버튼과 아이콘을 적절히 조합해서 얼마든지 원하는 용도의(물론 간단한 것이지만) 다이얼로그 기능을 수행하도록 할 수 있다.

MessageBox() 함수

MessageBox() 함수는 세 개의 인수를 받는데, 그 중에서 첫번째 인수가 가장 중요한 메시지 텍스트이다. 두번째 인수는 넣어도 되고, 넣지 않아도 되는 것인데, 메시지 박스의 타이틀바에 들어갈 텍스트이다. 세번째 인수 역시 선택적으로 쓰이며, 사용자에게 보일 아이콘과 버튼을 나타내는 플래그의 조합값이다. MessageBox()는 사용자가 어떤 버튼을 눌렀는지에 대한 ID 값을 반환하는데, 바로 이 ID 값은 세번째 인수로 넣어진 버튼 조합 ID가 무엇이었는가에 따라 달라진다.

Note

만약에 버튼과 아이콘을 나타내는 조합 ID를 세번째 인수로 사용하고 싶다면 두번째 인수(메시지 박스의 타이틀)도 반드시 넣어주어야 한다.

MessageBox()에 사용할 수 있는 버튼 조합도 무제한은 아니다. [표 5.1]에 나와 있는 플래그 값 밖에 사용할 수 없기 때문이다. 만약에 여러분의 용도에 맞는 버튼 스타일이 없거나 마음에 들지 않는다면 메시지 박스처럼 보이는 커스텀 다이얼로그 윈도우를 직접 만들면 된다.

[표 5.1] MessageBox()의 버튼 조합 ID

ID	버튼
MB_ABORTRETRYIGNORE	취소(Abort), 재시도(Retry), 무시(Ignore)
MB_OK	확인(OK)
MB_OKCANCEL	확인(OK), 취소(Cancel)
MB_RETRYCANCEL	재시도(Retry), 취소(Cancel)
MB_YESNO	예(Yes), 아니오(No)
MB_YESNOCANCEL	예(Yes), 아니오(No), 취소(Cancel)

표시되는 아이콘을 설정하려면 아이콘 ID를 버튼 조합 ID에 추가시켜 주면 되며, [표 5.2]를 참고하기 바란다. 만약 아이콘 조합이나 버튼 조합 중 하나만 세번째 인수로 넘기면, 사용되지 않은 부분은 디폴트 값으로 남는다.

일단 버튼 조합 ID를 설정해 두었으면 사용자가 누른 버튼이 어떤 아이콘인지도 알 아낼 수 있다. MessageBox() 함수의 리턴값이 바로 그 대상이며, [표 5.3]을 보도록 하자.

[표 5.2] MessageBox() 아이콘 ID

ID	아이콘
MB_ICONINFORMATION	i자 아이콘
MB_ICONQUESTION	물음표 아이콘
MB_ICONSTOP	중지 표시 아이콘
MB_ICONEXCLAMATION	느낌표 아이콘

[표 5.3] MessageBox()가 반환하는 ID

ID	클릭된 버튼
IDABORT	취소(Abort)
IDRETRY	재시도(Retry)
IDIGNORE	무시(Ignore)
IDYES	예(Yes)
IDNO	아니오(No)
IDOK	확인(OK)
IDCANCEL	취소(Cancel)

메시지 박스를 사용하는 애플리케이션을 만들자

MessageBox() 함수를 어떻게 사용할지 머릿속에 두고 끙끙대는 것보다는 직접 간단한 애플리케이션을 만들어 보는 것이 낫다. 여러분이 이제 만들 애플리케이션은 두 개의 버튼을 가지고 있는데, 버튼을 누르면 각각 다른 메시지 박스가 뜨게 되어 있다. 순전히 MessageBox() 함수의 여러 가지 선택 사항을 확인할 수 있도록 만든 예제이니까 후회 말고 잘 따라오기 바란다. 이번 장의 후반부에서는 파일 열기(File Open)ダイ얼로그를 사용해서 파일 이름을 설정하는 등의 표준 기능을 사용할 수 있게도 해볼 것이다. 그리고 마지막에는 여러분이 직접 만든 파일로그 윈도우를 띄우고, 사용자가 여기에 어떤 정보를 입력하고 파일로그 윈도우를 닫으면 메인 애플리케이션 파일로그에서 직접 이 값을 읽어낼 수 있는 방법까지 공부하도록 하겠다.

자, 이제 시작해 보도록 하자.

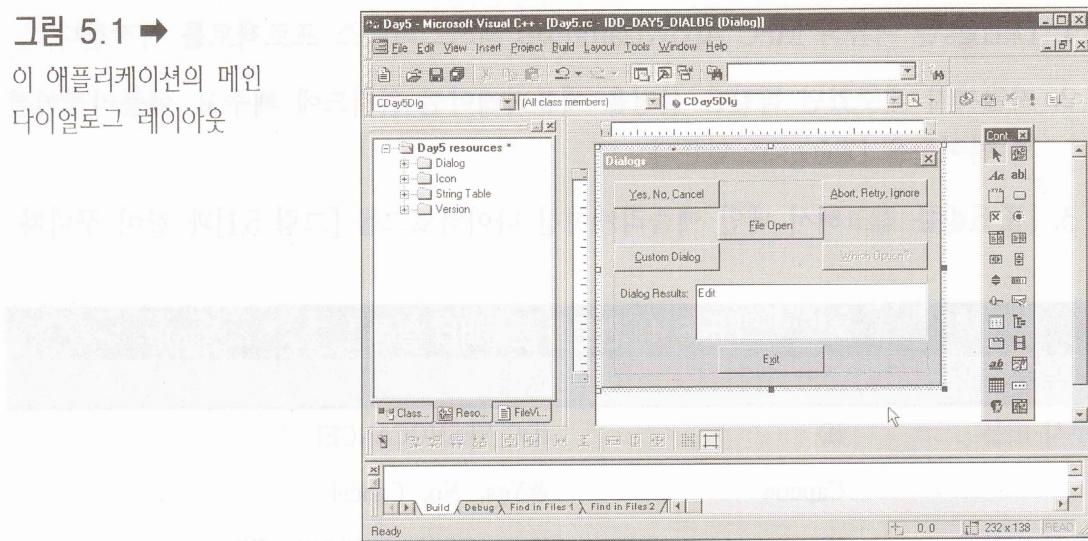
1. Dialogs란 이름의 MFC AppWizard(exe) 워크스페이스 프로젝트를 시작한다.
2. 지금까지 해주었던 똑같은 설정을 애플리케이션 위저드에 해주고, 애플리케이션의 타이틀을 Dialogs로 만든다.
3. [표 5.4]를 참고해서 메인 애플리케이션 다이얼로그를 [그림 5.1]과 같이 꾸미자.

[표 5.4] 컨트롤 프로퍼티 설정

컨트롤	프로퍼티	설정
푸시 버튼	ID	IDC_YESNOCANCEL
	Caption	&Yes, No, Cancel
푸시 버튼	ID	IDC_ABORTRETRYIGNORE
	Caption	&Abort, Retry, Ignore
푸시 버튼	ID	IDC_FILEOPEN
	Caption	&File Open
푸시 버튼	ID	IDC_BCUSTOMDIALOG
	Caption	&Custom Dialog
푸시 버튼	ID	IDC_BWHICHOPTION
	Caption	&Which Option?
	Disabled	체크
푸시 버튼	ID	IDC_EXIT
	Caption	E&xit
정적 텍스트	ID	IDC_STATIC
	Caption	Dialog Results:
에디트 박스	ID	IDC_RESULTS
	Multiline	체크
	Auto Vscroll	체크

4. 클래스위저드를 사용해서 [표 5.5]에 나온 대로 컨트롤에 변수를 물려두자.
5. 클래스위저드를 사용해서 Exit 버튼에 함수를 추가한 다음, 애플리케이션을 종료하는 코드를 넣는다(이미 알고 있을 것이다).

그림 5.1 →
이 애플리케이션의 메인
다이얼로그 레이아웃



(표 5.5) 컨트롤에 물려줄 변수

컨트롤	이름	카테고리	타입
IDC_RESULTS	m_sResults	Value	CString
IDC_BWHICHOPTION	m_cWhichOption	Control	CButton

메시지 박스 다이얼로그를 코딩하자

첫번째 푸시 버튼(Yes, No, Cancel 버튼)의 클릭 이벤트에 대한 함수를 클래스위저드를 사용해서 추가하고, [리스트 5.1]에 나온 코드를 입력하자.

리스트 5.1 OnYesnocancel() 함수

```

1: void CDialogsDlg::OnYesnocancel()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     // 새로 넣을 코드가 여기서부터 시작된다
6:     // ///////////////////////////////
7:     // ///////////////////////////////
8:
9:     int iResults; // 어떤 버튼이 눌렸나를 알아내기 위한 변수
10:
11:    // Ask the user 사용자에게 메시지 박스를 띄운다
12:    iResults = MessageBox("Press the Yes, No, or Cancel button",
13:                          "Yes, No, Cancel Dialog",
14:                          MB_YESNOCANCEL | MB_ICONINFORMATION);
15:
16:    // 어떤 버튼이 눌렸는지 판별한다
17:    // 사용자에게 어떤 버튼이 눌렸는지 알려준다

```

```

18:     switch (iResults)
19:     {
20:         case IDYES: // Yes 버튼이냐?
21:             m_sResults = "Yes! Yes! Yes!";
22:             break;
23:         case IDNO: // No 버튼이냐?
24:             m_sResults = "No, no, no, no, no.";
25:             break;
26:         case IDCANCEL: // Cancel 버튼이냐?
27:             m_sResults = "Sorry, canceled.";
28:             break;
29:     }
30:
31:     // Update the dialog
32:     UpdateData(FALSE);
33:
34:     ///////////////////////////////
35:     // 새로 넣을 코드는 여기서 끝난다
36:     ///////////////////////////////
37: }
```

이제 애플리케이션을 컴파일하고 실행하자. 메시지 박스 상의 버튼을 눌러보면, 다른 버튼에 대한 작업도 비슷하게 해주면 되겠다는 예상이 들 것이다. “Abort, Retry, Ignore” 버튼의 클릭 이벤트에 함수를 추가하고 [리스트 5.1]에 나온 똑같은 코드를 넣은 다음, MB_ABORTRETRYIGNORE와 MB_ICONQUESTION 아이콘 플래그로 대치하면 다른 버튼과 아이콘 조합이 메시지 박스에 나타나게 될 테니 말이다.

이들 컨트롤 이벤트의 함수 둘은 근본적으로는 똑같다. MessageBox() 함수의 반환 값을 담기 위한 정수 타입의 변수가 선언되었고, 표시될 메시지 텍스트와 메시지 박스의 타이틀 바에 놓일 텍스트, 그리고 버튼 조합 ID와 아이콘 ID의 조합 값을 인수로 한 MessageBox() 함수가 호출되었다.

MessageBox() 함수의 반환값을 담은 다음에는 이 값을 switch 문장에서 사용해서 ID마다 각기 다른 메시지를 사용자에게 보인다. if 문장을 한, 두 개 사용해서 동일한 작업을 수행할 수 있겠지만 switch를 사용하는 것이 조금 더 깔끔하다.

이제 애플리케이션을 컴파일하고 실행하면, [그림 5.2]와 같이 두 버튼 중 하나를 클릭해서 해당하는 메시지 박스를 보게 될 것이다. 메시지 박스가 떴을 때 메시지 박스에 있는 버튼 중 하나를 클릭하면, 어떤 버튼을 클릭했는지에 관한 정보가 [그림 5.3]과 같이 메인ダイ얼로그에 달린 에디트 박스에 나타나게 될 것이다.

그림 5.2 →

세 개의 선택 버튼을 가진
메시지 박스

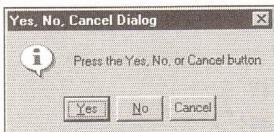
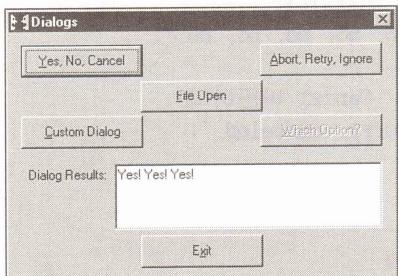


그림 5.3 →

어떤 버튼이 눌렸는가에 따라
표시되는 메시지가 다르다.



■ 공통ダイアログ 윈도우를 사용하자

공통ダイアログ 윈도우를 사용하는 일은 MessageBox() 함수를 사용하는 일만큼 누워 떡먹기는 아니지만, 쉬운 작업의 축에 속한다. 마이크로소프트 기반 클래스(MFC)에서 이미 공통 윈도우ダイアログ에 대한 클래스를 지원해 주고 있기 때문에 더 쉽다. 이들 클래스를 [표 5.6]에 정리해 보았다.

[표 5.6] 공통ダイア로그 담당 클래스

클래스	ダイ아로그의 타입
CFileDialog	파일 선택
CFontDialog	폰트 선택
CColorDialog	색깔 선택
CPageSetupDialog	인쇄 용지 셋업
CPrintDialog	인쇄
CFindReplaceDialog	찾기와 바꾸기

이들 클래스 안에 포장된 공통ダイアログ는 대부분의 윈도우 애플리케이션에서 파일을 열거나 저장할 때, 인쇄 옵션을 설정하거나 문서에서 찾아 바꾸기를 할 때 사용되는 표준ダイア로그이다. 이들 외에 OLE 공통ダイア로그란 것도 있어서, OLE나 ActiveX 컴포넌트와 애플리케이션에 공통적인 기능을 제공한다.

방금 설명한ダイアログ들은 생김새나 용도가 천차만별임에도 불구하고 모두 동일한 방법으로 사용할 수 있다. 순서 역시 몇 단계 되지도 않으며 간단하다.

1. 해당 클래스 타입의 변수를 선언한다.
2. 사용자에게 띄우기 전에 필요한 프로퍼티를 설정한다.
3. DoModal() 멤버 함수를 호출해서ダイアログ 윈도우를 띄운다.
4. DoModal() 함수의 반환값을 받아 내어서 사용자가 OK 버튼을 클릭했는지, Cancel 버튼을 클릭했는지를 결정한다.
5. 사용자가 OK 버튼을 클릭했으면 사용자가ダイアログ에 설정해 준 정보를 읽어낸다.

백견(見)이 불여 일행(行)인 법! 여러분의 애플리케이션에 CFileDialog 클래스를 추가하도록 하자. 클래스위저드를 사용해서 File Open 버튼의 클릭 메시지에 대한 함수를 추가하고, [리스트 5.2]에 나온 코드를 입력하자.

리스트 5.2 OnFileopen() 함수

```

1: void CDialogsDlg::OnFileopen()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     CFileDialog m_ldFile(TRUE);
10:
11:    // 파일 열기ダイアログ를 보이고 결과값을 받는다
12:    if (m_ldFile.DoModal() == IDOK)
13:    {
14:        // 선택된 파일 이름을 얻어낸다
15:        m_sResults = m_ldFile.GetFileName();
16:        //ダイアログ를 갱신한다
17:        UpdateData(FALSE);
18:    }
19:
20:    /////////////////
21:    // 새로 넣을 코드는 여기서 끝난다
22:    /////////////////
23: }
```

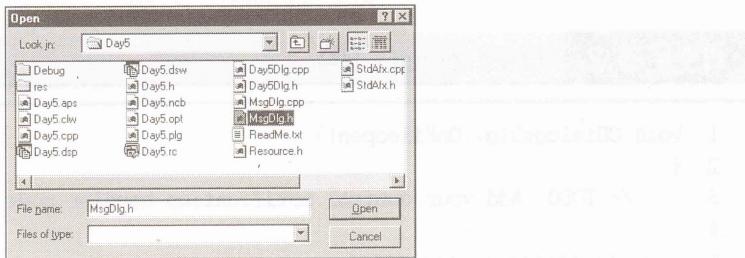
여기서 먼저 해준 일은 CFileDialog 클래스의 인스턴스를 선언한 것이다. 이 인스턴스를 만들 때, 클래스 생성자의 인수로 TRUE가 넘겨졌는데, 표시될ダイアログ 박스가 파일 열기(File Open)ダイア로그라는 것을 뜻한다. 만일 FALSE를 넘기면 파일

일 저장(File Save)ダイアルログが 나타난다. 사실 이 두ダイアル로그 간의 대단한 차이는 없으며 단지 타이틀 바만 다르다. 한편, 이 클래스의 생성자에는 파일 확장자, 시작할 파일과 위치, 표시될 파일들의 필터링 문자열 등의 인수를 추가로 더 넣어줄 수 있지만, 이들은 모두 디폴트 값을 가지고 있어서 현재로서는 그리 신경쓸 것이 아니다.

파일 열기ダイアル로그의 인스턴스를 만든 다음에는 DoModal() 함수를 호출한다. 이 함수는 원래 할아버지 격인 CDialog 클래스의 멤버 함수이기 때문에 모든ダイ얼로그 윈도우에서 사용이 가능하다. 아무튼 DoModal() 함수를 사용하면 [그림 5.4]와 같이 파일 열기(File Open)ダイ얼로그 윈도우가 나타난다. 이 함수의 반환값은 사용자가 열기(Open) 버튼이나 취소(Cancel) 버튼을 눌러서ダイ얼로그를 닫았을 때 얻어낼 수 있는데, 열기 버튼을 클릭했을 경우에는 IDOK가 반환되며, 이때 “사용자가 파일을 열기 원하는 구나”라고 인식하고 필요한 동작을 코딩해 주면 된다.

그림 5.4 →

파일 열기(File Open)
ダイ얼로그



Note

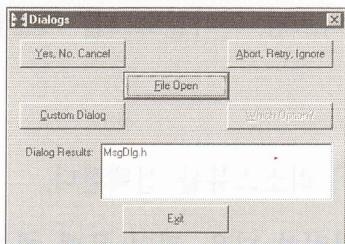
ダイ얼로그 윈도우가 사용자에게 보여지는 형태는 두 가지가 있다. 첫번째는 모달(Modal) 윈도우 형식이라고 하는 것으로, 표시되는 도중에는 이외의 모든 윈도우를 사용할 수 없다. 즉, 이ダイ얼로그가 닫히기 전에는 절대로 다른 윈도우를 사용할 수 없다. 좋은 예가 지금껏 사용해 왔던 메시지 박스로서, 버튼을 눌러서 닫기 전에는 애플리케이션을 계속 진행할 수 없었다.

두번째는 모들리스(Modeless) 윈도우라고 불리는 것으로, 표시되는 도중에도 계속 다른 윈도우 혹은 애플리케이션을 사용할 수 있으며, 사용자가ダイ얼로그 이외의 어떤 곳에서 무엇을 하든지 전혀 방해하지 않는다. 모들리스ダイ얼로그 윈도우의 좋은 예는 마이크로소프트 워드의 찾기(Find)와 찾아 바꾸기(Find and Replace)ダイ얼로그로서, 화면 상에 계속 떠있는 상태에서 단어의 찾기와 바꾸기를 계속 진행해 갈 수 있다.

자, 이젠 선택된 파일의 이름을 읽어서 표시해야 한다. 이때 사용하는 함수가 CFileDialog 클래스의 GetFileName() 함수이며, 이 함수의 반환값을 m_sResults에 대입한다. GetFileName()은 드라이브 이름이나 디렉토리 경로가 다 빠진 파일 이름만을 얻어낼 때 사용하는 함수([그림 5.5] 참조)인데, 디렉토리 경로(GetPathName())나 파일 확장자(GetFileExt())를 얻어내는 함수들도 모두 준비되어 있으니 걱정 말기 바란다.

그림 5.5 →

선택된 파일 이름을 표시한다.



■ 다이얼로그 원도우를 직접 만들어 사용하자

이제 표준 내장 다이얼로그를 사용하는 방법은 상당 수준 이해했을 것이다. 그러면, 여러분만의 커스텀 다이얼로그를 만들려면 어떻게 해야 할까? 질문은 어려운 듯 했지만 실은 매우 간단하다. 왜냐하면, 메인 다이얼로그 원도우를 만들기 위해 지금껏 실습해온 과정과 공통 다이얼로그에서 사용한 방법을 합쳐서 사용하면 그만이기 때문이다. 몇 단계가 더 추가되긴 하겠지만, 모두 이해될만한 것이니까 편안하게 읽어 가기만 하자.

■ 다이얼로그 원도우를 생성하자

여러분의 애플리케이션에서 사용하기 위한 커스텀 다이얼로그에는 사용자가 텍스트를 입력할 수 있는 에디트 박스 하나와 여러 옵션 중 하나를 선택할 수 있는 라디오 버튼 그룹 하나를 붙일 것이다. 사용자가 OK 버튼을 클릭하면 이 다이얼로그의 에디트 박스에 입력된 텍스트를 메인 애플리케이션의 다이얼로그 원도우의 표시 영역에다가 나타내도록 하겠다. 정리해 보면, 지금 실습해 볼 것은 여러분이 직접 만든 다이얼로그를 띄워서 사용자로부터 정보를 입력받은 다음, 이 다이얼로그 원도우가 닫힌 후 해당 정보를 읽어서 이용하는 절차이다.

일단 커스텀 다이얼로그를 만들어야 한다. 필요한 것은 다음과 같다.

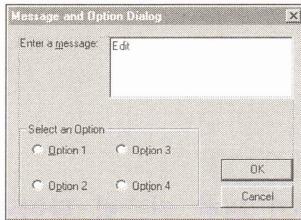
- ◆ 애플리케이션 리소스에 다이얼로그를 하나 더 추가하는 일
- ◆ 다이얼로그 원도우의 레이아웃을 꾸미는 일
- ◆ 방금 만든 다이얼로그 원도우를 담당하는 클래스가 상속받을 기본 클래스를 정하는 일
- ◆ 다이얼로그에 컨트롤을 붙이는 일

앞의 작업이 끝나면 여러분의 커스텀ダイ얼로그를 애플리케이션에 바로 사용할 수 있게 된다. 자, 그럼 시작하자.

1. 프로젝트 워크스페이스에서 리소스뷰를 선택한다.
2. Dialog 폴더에서 오른쪽 클릭한 다음, 팝업 메뉴에서 Insert Dialog를 선택한다.
3. 리소스뷰의 트리 구조에서 새로 생긴ダイ얼로그에서 오른쪽 클릭한 다음, 팝업 메뉴에서 Properties를 선택한다.
4. ID에 IDD_MESSAGEDLG라고 입력한다.
5. 새ダイ얼로그 윈도우의 OK와 Cancel 버튼을 지우지 말고 [그림 5.6]에 나온 위치로 옮긴다.

그림 5.6 →

커스텀ダイ얼로그 윈도우의
레이아웃



6. [표 5.7]을 참고해서ダイ얼로그 윈도우에 컨트롤을 놓고 프로퍼티를 설정하자.

[표 5.7] 커스텀ダイ얼로그의 컨트롤 프로퍼티 설정

컨트롤	프로퍼티	설정
정적 텍스트	ID	IDC_STATIC
	Caption	Enter a &message:
에디트 박스	ID	<u>IDC_MESSAGE</u>
	Multiline	체크
그룹 박스	Auto Vscroll	체크
	ID	STATIC
라디오 버튼	Caption	Select an Option
	ID	IDC_OPTION1
	Caption	&Option 1
	Group	<u>체크</u>

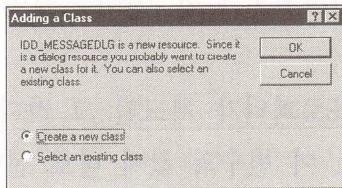
컨트롤	프로퍼티	설정
라디오 버튼	ID	IDC_OPTION2
	Caption	Option 2
라디오 버튼	ID	IDC_OPTION3
	Caption	Option 3
라디오 버튼	ID	IDC_OPTION4
	Caption	Option 4

7. 다이얼로그 꾸미기가 끝났으면 클래스워저드를 시작하자. [그림 5.7]과 같은 다이얼로그 박스가 나타날 것이다.

그림 5.7 →

Adding a Class

다이얼로그

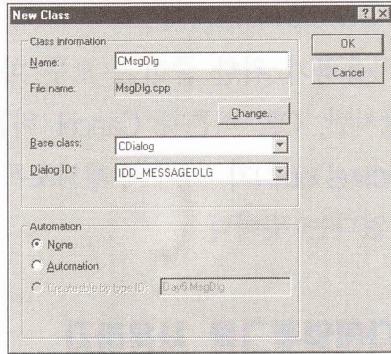


8. 디폴트로 선택되어 있는 Create a New Class로 놔둔 상태에서 OK를 클릭한다. 다른 다이얼로그 박스가 또 나타나면서 새 클래스의 이름과 이 클래스가 상속 할 기본 클래스를 정해주라고 안달할 것이다.

9. 클래스 이름으로 CMsgDlg를 Name 필드에 입력하고, Base Class에서 CDialog를 선택한다([그림 5.8] 참조).

그림 5.8 →

New Class 다이얼로그



10. 나머지는 모두 디폴트로 두고 OK를 클릭한다.

11. 클래스워저드가 열린 상태에서, [표 5.8]을 참고해서 새 다이얼로그 윈도우의 각 컨트롤에 대해 변수를 물려 두도록 한다.

(표 5.8) 컨트롤에 물려줄 변수들

컨트롤	이름	카테고리	타입
IDC_MESSAGE	m_sMessage	Value	CString
IDC_OPTION1	m_iOption	Value	int

커스텀 다이얼로그의 컨트롤 프로퍼티와 변수를 설정할 때 두 가지를 주의하도록 하자. 우선 라디오 버튼 그룹에서 첫번째 버튼에만 Group 프로퍼티를 설정해야 한다는 것이다. 이렇게 해야 첫번째 버튼을 포함한 그 다음의 버튼들이 같은 그룹에 속하게 된다. 만약에 모든 라디오 버튼에 Group 프로퍼티를 설정하면, 각 버튼은 독립된 그룹에 따로 따로 속하게 되는 엄한 상황이 생긴다. 즉, 모든 버튼을 동시에 선택할 수 있다는 뜻이다. 뭐 이렇게 되면 모든 버튼이 체크 박스처럼 동작하게 될 것 같지만, 사용자는 체크 해제에 심각한 어려움을 느낄 것이다. 모양새도 모양새이고 말이다.

주의해야 할 두번째는 Group 프로퍼티가 체크된 그 라디오 버튼 하나에만 정수 변수를 물려두어야 한다는 점이다. 이 변수의 값이 바로 선택된 라디오 버튼을 가리키는 것인데, 첫번째 라디오 버튼이 선택되면 0의 값을 가지며, 두번째 라디오 버튼이 선택되면 1의 값을 가지는 식으로 진행된다. 어떤 라디오 버튼을 자동으로 선택하고 싶으면 이 변수의 값을 범위 내에서 임의로 설정한 다음, UpdateData(FALSE)를 실행하면 될 것이다.

Note

우리는 C++ 프로그래밍 언어를 사용하고 있기 때문에, 모든 숫자는 1이 아니라 0부터 시작한다는 점도 아울러 잊지 말도록 하자.

이제 두번째 다이얼로그를 사용하기 위한 준비는 모두 마친 셈이다. UpdateData()를 한, 두번 호출해야 할 것 같지만, OK 버튼과 Cancel 버튼을 없애지 않았기 때문에 OK 버튼을 클릭했을 때 UpdateData()가 이미 수행되도록 되어 있다. 결과적으로 아무 코드도 연결해줄 필요가 없다는 뜻이다.

■ 메인 애플리케이션에서 다이얼로그를 사용하자

커스텀 다이얼로그가 이미 만들어진 상태에서는 더 이상 기칠 것이 없다. 공통 다이얼로그를 사용하듯이 하면 되는 것이다. 일단 커스텀 다이얼로그 클래스의 인스턴스를 선언하면 클래스 생성자가 자동으로 호출되고, 클래스의 인스턴스가 생성된다. 다

음, 이 다이얼로그 클래스의 DoModal() 함수를 호출하고 반환값을 받아낸다. 마지막으로 이 다이얼로그에 놓인 컨트롤에 물려둔 변수의 값을 읽는다.

다이얼로그 인스턴스를 생성하자

여러분의 애플리케이션에서 커스텀 다이얼로그를 사용하기 전에, 메인 다이얼로그 위도우에게 커스텀 다이얼로그의 존재를 알려줄 필요가 있다. 즉, 커스텀 다이얼로그에 관한 헤더 파일을 메인 애플리케이션 다이얼로그에 대한 소스 파일에서 인클루드 해주면 된다. 다음의 단계를 따르자.

1. 프로젝트 워크스페이스의 파일뷰 탭을 선택한다.
2. Dialog Files를 펼치고 Source Files 폴더를 펼친다.
3. DialogsDlg.cpp 파일을 더블클릭한다. 디벨로퍼 스튜디오의 에디터 영역에 메인 애플리케이션 다이얼로그에 대한 소스 파일이 열린다.
4. 소스 코드 파일의 위쪽으로 이동해서 #include 문장이 있는 부분을 찾은 후, [리스트 5.3]과 같이 DialogsDlg.h 파일의 앞에서 MsgDlg.h 파일을 인클루드하자.

리스트 5.3 헤더 파일의 인클루드 부분

```

1: // DialogsDlg.cpp : implementation file
2: //
3:
4: #include "Stdafx.h"
5: #include "Dialogs.h"
6: #include "MsgDlg.h"
7: #include "DialogsDlg.h"
8:
9: #ifdef _DEBUG
10: #define new DEBUG_NEW
11: #undef THIS_FILE
12: static char THIS_FILE[] = __FILE__;
13: #endif
14:
15: /////////////////////////////////////////////////
16: // CAboutDlg dialog used for App About

```

반드시 DialogsDlg.h 파일이 인클루드된 앞에 MsgDlg.h 파일을 인클루드하기 바란다. 왜냐하면, 메인 다이얼로그의 헤더 파일 안의 메인 다이얼로그 클래스에 커스텀 다이얼로그의 변수 선언을 해둘 것이기 때문이다. 만일 인클루드 순서가 제대로 되지 않으면 컴파일러가 클래스 선언을 못찾겠다고 계속 야단법석을 피울 테니까 말이다.


Note

#include 문장은 C와 C++ 프로그래밍 언어에서 사용하는 컴파일러 지시자로서, 컴파일러에게 어떤 파일을 읽어서 컴파일될 소스 코드 내의 이름을 참조하라고 말해주는 기능을 가진다. 인클루드 문장은 헤더 파일 내의 정보를 알아야 하는 소스 코드에 포함될 수 있는 각각의 파일에 클래스, 구조체, 함수 선언들을 분리해서 저장하고 관리할 때 사용한다. #include 문장이 어떻게 작동되는지에 관한 추가 정보가 필요하면 부록 A, “C++도 다시 보자”를 참고하기 바란다.

별로 해줄 일도 없는데 말만 길어진 것 같다. 아무튼 메인 다이얼로그에게 커스텀 다이얼로그를 알려주었으면 다시 다음 단계를 따르자.

1. 프로젝트 워크스페이스에서 클래스뷰 탭을 선택한다.
2. CDIALOGSDLG 클래스에서 오른쪽 클릭하여 팝업 메뉴를 띄운다.
3. 메뉴에서 Add Member Variable를 선택한다.
4. Variable Type에는 CMsgDlg, Variable Name에는 m_dMsgDlg를 입력하고, 액세스 지정자로는 Private를 선택한다. OK를 클릭해서 메인 다이얼로그에 변수 추가를 끝낸다.

이제 클래스뷰의 트리 구조에서 CDIALOGSDLG 클래스를 펼치면, 메인 애플리케이션 다이얼로그 클래스인 CDIALOGSDLG의 아래에 방금 추가한 커스텀 다이얼로그 클래스 타입의 변수가 멤버로 들어가 있음을 알 수 있다. 즉, 여러분의 애플리케이션에서 이 다이얼로그를 사용할 수 있게 되었다는 뜻이다.

다이얼로그를 띄우고 변수를 읽어내자

여러분은 이제 메인 애플리케이션 다이얼로그에 커스텀 다이얼로그를 멤버 변수로 추가한 상태이며, 다이얼로그를 띄울 코드만 만들면 된다. 조금밖에 안남았다.

1. 클래스위저드를 열고 IDC_BCUSTOMDIALOG 버튼의 클릭 이벤트 메시지에 대한 함수를 추가한다.
2. IDC_BWHICHOPTION 버튼의 클릭 이벤트 메시지에 대한 함수를 추가한다.
3. [리스트 5.4]와 같이 OnBcustomdialog() 함수를 작성하자.

리스트 5.4 OnBcustomdialog 함수

```

1: void CDIALOGS::OnBcustomdialog()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 메시지ダイアログ를 띄우고 결과값을 얻어낸다
10:    if (m_dMsgDlg.DoModal () == IDOK)
11:    {
12:        // OK를 클릭했으면 적절한 메시지를
13:        // 메시지ダイアログ로 띄운다.
14:        m_sResults = m_dMsgDlg.m_sMessage;
15:        //ダイアログ를 갱신한다
16:        UpdateData(FALSE);
17:        // 버튼을 활성화시킨다
18:        m_cWhichOption.EnableWindow(TRUE);
19:    }
20:
21:    /////////////////
22:    // 새로 넣을 코드는 여기서 끝난다
23:    /////////////////
24: }
```

4. [리스트 5.5]와 같이 OnBwhichoption() 함수를 작성하자.

리스트 5.5 OnBwhichoption() 함수

```

1: void CDIALOGS::OnBwhichoption()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 어떤 라디오 버튼이 선택되었는지 결정하고,
10:    // 적절한 메시지를 나타낸다
11:    switch(m_dMsgDlg.m_iOption)
12:    {
13:        case 0: // 첫번째 라디오 버튼이오?
14:            m_sResults = "The first option was selected.";
15:            break;
16:        case 1: // 두번째 라디오 버튼이오?
17:            m_sResults = "The second option was selected.";
18:            break;
19:    }
20:
```

```

19:     case 2: // 세번째 라디오 버튼이오?
20:         m_sResults = "The third option was selected.";
21:         break;
22:     case 3: // 네번째 라디오 버튼이오?
23:         m_sResults = "The fourth option was selected.";
24:         break;
25:     default: // 다섯번째 라디오 버튼이오?
26:         m_sResults = "To option was selected.";
27:         break;
28:     }
29:
30:     //ダイ얼로그를 갱신한다
31:     UpdateData(FALSE);
32:
33: ///////////////////////////////////////////////////////////////////
34: // 새로 넣을 코드는 여기서 끝난다
35: ///////////////////////////////////////////////////////////////////
36:

```

커스텀ダイ얼로그의 DoModal() 함수를 호출하여 사용자에게ダイ얼로그박스를 보였으며, [그림 5.9]와 같이 사용자가ダイ얼로그상의 두 버튼 중 하나를 누를 때까지 애플리케이션이 대기하고 있게 된다. 만일 OK 버튼을 클릭하면, 사용자가 에디트 박스에 입력한 텍스트가 에디트 박스에 물려둔 변수로 복사되어 메인 애플리케이션의 윈도우에 나타나게 된다. 메인ダイ얼로그의 디스플레이를 갱신한 후에는 [그림 5.10]과 같이 Which Option 버튼을 활성화시킨다. 만일 Cancel 버튼을 클릭했다면 아무 신경도 쓰지 않는다.

그림 5.9 →

사용자가 메시지를
입력할 수 있다.

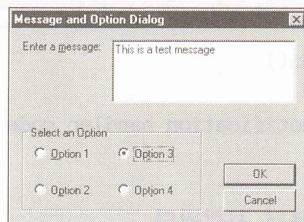
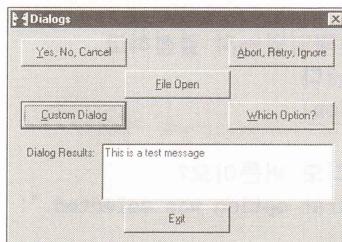


그림 5.10 →

커스텀ダイ얼로그에
입력한 메시지가
사용자에게 보여진다.

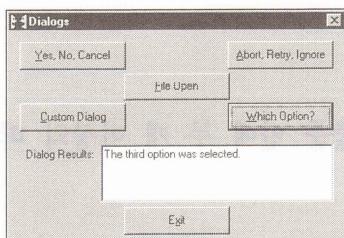


사용자가 Which Option 버튼을 클릭하면, 커스텀ダイ얼로그상의 라디오 버튼에 물려둔 변수가 switch 문장으로 넘어가고, [그림 5.11]과 같이 어떤 라디오 버튼이 선택

되었는지에 관한 메시지가 출력될 것이다. 메인 다이얼로그에서도 커스텀 다이얼로 그상의 컨트롤에 물려둔 변수를 얼마든지 액세스할 수 있다는 점을 꼭 알아두기 바란다. 왜냐하면, 클래스위저드가 컨트롤에 물려진 변수는 자동적으로 `public` 멤버로 만들어 놓기 때문이다. 굳이 다른 클래스에서 이 변수를 보지 못하게 하려면 `public:` 액세스 지정자를 `private:`으로 만들어 두어도 상관없지만 여기서는 그렇게 하지 않는다. 또한, 이들 변수가 선언되어 있는 `//{{AFX_DATA` 라인 뒤에는 되도록 어느 것도 두지 않는 것이 좋다. 이 부분은 MFC의 클래스위저드가 사용하는 매크로이며, 애플리케이션을 컴파일할 때 디벨로퍼 스튜디오에 속해 있는 위저드들이 필요할 때마다 비주얼 C++ 컴파일러를 방해하지 않은 채 변수를 위치시키고 조작할 수 있게 해주기 때문이다.

그림 5.11 →

사용자가 선택한 버튼에 대해 메시지가 출력된다.



요약

이번 장에서는 애플리케이션에 추가적인 다이얼로그 윈도우를 사용해서 더욱 사용자와 친숙한 환경을 만드는 방법에 대해 공부하였다. 일단 간단한 `MessageBox()` 함수에 사용할 수 있는 옵션에 대해 알아봄과 동시에 몇 가지의 버튼 조합을 메시지 박스에 붙여 본 후, 사용자가 클릭한 버튼의 ID도 알아내는 방법을 익혔다. 이 정보에 따라 어떤 처리를 해줄 것인가는 여러분에게 달린 문제이다.

이것 뿐만이 아니다. 윈도우 운영체제에 내장된 공통 다이얼로그와 MFC 클래스 라이브러리에서 어떻게 C++ 클래스로 포장되었는지에 대해 알아본 다음, 표준적인 파일 선택 기능을 위한 파일 열기(File Open) 다이얼로그를 띄우는 코드도 직접 작성하였다.

마지막으로, 사용자의 입력을 받아내기 위한 여러분만의 다이얼로그 윈도우를 직접 만들어서 화면에 띄우고, 메인 애플리케이션에서 적절한 정보를 읽어내는 방법도 직접 실습하면서 확인하는 것으로 이 장의 공부를 마쳤다.

Q&A

① 커스텀ダイアル로그에는 코드가 하나도 없네요. 만약에 제ダイ얼로그를 하나 더 만든다면 또 이렇게 해야 하나요? 아니면 코드를 넣어도 되나요?

② 커스텀ダイアル로그 윈도우는 지금까지 여러분이 만들어온 애플리케이션에서 사용했던 메인ダイ얼로그 윈도우와 하등 다른 것이 없습니다. 상호대화적인 조작을 위해 다른柴油로그 윈도우를 조절할 필요가 있다면 필요한 코드를 넣어도 되겠지요. 이번 장에서 같이 만들어본 커스텀ダイ얼로그 윈도우의 경우에는 코드를 굳이 추가할 필요가 없기 때문에 코드를 넣어주지 않은 것입니다. OK 버튼을 누를 때 UpdateData()를 호출해서 컨트롤의 내용을 읽어내는 일만 하면 되는데, 이것은 CDlg 클래스의 OnOK() 함수에 이미 구현되어 있을 것입니다. IDOK를 ID로 가진 버튼을 지우지도 않았구요. 그렇지요?

③ 똑같은 MessageBox 함수 호출문 안에 두 개 이상의 버튼 조합 ID를 쓰면 어떻게 될까요?

④ 무슨 큰일이라도 기대하십니까? 아무 일도 일어나지 않습니다. 애플리케이션은 정상적으로 컴파일되지만, MessageBox()가 호출될 때 메시지 박스가 열리지 않습니다. 즉, 죄없는 사용자들은 아무 메시지도 받지 못한다는 뜻이죠.

⑤ 파일 열기ダイ얼로그가 제가 설정해 준 특정한 디렉토리를 열어주도록 할 수는 없나요?

⑥ CFileDialog 클래스에는 m_ofn이라 불리는 public 멤버 변수가 있습니다. 이 변수는 실제로는 파일 열기ダイ얼로그의 모든 속성을 가지고 있는 구조체이죠. 이 속성에는 초기 디렉토리도 속해 있습니다. 이것을 알고 있다면, [리스트 5.6]을 보도록 합시다.

리스트 5.6 OPENFILENAME 구조체

```

1: typedef struct tagOFN { // ofn
2:     DWORD      lStructSize;
3:     HWND       hwndOwner;
4:     HINSTANCE  hInstance;
5:     LPCTSTR    lpstrFilter;
6:     LPTSTR     lpstrCustomFilter;
7:     DWORD      nMaxCustFilter;
8:     DWORD      nFilterIndex;
9:     LPTSTR    lpstrFile;

```

```

10:     DWORD      nMaxFile;
11:     LPTSTR     lpstrFileTitle;
12:     DWORD      nMaxFileTitle;
13:     LPCTSTR    lpstrInitialDir;
14:     LPCTSTR    lpstrTitle;
15:     DWORD      Flags;
16:     WORD       nFileOffset;
17:     WORD       nFileExtension;
18:     LPCTSTR    lpstrDefExt;
19:     DWORD      lCustData;
20:     LPOFNHOOKPROC lpfnHook;
21:     LPCTSTR    lpTemplateName;
22: } OPENFILENAME;

```

파일 열기ダイアログ의 세세한 성질을 조절해 주기 위해서는 DoModal() 함수를 호출하기 전에 m_ofn 구조체의 멤버를 수정해 주면 됩니다. 예를 들어, C:\Temp를 시작 디렉토리로 정해주려면 [리스트 5.7]처럼 해줍시다.

리스트 5.7 고쳐진 OnFileopen () 함수

```

1: void CDialogsDlg::OnFileopen()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     CFileDialog m_ldFile(TRUE);
10:
11:    // 시작 디렉토리를 초기화한다
12:    m_ldFile.m_ofn.lpstrInitialDir = "C:\\\\Temp\\\\";
13:
14:    // 파일 열기ダイアログ를 보이고 결과값을 받아낸다
15:    if (m_ldFile.DoModal() == IDOK)
16:    {
17:        // 선택된 파일의 이름을 얻는다
18:        m_sResults = m_ldFile.GetFileName();
19:        //ダイアログ를 갱신한다
20:        UpdateData(FALSE);
21:    }
22:
23:    /////////////////
24:    // 새로 넣을 코드는 여기서 끝난다
25:    /////////////////
26: }

```

실습해 보기

“실습해 보기” 절에서는 배운 것을 확인하는 퀴즈와 이를 활용해서 응용력을 높이기 위한 연습문제를 풀어볼 기회를 가질 수 있게 될 것이다. 퀴즈와 연습문제의 답은 부록 B, “퀴즈 및 연습문제 해답”에 있지만.. 인생은 사필귀정이다.

■ 퀴즈

1. MessageBox() 함수에 MB_RETRYCANCEL 버튼 조합 ID를 같이 써서 호출하면 어떤 반환 코드를 얻어낼 수 있겠는가?
2. MFC 클래스에서 정의되어 있는 윈도우 운영체제에 내장된 공통 다이얼로그에는 무엇이 있는가?
3. 모달 다이얼로그와 모듈리스 다이얼로그의 차이는?
4. 여러분의 애플리케이션에서 파일 열기(File Open) 다이얼로그 대신에 파일 저장(File Save) 다이얼로그를 나타내려면 어떻게 할 수 있을까?
5. 이번 장에서 만들어 본 커스텀 다이얼로그에는 왜 코드를 넣어줄 필요가 없었는가?

■ 연습문제

1. 파일 열기 다이얼로그를 사용할 때 디렉토리도 포함할 수 있도록 애플리케이션을 고쳐보자(힌트 : GetPathName() 함수는 파일 열기 다이얼로그에서 선택된 경로와 파일 이름을 모두 반환한다).
2. 커스텀 다이얼로그에 버튼을 추가해서 예(Yes)와 아니오(No) 버튼을 단 메시지 박스를 띄울 수 있도록 해보자. MessageBox()의 결과값은 메인 애플리케이션 다이얼로그로 되돌리게 하자.