## >> 저자 머리말 MFC 학습의 좋은 벗이자 안내자가 되기를..

시중에 판매되는 MFC 학습용 서적은 크게 두 부류로 나눌 수 있다. 하나는 바이블 형태로 방대한 내용과 예제를 제공하며, 다른 하나는 따라하기 형태로 실습 화면을 모두 보여줌으로써 학습자가 그대로 따라하며 배우도록 한다. 전자는 지나치게 방대한 내용으로 인해 처음 배우는 이들에게는 적합하지 않고, 후자는 예제를 중심으로 진행하기 때문에 정작 중요한 개념과 구조적인 이해를 소홀히 하는 경향이 있다. 이 책은 강의 교재를 염두에 두되 두 부류 서적의 장점을 종합하여 이론과 실습이 균형을 이룰 수 있도록 구성했다. 이론적인 설명은 군더더기 없이 명확한 용어와 정의를 사용했고, 이론에서 다룬 내용은 곧바로 실습하면서 이해할 수 있도록 구성했다. 바이블 서적에 비해 다루는 주제의 폭은 좁지만, 각각의 주제를 충실하게 다루므로 강의 교재는 물론이고 처음 MFC를 접하는 개인 학습자에게도 좋은 벗이 될 것이다.

강사가 100을 설명해도 듣는 사람은 보통 20~30 밖에 알지 못하고, 집중을 잘 하는 사람도 50 이상을 알기는 힘들다고 한다. 강의할 때 항상 쉽게 설명하려고 노력해도 한 번에 전달되는 분량에는 한계가 있다. 책도 마찬가지로, 한 번 보는 것으로 저자가 알려주는 내용을 모두 얻을 수는 없다. 하지만 책은 강의와 달리 시간이 지나도 사라지지 않는다. 부디 한 번에 그치지 않고, 시간을 두고 여러 번 학습하길 바란다. 또한 실습도 충실히 하여 책에서 전달하는 100보다 더 많은 것을 얻어 나중에 좋은 개발자로 만날 수 있기를 기원한다.

### [2010 버전을 준비하며]

기존 윈도우 프로그래밍(개정판)과 비교하여 추가하거나 삭제한 장은 없지만, 세부적으로 달라진 내용은 다음과 같다.

1. **프로그램 버전업 및 버전별 소스 코드 제공** : 비주얼 C++ 2010을 기준으로 모든 실습 과정을 재작성하였다. 본문 예제와 실전 프로젝트 코드는 비주얼 C++ 2008/2010/2012/2013으로 각각 컴파일할 수 있도록 별도의 프로젝트로 제공된다.

2. **유니코드 문자 집합 지원** : 기존 개정판과 달리 유니코드 문자 집합을 지원하도록 모든 예제를 수정하였다.

3. **본문 내용 수정 및 보완** : 모든 장에 걸쳐 기존 내용을 세밀하게 검토하고 수정/보완하였고, 특히 14장(데이터베이스)은 현재 기술 동향에 맞게 후반부를 새롭게 작성하였다.

4. **'현장 팁'과 '현장의 목소리' 보완** : 기존의 내용을 수정/보완하고 본문 흐름에 맞게 재배치하였다. 또한 새로운 주제를 담은 글도 추가하여 현업 개발자의 생생한 노하우를 전달하는 데 노력을 기울였다.

# >> 워밍업

## 이 책을 읽기 전에

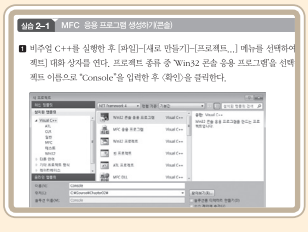
### 누구를 위한 책인가

주 대상은 C++ 언어를 습득한 후, 처음으로 윈도우 프로그래밍을 시작하는 소프트웨어 관련학과 학부생이다. 바이블 서적처럼 광범위한 내용을 다루지는 않지만 중요 주제를 깊이 있게 다루기 때문에 강의 교재로 사용하기 적합할 뿐만 아니라 일반인이나 비전공자에게도 충분히 도움이 될 것이다.

### 선수 연계 과목

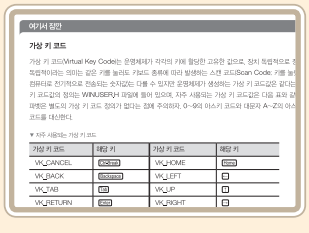
C++ 언어를 습득한 후, 처음으로 윈도우 프로그래밍을 시작하는 독자를 위한 책이므로 C++ 언어 관련 지식이 필요하다. 본문에서는 따로 C++ 언어를 복습하지 않으므로 부족한 지식이 있다면 관련 서적을 참조하기 바란다. 윈도우 프로그래밍 경험은 필요하지 않지만 윈도우 운영체제에 대해서는 익숙해야 원활하게 학습할 수 있다. 윈도우 API에 대한 어느 정도의 지식이 있다면 MFC를 좀더 쉽게 배울 수 있으므로 시간을 내서 공부해 두는 것도 좋다.

## 이 책의 구성 요소



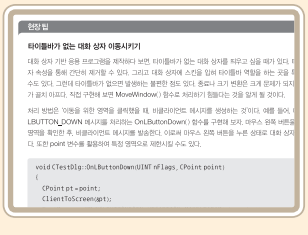

**실습**

응용 프로그램 제작 과정을 단계별로 정리했다.
개념 설명 이후에 바로 이어지기 때문에 이해력을 높일 수 있고,
그림과 코드를 하나하나 보여주기 때문에 쉽게 따라할 수 있다.




**여기서 잠깐**

본문에 대한 보충 설명, 참고 사항을
본문의 흐름을 해치지 않도록 본문과 분리해 정리했다.
본문 학습 중에 참고하면 도움이 될 것이다.




**현장 팁**

저자가 현장에서 생각하고 발견하는 유용한 팁,
함정을 미리 피해갈 수 있는 주의 사항,
참고로 알아두어야 할 사항을 별도로 정리했다.

## 이 책의 뼈대만 빨리 보기

이 책은 MFC 준비, 기본, 심화, 실전 프로젝트 형태로 구성된다. 1부 준비 학습과 2부 기본 학습에서는 MFC 프로그래밍의 기본을 다지고, 3부 심화 학습을 통해 고급 MFC 프로그래밍 기술을 익힐 수 있다. 4부 실전 프로젝트에서는 좀더 큰 규모의 소프트웨어 제작을 통해 1~3부에 걸쳐 배운 내용을 복습하면서 실무에 필요한 응용력을 키울 수 있다.

### ❶ 1부. 준비 학습(1~3장)

MFC 기초를 다지기 위한 기본적이고 필수적인 내용을 다룬다. 1장에서는 윈도우 응용 프로그램의 구조와 동작 원리를 이해하기 위해 SDK 프로그램과 MFC 프로그램을 비교 분석한다. 2장에서는 주요 데이터 타입과 유틸리티 클래스를 연습한다. 3장에서는 MFC의 주요 특징과 서비스를 살펴보고, 자동으로 생성한 MFC 프로그램을 분석한다.

### ❷ 2부. 기본 학습(4~10장)

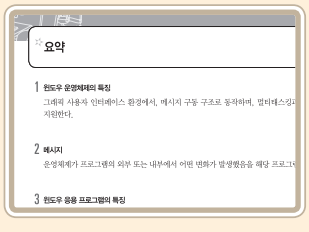
화면 출력부터 도큐먼트/뷰 구조에 이르기까지 MFC 프로그램의 핵심 요소를 주제별로 다룬다. 이론은 충실하게 설명하되, 실습은 이론을 쉽게 이해할 수 있도록 간단하지만 실용적인 예제로 구성하여 이론과 실습의 균형을 맞추었다.

### ❸ 3부. 심화 학습(11~15장)

기본 학습에서 다루지 않은 부분 중 실전에서 자주 사용되는 핵심 주제를 중심으로 구성하였다. 11~12장은 사용자 인터페이스, 13장은 다중 처리, 14장은 데이터 입출력, 15장은 원격 통신과 관련된 주제를 다룬다. 순서와 관계 없이 진행할 수 있으므로 당장 필요한 부분을 찾아 학습하면 된다.

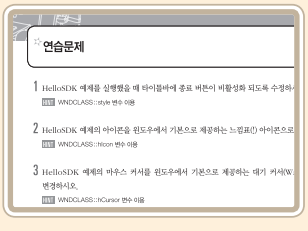
### ❹ 4부. 실전 프로젝트(16장)

좀더 큰 규모의 응용 프로그램을 개발하는 과정을 통해 본문에서 배운 다양한 기법을 총정리할 수 있다. 본문에서 미처 다루지 못한 부분도 실습을 통해 새롭게 배울 수 있으며, 실전 응용력을 키울 수 있도록 구성했다.



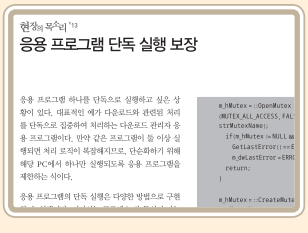

**요약**

해당 장이 끝날 때마다 핵심적인 내용을 요약해서 정리한다. 본문에서 익힌 세분화된 지식을 여기서 전체적으로 조립하여 완성된 모습을 볼 수 있다.




**연습문제**

본문에서 익힌 핵심 내용을 확인하고 응용력을 기를 수 있다.




**현장의 목소리**

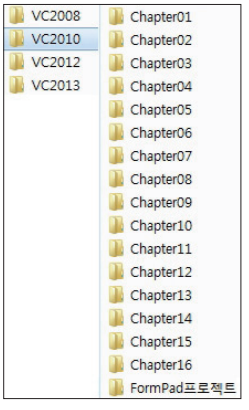
현장에서 유용하게 적용할 수 있는 테크닉과 팁 등을 풍부한 강의 경험을 가진 저자와 현업에 종사하는 현장 전문가의 입을 통해 직접 들을 수 있다.

## 예제 소스 구성

예제 소스는 다음과 같이 구성되어 있다.

사용하는 프로그램에 맞게 실습해볼 수 있도록 버전별로 나누어 정리했다. 각 폴더에는 본문 예제 소스와 실행 파일, FormPad 프로젝트가 장별로 정리되어 있다.

- VC2008\Chapter# : 비주얼 C++ 2008 버전
- VC2010\Chapter# : 비주얼 C++ 2010 버전
- VC2012\Chapter# : 비주얼 C++ 2012 버전
- VC2013\Chapter# : 비주얼 C++ 2013 버전
- FormPad 프로젝트 : 실전 프로젝트 소스와 실행 파일이 각 절별로 정리되어 있고, 추가 프로그래밍 주제에 대한 코드도 함께 담았다.




## 참고 문헌 및 사이트

■ **『윈도우즈 API 정복(개정판)』**, 김상형, 한빛미디어, 2006

API 프로그래밍 서적 중 가장 방대한 내용과 풍부한 예제를 담고 있다. MFC를 배우기 전에 API 프로그래밍의 기초를 닦고자 한다면 이 책의 1장부터 8장까지 학습하기를 권장한다.

■ **코드구루(http://www.codeguru.com), 코드프로젝트(http://www.codeproject.com)**
– 프로그래밍 소스 코드 사이트

프로그래밍 언어별로 소스 코드와 튜토리얼이 제공되는 사이트로, 오랜 역사만큼 세부 항목에 관한 자료가 많이 축적되어 있다. 특히 책을 통해 해결하기 어려운 다양한 GUI 항목에 대한 해결책 및 소스 코드가 존재하므로 필수 방문 사이트 중 하나다.

■ **스택오버플로우(http://www.stackoverflow.com)** – 프로그래밍 Q&A 사이트

프로그래밍 전 분야에 걸쳐 질의 응답 및 토론이 활발하게 이뤄지는 곳이다. 실제 적용 가능한 기법들이 소개되며, 프로그래밍 도중 알 수 없는 문제에 직면했을 때 가장 먼저 방문하여 해결 방안을 검색해 볼 만한 사이트다.

■ **소스포지(http://www.sourceforge.net)** – 오픈 소스 프로젝트 사이트

오픈 소스 소프트웨어 개발/관리 사이트로, 유명한 오픈 소스 프로그램들의 탄생지다. 프로젝트 단위로 공개된 소스 코드 분석을 통해 실전 감각과 실력을 기를 수 있다.

# >> 숲과 나무 이야기

## MFC 윈도우 프로그래머 성장 전략 맵

MFC를 공부할 때 C++ 언어를 얼마나 잘 알고 있어야 하나요?

MFC는 C++ 언어가 제공하는 거의 모든 문법적 요소를 잘 적용하여 윈도우 API를 클래스 형태로 제공하는 방대한 라이브러리입니다. 따라서 C++ 언어가 제공하는 다양한 개념을 이해하고 있어야 합니다. 클래스와 인스턴스, this 포인터, 생성자와 소멸자, 정적 변수와 정적 함수, 연산자와 함수 중복 정의, 상속, 가상 함수, 템플릿 등의 개념을 미리 복습해 두기 바랍니다. C++ 언어를 잘 알수록 MFC의 구조를 이해하고 활용하는 데 큰 도움이 됩니다.

MFC로 프로그램을 만들 수 있는데 굳이 API를 배울 필요가 있나요?

MFC를 처음 배울 때 API를 잘 알고 있어야 하는 것은 아닙니다. 그러나 MFC는 윈도우 응용 프로그램 개발 과정에서 자주 반복되는 부분이나 구현이 매우 복잡한 부분을 클래스로 정리한 것으로, 모든 API 기능을 제공하는 것은 아닙니다. 실전에서는 API 함수를 섞어서 사용하는 것이 일반적이므로 자신에게 필요한 기술을 API 수준에서 꾸준히 공부해 두는 것이 좋습니다.

**1단계 : 기초 체력 다지기**

MFC를 이용한 윈도우 프로그래밍을 하려면 다음 항목의 기본을 제대로 익혀두어야 합니다.

- 객체지향 개념
- C++ 언어

목표 : C++ 언어를 이용한 객체지향적 프로그램 설계와 구현
수준 : 학교에서 숙제를 잘 한다.

이 책은 여기까지

**2단계 : 초보자 탈피하기**

MFC를 이용하여 다음 내용을 배웁니다. 여기까지 배우면 윈도우 프로그래밍의 기본은 갖춰진 셈입니다.

- 윈도우 프로그래밍 기초
- 사용자 인터페이스 구현
- 도큐먼트/뷰 구조 이해 및 응용

목표 : MFC를 이용해 기능적인 프로그램 제작
수준 : 현장에서 기본은 한다.

**3단계 : 중급자로 거듭나기**

MFC를 이용하여 고급 프로그래밍 기법을 배웁니다. 이젠 좀더 큰 규모의 실용적인 응용 프로그램을 제작할 수 있는 준비가 된 것입니다.

- 멀티스레드, 데이터베이스, 네트워크 등 고급 기술 습득
- 프로젝트 수행 방법론

목표 : MFC를 응용한 실용적인 프로그램 제작
수준 : 현장에서 제 몫을 한다.

**4단계 : 전문가로 거듭나기**

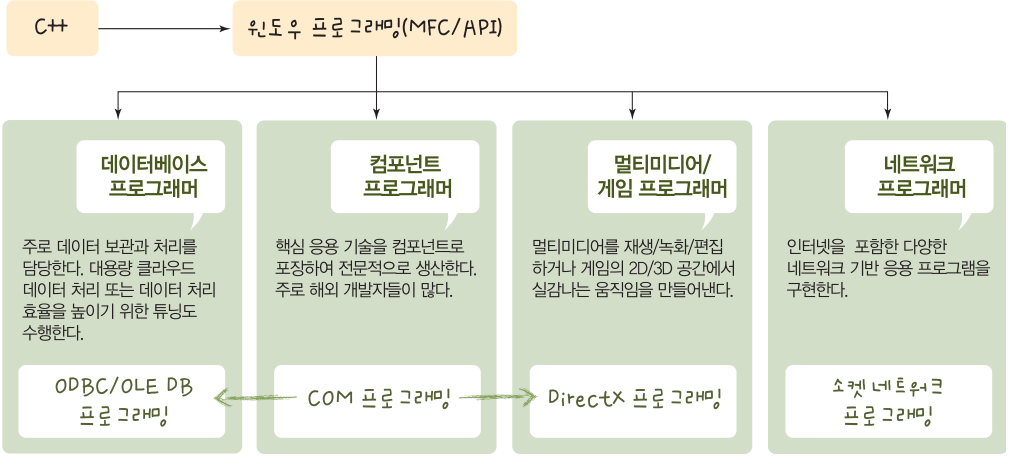
윈도우 프로그래밍은 다양한 분야와 연관되어 있습니다. 적어도 한 분야는 집중적으로 학습해보기 바랍니다.

- 컴포넌트 프로그래밍
- 웹/데이터베이스 프로그래밍
- 그래픽/멀티미디어 프로그래밍
- 네트워크 프로그래밍
- 시스템 프로그래밍

목표 : 고급 기술을 적용하여 프로그램 제작
수준 : 하고 싶은 일을 선택한다.

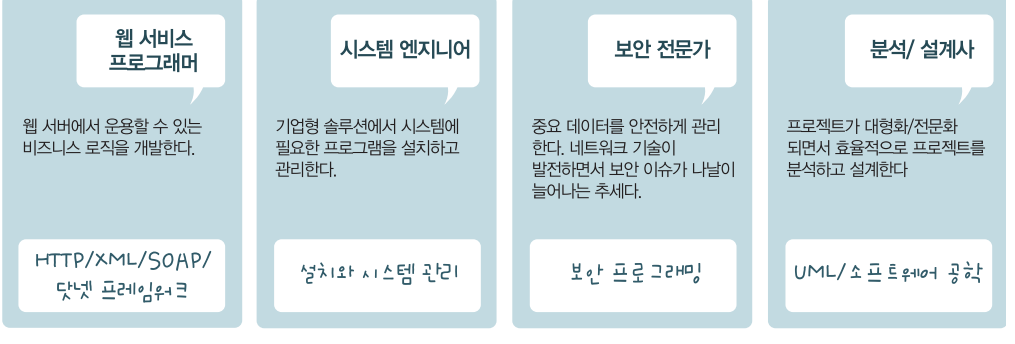
## 윈도우 프로그래밍 관련 학습 로드맵

윈도우 프로그래밍을 학습한 후, 향후 나아갈 수 있는 방향을 주요 직업 및 기술과 함께 제시하면 다음과 같다.




각각의 분야에서 고급 프로그래머로 성장하려면 기반 기술인 COM을 익히면서 윈도우 기술 및 가상 함수 컨셉을 좀더 깊게 이해해야 합니다.

네트워크 프로그래밍(소켓), 데이터베이스 프로그래밍(ODBC/OLE DB), 컴포넌트 프로그래밍(COM) 등 IT 솔루션 기반 기술을 익히고 나면 기업형 솔루션 개발 쪽으로도 나아갈 수 있다.



Chapter

09

# 대화 상자

**✻ 학습 목표**

- 대화 상자 디자인 방법을 익힌다.
- 모드형 대화 상자와 비모드형 대화 상자의 차이를 이해하고 작성 방법을 익힌다.
- 대화 상자 기반 응용 프로그램의 동작 원리를 이해하고 작성 방법을 익힌다.
- 공용 대화 상자를 다루는 방법을 배운다.

Section

# 01 대화 상자 기초

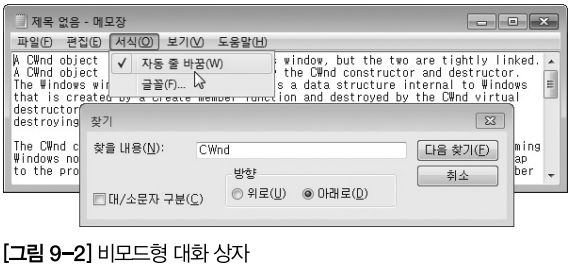
대화 상자(Dialog Box)는 다양한 컨트롤을 포함하고 있는 일종의 윈도우로, 사용자의 입력을 받거나 정보를 출력하는 용도로 사용한다. [그림 9-1]은 그림판 응용 프로그램에서 이미지 속성을 지정하는 대화 상자다. 8장에서 배운 정적, 버튼, 편집 컨트롤을 이용해 입출력을 처리함을 알 수 있다.


**[그림 9-1]** 대화 상자 예

대화 상자는 동작 방식에 따라 모드형 대화 상자와 비모드형 대화 상자로 구분한다.

- **모드형(Modal) 대화 상자** : 대화 상자를 닫아야 응용 프로그램이 다른 작업을 할 수 있다.
- **비모드형(Modeless) 대화 상자** : 대화 상자를 닫지 않아도 응용 프로그램이 다른 작업을 할 수 있다.

[그림 9-1]은 〈확인〉 또는 〈취소〉를 클릭하여 대화 상자를 닫아야 다른 작업을 할 수 있는 모드형 대화 상자다. 반면 [그림 9-2]는 메모장에서 [편집]-[찾기...] 메뉴를 선택하면 열리는 대화 상자로, 대화 상자가 열린 상태에서도 메뉴 선택이나 편집 같은 다른 작업을 할 수 있는 비모드형 대화 상자다.



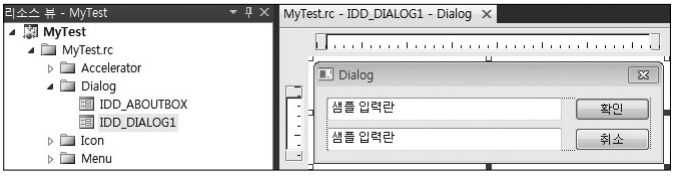

**[그림 9-2]** 비모드형 대화 상자

어떤 대화 상자를 생성하든지 맨 먼저 대화 상자를 디자인해야 한다. 이 절에서는 대화 상자 디자인과 관련하여 비주얼 C++가 제공하는 편리한 작업 환경을 사용하는 방법과 더불어 프로그래밍 요소를 제외한 몇 가지 이론적인 내용을 다룰 것이다.

## ❶ 대화 상자 템플릿

대화 상자는 운영체제 내부적으로 대화 상자 템플릿이라는 형태로 존재한다. 대화 상자 템플릿(Dialog Box Template)은 대화 상자 자체와 대화 상자에 포함된 컨트롤에 대한 모든 정보를 가진 이진(Binary) 데이터다. 운영체제는 사용자가 미리 디자인하여 실행 파일에 포함해 놓은 대화 상자 리소스를 로드하여 자동으로 대화 상자 템플릿을 생성한다.

[그림 9-3]은 MFC 응용 프로그램 프로젝트를 임의로 만들고 IDD_DIALOG1 대화 상자 리소스를 생성한 후 편집 컨트롤을 두 개 추가한 모습이다.




**[그림 9-3]** 대화 상자 디자인

메모장으로 '프로젝트명.RC' 파일을 열면 다음과 같이 대화 상자에 관한 세부 정보를 확인할 수 있다. 앞 부분에 대화 상자의 리소스 ID, 스타일, 캡션, 폰트 정보가 들어 있고 BEGIN~END 사이에 네 개의 컨트롤에 관한 정보가 나열되어 있다.

```
IDD_DIALOG1 DIALOGEX 0, 0, 229, 46
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
  DEFPUSHBUTTON  "확인",IDOK,172,7,50,14
  PUSHBUTTON     "취소",IDCANCEL,172,25,50,14
  EDITTEXT       IDC_EDIT1,7,7,156,14,ES_AUTOHSCROLL
  EDITTEXT       IDC_EDIT2,7,25,156,14,ES_AUTOHSCROLL
END
```

이와 같이 텍스트 형태로 기술된 *.RC 파일을 리소스 스크립트(Resource Script)라고 부르는데, 일반적으로 텍스트 편집기보다 비주얼 C++의 리소스 편집기를 이용해 작성한다. 완성된 리소스 스크립트는 컴파일 단계에서 리소스 컴파일러가 이진 형태인 *.RES 파일로 변환

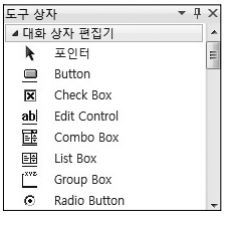
하고, 링크 단계에서 실행 파일에 포함된다. 응용 프로그램이 대화 상자를 생성하도록 요청하면, 운영체제는 실행 파일에 포함된 대화 상자 리소스를 토대로 대화 상자 템플릿을 메모리에 구성하여 대화 상자를 만들어 낸다.

## 2. 대화 상자 편집기

비주얼 C++에서는 리소스를 시각적으로 편집하는 툴을 총칭해서 리소스 편집기(Resource Editor)라 부른다. 리소스의 종류에 따라 구체적인 툴의 명칭은 달라지는데, 예를 들어 대화 상자를 편집하는 툴은 대화 상자 편집기(Dialog Editor)라 부른다. 여기서는 대화 상자 편집기의 사용법과 관련된 사항을 정리해 보자.

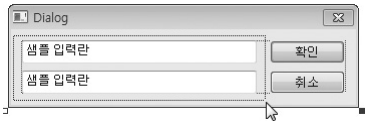
### 컨트롤 추가와 삭제

도구 상자([그림 9-4])에서 컨트롤을 선택한 후 대화 상자의 원하는 위치에서 마우스를 클릭하면 컨트롤이 추가된다(도구 상자가 보이지 않으면 [보기]-[도구 상자] 메뉴를 선택한다). 같은 종류의 컨트롤을 계속 추가하고 싶을 때는 Ctrl 키를 누른 상태에서 컨트롤을 클릭한다. 이렇게 하면 Esc 키를 누르기 전까지 해당 컨트롤이 도구 상자에서 선택된 상태로 남아 있으므로 마우스 클릭만으로 컨트롤을 계속 추가할 수 있다.

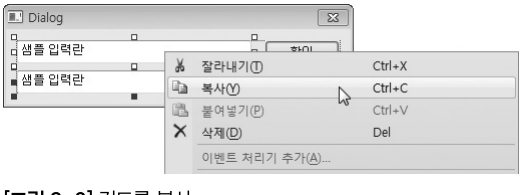



**[그림 9-4]** 도구 상자

여러 컨트롤을 한꺼번에 선택하려면 [그림 9-5]와 같이 마우스로 드래그하거나 Ctrl 또는 Shift 키를 누른 채 해당 컨트롤을 클릭하면 된다. 선택한 컨트롤은 잘라내기, 복사, 붙여넣기 등의 작업이 가능하고 Delete 키를 눌러 제거할 수도 있다.




**[그림 9-5]** 컨트롤 선택




**[그림 9-6]** 컨트롤 복사

## 컨트롤 배치

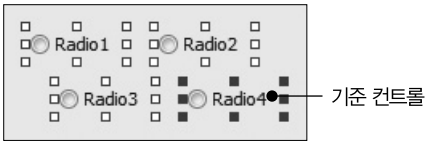
대화 상자 편집기를 사용할 때는 [그림 9-7]과 같이 비주얼 C++ 최상위 메뉴에 [서식] 메뉴가 추가된다. 이 메뉴를 이용하거나 [그림 9-8]과 같은 대화 상자 편집기 툴바를 사용해 컨트롤의 크기, 정렬, 간격 등을 설정할 수 있다. 대화 상자를 디자인할 때는 컨트롤을 눈짐작으로 배치하기보다는 [서식] 메뉴나 대화 상자 편집기 툴바를 이용하는 편이 빠르고 정확하다.

대화 상자 편집기 툴바가 보이지 않으면 비주얼 C++의 메뉴나 툴바 영역에서 컨텍스트 메뉴를 불러서 [대화 상자 편집기] 메뉴를 선택한다.


**[그림 9-7]** [서식] 메뉴

**[그림 9-8]** 대화 상자 편집기 툴바

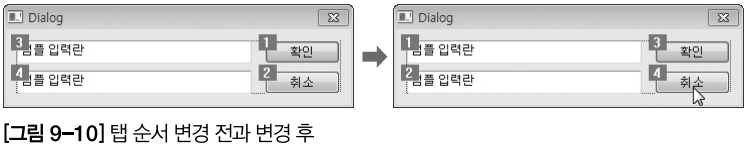
[서식] 메뉴나 대화 상자 편집기 툴바의 기능을 적용할 때는 컨트롤 하나가 기준이 되는데, 이 컨트롤은 [그림 9-9]와 같이 선택 표시가 다르므로 쉽게 구분할 수 있다. 일반적으로 마지막에 선택한 컨트롤이 기준이 되지만 Ctrl 키를 누른 상태에서 원하는 컨트롤을 마우스로 클릭하면 기준 컨트롤이 바뀐다.




**[그림 9-9]** 기준 컨트롤

## 탭 순서

탭 순서(Tab Order)는 대화 상자에서 Tab 키를 눌렀을 때 키보드 포커스가 이동하는 순서를 말한다. [서식]-[탭 순서] 메뉴를 선택하면 [그림 9-10]과 같이 현재 설정된 탭 순서가 표시되는데, 마우스로 컨트롤을 클릭하면 그 순서대로 탭 순서가 재설정된다. 마지막 컨트롤을 클릭한 후 대화 상자 바깥쪽을 클릭하면 탭 순서 변경이 끝난다.



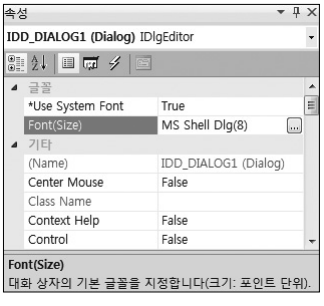

**[그림 9-10]** 탭 순서 변경 전과 변경 후

탭 순서 변경은 실제로 리소스 스크립트에서 컨트롤을 기술한 문장의 순서를 바꾸는 것에 불과하다. 다음은 탭 순서를 변경한 후에 BEGIN~END 사이의 문장 순서가 바뀐 것을 보여준다. 운영체제는 리소스 스크립트에 나열된 순서대로 컨트롤을 생성하며, Tab 키를 누르면 이 순서대로 키보드 포커스를 이동시킨다. 탭 순서 1번에 해당하는 컨트롤은 대화 상자 생성 시 자동으로 키보드 포커스를 갖게 된다는 점도 기억해 두자.

```
IDD_DIALOG1 DIALOGEX 0, 0, 229, 46
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
  EDITTEXT       IDC_EDIT1,7,7,156,14,ES_AUTOHSCROLL
  EDITTEXT       IDC_EDIT2,7,25,156,14,ES_AUTOHSCROLL
  DEFPUSHBUTTON  "확인",IDOK,172,7,50,14
  PUSHBUTTON     "취소",IDCANCEL,172,25,50,14
END
```

### 대화 상자 속성

대화 상자에 포함된 컨트롤은 물론이고 대화 상자 자체에도 다양한 속성이 있다. 대화 상자 편집기에서 컨트롤이 아닌 대화 상자 자체를 선택한 상태에서 속성 창(Alt+Enter)을 열면 [그림 9-11]과 같이 다양한 속성을 볼 수 있다.




**[그림 9-11]** 대화 상자 속성

대화 상자는 일종의 윈도우이므로 일반 윈도우의 속성을 그대로 갖고 있고 대화 상자만의 독특한 속성도 있다. 예를 들면 대화 상자에서 사용할 폰트를 선택할 수 있는데, 폰트를 변경하면 대화 상자 자체의 크기는 물론이고 대화 상자에 포함된 컨트롤의 크기도 그에 비례해서 변경된다. 리소스 스크립트에 기술된 대화 상자나 컨트롤의 크기와 위치는 절대적인 값이 아니라 현재 설정된 폰트의 크기에 비례해서 변하는 값이기 때문이다.

**현장 팁**

### 타이틀바가 없는 대화 상자 이동 방법

대화 상자 기반 응용 프로그램을 제작하다 보면, 타이틀바가 없는 대화 상자를 띄우고 싶을 때가 있다. 타이틀바는 대화 상자 속성을 통해 간단히 제거할 수 있다. 그리고 대화 상자에 스킨을 입혀 타이틀바 역할을 하는 곳을 특정 위치로 지정할 수도 있다. 그런데 타이틀바가 없으면 발생하는 불편한 점도 존재한다. 종료나 크기 변환은 크게 문제가 되지 않지만, 이동 처리가 골치 아프다. 직접 구현해 보면 MoveWindow() 함수로 처리하기 힘들다는 것을 알게 될 것이다.

처리 방법은 '이동을 위한 영역을 클릭했을 때, 비클라이언트 메시지를 생성하는 것'이다. 예를 들어, 다음과 같이 WM_LBUTTON_DOWN 메시지를 처리하는 OnLButtonDown() 함수를 구현해 보자. 마우스 왼쪽 버튼을 누르면 해당 영역을 확인한 후, 비클라이언트 메시지를 발송한다. 이로써 마우스 왼쪽 버튼을 누른 상태로 대화 상자를 이동시킬 수 있다. 또한 point 변수를 활용하여 특정 영역으로 제한시킬 수도 있다.

```
void CTestDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
  CPoint pt = point;
  ClientToScreen(&pt);
  PostMessage(WM_NCLBUTTONDOWN, HTCAPTION, MAKELPARAM(pt.x, pt.y));
  CDialog::OnLButtonDown(nFlags, point);
}
```

다른 방법으로, WM_NCHITTEST 메시지를 처리하는 OnNcHitTest() 함수를 사용할 수 있다. 아래 코드는 클라이언트 영역에서 타이틀바를 의미하는 HTCAPTION을 리턴하는 예제다.

```
UINT CTestDlg::OnNcHitTest(CPoint point)
{
  int nHit = CDialog::OnNcHitTest(point);
  if(nHit == HTCLIENT)
    nHit = HTCAPTION;
  return nHit;
}
```

또는 다음과 같이 영역과 관계없이 무조건 HTCAPTION을 리턴하여 간단히 처리할 수도 있다.

```
UINT CTestDlg::OnNcHitTest(CPoint point)
{
  return HTCAPTION;
}
```

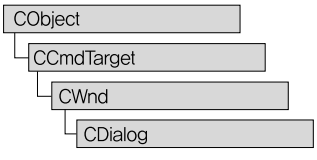
HTCAPTION을 리턴하기 전에 point 변수를 사용하여 필요에 따라 특정 영역으로 제한시킬 수 있으며, 특별히 영역을 지정하지 않으면 대화 상자의 배경을 클릭하여 이동시킬 수 있다. 만약 컨트롤이 있는 경우 해당 컨트롤이 메시지를 처리해 버리기 때문에 대화 상자에 그 메시지가 전달되지 않는다. 따라서 대화 상자 동작이 달라질까 특별히 걱정할 필요가 없다.

Section

# 2 모드형 대화 상자

모드형 대화 상자에서 '모드형(Modal)'은 현재 대화모드(Dialog-Mode)이므로 반드시 사용자가 대화 상자를 통해 입력을 하거나 취소를 함으로써 대화를 마쳐야 다음으로 진행할 수 있다는 의미다. 모드형 대화 상자는 개념도 쉽고 코딩 방법도 비모드형 대화 상자에 비해 간단하기 때문에 많이 사용된다.

MFC에서 모드형 대화 상자를 다룰 때는 CDialog 클래스를 사용하는데, 형식 차이가 있을 뿐 비모드형 대화 상자에서도 같은 클래스를 사용한다. [그림 9-12]에서 보는 바와 같이 CDialog 클래스는 CWnd 클래스를 상속받는다. 이는 대화 상자 역시 윈도우라는 점을 클래스 설계에 반영한 것이며, 실제로 CWnd 클래스의 일부 멤버 함수는 대화 상자 작성에도 유용하다.




**[그림 9-12]** MFC 클래스 계층도

## 1. 모드형 대화 상자 생성

MFC를 이용해 모드형 대화 상자를 만드는 순서는 다음과 같다.

① 대화 상자 리소스를 작성한다.

② CDialog 클래스 또는 CDialog의 파생 클래스 객체를 만든다. 이때 대화 상자 리소스 ID를 생성자의 인자로 넘겨준다.

③ CDialog::DoModal( ) 함수를 호출한다.

실습을 통해 가장 간단한 형태의 모드형 대화 상자를 만들고, CDialog 클래스가 제공하는 기능을 자세히 살펴보자.

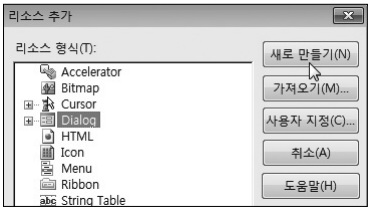
## 실습 9-1 모드형 대화 상자 만들기

클라이언트 영역에서 마우스 왼쪽 버튼을 누르면 [그림 9-13]과 같은 간단한 형태의 대화 상자가 열리는 프로그램을 작성해 보자. 〈확인〉 또는 〈취소〉를 클릭하면 대화 상자가 닫히고, 메시지 상자로 결과를 표시하는 기능도 제공할 것이다.


**[그림 9-13]** 실행 결과

**1** 응용 프로그램 마법사를 이용하여 ModalDialog1 프로젝트를 만든다. 생성 옵션은 7장의 FileIoTest 프로그램과 동일하다.

**2** '리소스 뷰'로 전환한 후 [프로젝트]-[리소스 추가...] 메뉴를 선택하여 대화 상자 리소스를 추가한다. 대화 상자 ID는 자동으로 할당되는 IDD_DIALOG1을 그대로 사용한다.




**[그림 9-14]** 대화 상자 리소스 추가

**3** 속성 창을 이용해 뷰 클래스에 WM_LBUTTONDOWN 메시지 핸들러를 추가한 후 다음 코드를 작성한다. CDialog::DoModal( ) 함수의 리턴값을 검사하면 어느 버튼을 눌러서 대화 상자를 닫았는지 알 수 있다.

```
void CModalDialog1View::OnLButtonDown(UINT nFlags, CPoint point)
{
  CDialog dlg(IDD_DIALOG1); // 대화 상자 리소스 ID를 생성자에 넘겨준다.
  if(dlg.DoModal() == IDOK)
    MessageBox(_T("확인 버튼 누름"));
  else // 리턴값이 IDCANCEL인 경우
    MessageBox(_T("취소 버튼 누름"));
}
```

실습을 통해 모드형 대화 상자를 작성하는 방법을 살펴봤지만, 예제처럼 CDialog 클래스를 직접 사용하는 경우는 많지 않다. CDialog 클래스가 제공하는 기본 동작은 응용 프로그램의 요구 사항 대부분을 만족시킬 수 없기 때문이다. 따라서 대개는 CDialog 클래스를 상속받아 새로운 클래스를 만들고 기존 동작을 변경하거나 기능을 추가한다.

CDialog 클래스의 멤버 함수를 재정의(Override)할 때 주요 대상이 되는 것은 가상 함수다. 가상 함수는 대개 MFC 내부에서 자동으로 호출되는 것이므로 호출 시점을 잘 알고 있어야 제대로 활용할 수 있다. 자주 사용되는 CDialog 클래스의 가상 함수를 호출 시점과 용도를 중심으로 정리하면 다음과 같다.

### ■ virtual BOOL CDialog::OnInitDialog();

- **호출 시점** : 응용 프로그램이 CDialog::DoModal( ) 함수를 호출하여 대화 상자 생성을 시작하면 WM_CREATE 메시지가 발생한다. 운영체제는 이 메시지를 받아 내부적으로 대화 상자 템플릿을 구성하고 대화 상자 자체와 거기에 포함된 컨트롤을 생성한다. 대화 상자가 만들어지면 화면에 보이기 직전에 WM_INITDIALOG 메시지가 발생하는데, 이 메시지에 대응하여 OnInitDialog( ) 함수가 호출된다. 즉, OnInitDialog( ) 함수는 WM_INITDIALOG 메시지 핸들러라고 할 수 있다. 기능으로 보면 메시지 핸들러이지만 가상 함수로 정의되어 있으므로 메시지 맵을 사용하지 않아도 자동으로 호출된다.
- **용도** : 컨트롤을 초기화하거나 키보드 포커스를 변경한다.

### ■ virtual void CDialog::OnOK();

- **호출 시점** : ID가 IDOK인 버튼(앞의 예제의 〈확인〉 버튼)을 누르면 호출된다. 즉, OnOK( ) 함수는 ID가 IDOK인 버튼의 클릭을 처리하는 메시지 핸들러라고 할 수 있다. 기능으로 보면 메시지 핸들러지만 가상 함수로 정의되어 있으므로 메시지 맵을 사용하지 않아도 자동으로 호출된다.
- **용도** : 컨트롤의 값을 읽거나 값의 타당성 여부를 검사한 후 대화 상자를 닫는다.

### ■ virtual void CDialog::OnCancel();

- **호출 시점** : ID가 IDCANCEL인 버튼(앞의 예제의 〈취소〉 버튼), 윈도우의 〈종료〉 버튼, Esc 키를 누르면 호출된다. 즉, OnCancel( ) 함수는 이들 세 가지 이벤트를 처리하는 메시지 핸들러라고 할 수 있다. 기능으로 보면 메시지 핸들러지만 가상 함수로 정의되어 있으므로 메시지 맵을 사용하지 않아도 자동으로 호출된다.
- **용도** : 대화 상자를 닫는다.

모드형 대화 상자에서 가상 함수를 재정의할 때는 보통 자신의 프로그램에 맞게 처리한 후 베이스 클래스(CDialog)의 가상 함수를 호출한다. CDialog::OnOK( ) 함수나 CDialog::OnCancel( ) 함수에는 대화 상자를 닫는 기능이 기본으로 구현되어 있다. 따라서 단순히 대화 상자를 닫기만 할 때는 가상 함수를 재정의할 필요가 없다. 두 함수의 구현 코드를 보면 다음과 같이 내부적으로 CDialog::EndDialog( ) 함수를 호출하여 대화 상자를 종료한다. CDialog::EndDialog( ) 함수에 넘겨주는 정수형 인자는 CDialog::DoModal( ) 함수의 리턴값이 되어 응용 프로그램에 전달된다.

```
void CDialog::OnOK()
{
  UpdateData(TRUE); // 이 함수의 의미는 DDX/DDV에서 배운다.
  EndDialog(IDOK);
}

void CDialog::OnCancel()
{
  EndDialog(IDCANCEL);
}
```

〈확인〉이나 〈취소〉 버튼을 제외한 다른 버튼을 눌렀을 때 대화 상자가 종료되게 하려면, 해당 버튼의 BN_CLICKED 통지 메시지 핸들러에서 CDialog::EndDialog( ) 함수를 호출한다. 이때 CDialog::EndDialog( ) 함수의 인자로 전달하는 정수값은 IDOK, IDCANCEL 등의 정의된 상수값과 겹치지 않아야 한다. WINUSER.H 파일을 찾아보면 대화 상자를 위한 상수값이 다음과 같이 정의되어 있으므로 이 값과 충돌하지 않는 범위에서 자유롭게 선택해 사용한다. 비주얼 C++는 사용자가 정의한 버튼에 대해 자동으로 숫자값(ID)을 할당해 주는데, 이 값은 아래에 나열된 값과 중복되지 않는다. 따라서 버튼 ID를 CDialog::EndDialog( ) 함수에 그대로 사용하면 편리하다.

```
/* Dialog Box Command IDs */
#define IDOK 1
#define IDCANCEL 2
#define IDABORT 3
#define IDRETRY 4
#define IDIGNORE 5
#define IDYES 6
#define IDNO 7
#define IDCLOSE 8
#define IDHELP 9
```

## 실습 9-2 모드형 대화 상자 활용하기

모드형 대화 상자 연습을 위해 다음과 같은 응용 프로그램을 작성해 보자. 클라이언트 영역에서 마우스 왼쪽 버튼을 누르면 모드형 대화 상자가 열린다. 편집 컨트롤과 콤보 박스 컨트롤로 텍스트와 폰트명을 입력받고 〈확인〉을 클릭하면, 지정한 폰트로 텍스트를 화면에 출력한다. 〈지우기〉를 클릭하면 화면을 지운다. 이전에 입력한 값을 일부만 바꿀 경우를 대비해서 대화 상자가 열릴 때마다 편집 컨트롤과 콤보 박스 컨트롤은 이전 값으로 초기화된다.

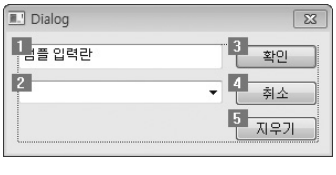

**[그림 9-15]** 실행 결과

❶ 응용 프로그램 마법사를 이용하여 ModalDialog2 프로젝트를 만든다. 생성 옵션은 7장의 FileIoTest 프로그램과 동일하다.

❷ '리소스 뷰'로 전환한 후 [프로젝트]-[리소스 추가...] 메뉴를 선택하여 대화 상자 리소스를 추가한다. 대화 상자 ID는 자동으로 할당되는 IDD_DIALOG1을 그대로 사용한다.

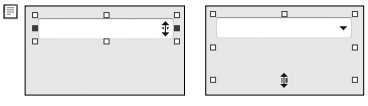

**[그림 9-16]** 대화 상자 리소스 추가

❸ 대화 상자에 편집 컨트롤(1), 콤보 박스 컨트롤(2), 버튼 컨트롤(5)을 추가한다. 또한 표를 참고하여 컨트롤 속성을 변경한다. 그리고 [서식]-[탭 순서] 메뉴를 이용해 그림에 표시된 번호대로 탭 순서를 설정한다. 〈확인〉과 〈취소〉 버튼은 기본값을 그대로 사용하므로 따로 표시하지 않았다.



| 탭 순서 | ID | 변경 사항 |
|---|---|---|
| 1 | IDC_STR | 없음 |
| 2 | IDC_FONT | 'Data'에 "굴림;궁서;바탕"을 입력한다.<br>'Type'을 Drop List로, 'Sort'를 False로 변경한다. |
| 5 | IDCLEAR | 'Caption'을 "지우기"로 변경한다. |

**[그림 9-17]** 대화 상자 디자인과 컨트롤 속성 설정



콤보 박스를 대화 상자에 배치할 때 미리 높이를 조정해 주지 않으면 나중에 실행할 때 높이가 너무 낮아서 항목을 선택할 때 불편할 수 있다. 왼쪽 그림처럼 디자인 화면에서 콤보 박스 우측 ▼을 클릭하면 오른쪽 그림과 같이 실행 시 크기를 미리 볼 수 있다. 마우스로 드래그하여 적당한 높이로 변경한다.

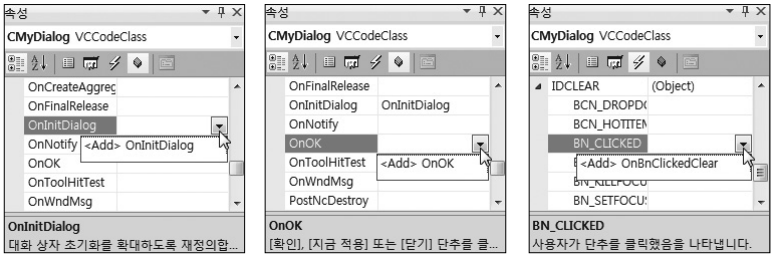
**4** [프로젝트]-[클래스 추가...] 메뉴를 선택하거나 대화 상자 디자인 화면에서 마우스 오른쪽 버튼을 눌러 [클래스 추가...] 메뉴를 선택하면 [그림 9-18]과 같은 MFC 클래스 추가 마법사가 열린다. 클래스 이름으로 "CMyDialog"를 입력하고 기본 클래스로 CDialog를 선택한 후 〈마침〉을 클릭한다.


**[그림 9-18]** 대화 상자 클래스 생성

**5** 대화 상자 디자인 화면에서 컨트롤을 선택하고 마우스 오른쪽 버튼을 눌러 [변수 추가...] 메뉴를 실행하면 컨트롤 변수를 만들 수 있다. 콤보 박스 컨트롤에 대응하는 컨트롤 변수 m_combo를 만든다.


**[그림 9-19]** 컨트롤 변수 생성

❻ '클래스 뷰'에서 CMyDialog 클래스를 선택한 상태로 속성 창을 열고, [그림 9-20]을 참고하여 가상 함수(OnInitDialog( ), OnOK( ))와 IDCLEAR 버튼의 BN_CLICKED 통지 메시지 핸들러를 추가한다.




**[그림 9-20]** 가상 함수와 통지 메시지 핸들러 추가

❼ CMyDialog::OnBnClickedClear( ) 함수에 다음 한 줄을 추가한다. 〈지우기〉 버튼을 클릭하면 대화 상자가 닫히고, 대화 상자를 호출한 쪽에는 CDialog::DoModal( ) 함수의 리턴값으로 IDCLEAR가 전달된다.

```
void CMyDialog::OnBnClickedClear()
{
  EndDialog(IDCLEAR);
}
```

❽ CMyDialog 클래스 헤더 파일에 멤버 변수 두 개를 추가한다. m_str은 입력한 텍스트를, m_font는 선택한 폰트 번호를 저장할 것이다.

```
class CMyDialog : public CDialog
{
...
public:
  CComboBox m_combo;
  CString m_str; // 사용자가 입력한 텍스트 저장
  int m_font; // 사용자가 선택한 폰트 번호 저장
...
};
```

❾ 가상 함수 CMyDialog::OnInitDialog( )와 CMyDialog::OnOK( )에 다음 코드를 추가한다. 대화 상자가 생성될 때 호출되는 OnInitDialog( ) 함수에서는 m_str, m_font 변수 값으로 편집 컨트롤과 콤보 박스 컨트롤을 초기화한다. 〈확인〉 버튼을 클릭할 때 호출되는 OnOK( ) 함수에서는 컨트롤 값을 읽어서 m_str, m_font 변수에 저장하고 대화 상자를 닫는다. 여기까지 작성한 후 컴파일, 링크가 잘 되는지 확인하고 다음 단계로 진행한다.

```
BOOL CMyDialog::OnInitDialog()
{
  CDialog::OnInitDialog();
  SetDlgItemText(IDC_STR, m_str);
  m_combo.SetCurSel(m_font);
  return TRUE;
}

void CMyDialog::OnOK()
{
  GetDlgItemText(IDC_STR, m_str);
  m_font = m_combo.GetCurSel();
  CDialog::OnOK();
}
```

10 이제 대화 상자를 생성하는 코드를 뷰 클래스에 작성해 보자. 모드형 대화 상자는 열리고 닫힐 때마다 생성과 파괴가 반복되는 특징이 있으므로 입력된 데이터를 계속 유지할 수 없다는 문제가 있다. 따라서 대화 상자를 생성하는 쪽에서 대화 상자 데이터를 저장해 두어야 한다. 이 예제는 클라이언트 영역에서 마우스 왼쪽 버튼을 누르면 대화 상자를 생성하므로 뷰 객체에서 대화 상자 데이터를 유지하는 것이 자연스럽다. 다음과 같이 뷰 클래스 헤더 파일에 대화 상자 데이터를 유지하는 멤버 변수를 추가하고 생성자에서 초기화하자.

```
class CModalDialog2View : public CView
{
protected: // serialization에서만 만들어집니다.
  CModalDialog2View();
  DECLARE_DYNCREATE(CModalDialog2View)

// 특성입니다.
public:
  CModalDialog2Doc* GetDocument() const;
  CString m_str;
  int m_font;
```

```
CModalDialog2View::CModalDialog2View()
{
  m_str = _T("");
  m_font = 0;
}
```

11 뷰 클래스에 WM_LBUTTONDOWN 메시지 핸들러를 추가하고 다음 코드를 작성한다. CMyDialog 클래스를 사용하므로 관련 헤더 파일을 포함하는 것을 잊지 않도록 하자.

```
#include "ModalDialog2Doc.h"
#include "ModalDialog2View.h"
#include "MyDialog.h"

...

void CModalDialog2View::OnLButtonDown(UINT nFlags, CPoint point)
{
  CMyDialog dlg; // C++ 대화 상자 객체를 생성한다.
  dlg.m_str = m_str; // 뷰 객체의 멤버 변수값으로 대화 상자 객체의
  dlg.m_font = m_font; // 멤버 변수를 초기화한다.

  int result = dlg.DoModal(); // 모드형 대화 상자를 연다.
  if(result == IDOK){
    m_str = dlg.m_str; // 대화 상자 객체의 멤버 변수값을
    m_font = dlg.m_font; // 뷰 객체의 멤버 변수에 저장한다.
    Invalidate(); // 뷰 화면에 출력한다.
  }
  else if(result == IDCLEAR){
    m_str = _T("");
    Invalidate(); // 뷰 화면에 출력한다.
  }
}
```

12 마지막으로, 뷰 클래스의 OnDraw( ) 함수에 출력 코드를 작성한다. 뷰 클래스에서 유지하는 m_font 값에 따라 폰트를 선택하고 m_str에 저장된 텍스트를 화면에 출력하면 된다. 함수 인자가 초기에 주석 처리되어 있으므로 주석을 제거하는 것을 잊지 않도록 하자.

```
void CModalDialog2View::OnDraw(CDC* pDC)
{
  CModalDialog2Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
    return;

  CFont font;
  CString fontname;
```

```
    if(m_font == 0) fontname = _T("굴림");
    else if(m_font == 1) fontname = _T("궁서");
    else if(m_font == 2) fontname = _T("바탕");
    font.CreatePointFont(400, fontname);

    pDC->SelectObject(&font);
    pDC->TextOut(10, 10, m_str);
}
```

**현장 팁**

### 문자열 리턴하기

많은 초보자가 문자열을 그대로 리턴하는 실수를 한다. 예를 들면 다음과 같다.

```
TCHAR *MyFunc(void)
{
  TCHAR szName[100];
  ...
  return szName;
}
```

언뜻 보면 동작할 것 같지만 실제로는 문제가 많다. szName은 지역 변수로 스택(stack)에 정의된 것이기 때문에 함수가 종료되면 바로 소멸된다. 운 좋게 소멸되기 전에 사용되어 동작하는 경우도 있지만, 언젠가는 잘못된 메모리를 참조하는 문제가 발생한다. 디버그 모드에서는 동작하지만 릴리즈 모드에서는 동작하지 않는 상황이 대표적인 경우다. 따라서 다음과 같이 문자열을 인자로 받아가는 방법을 사용할 수 있다.

```
bool MyFunc(TCHAR szName[100])
{
  _tcscpy(szName, _T("Max")); // TCHAR용 strcpy() 함수 호출
  return true;
}
```

그런데 넘겨주는 배열의 크기도 알아야 하고, 호출하는 쪽에서 미리 선언해야 하므로 불편하다. 따라서 다음과 같이 크기를 넘겨주는 방식을 API 함수에서 많이 볼 수 있다.

```
bool MyFunc(TCHAR *pzName, int iSize)
{
  _tcscpy(szName, _T("Max")); // TCHAR용 strcpy() 함수 호출
  return true;
}
```

따로 기술하지 않았지만 전달받은 크기를 넘지 않도록 복사할 값의 크기와 비교하는 부분이 있어야 한다. 다시 처음으로 돌아가서, 인자로 넘기지 않고 내부에서 문자열을 리턴하는 방법을 살펴보자. C나 C++에서 가장 일반적인 방식은 다음과 같이 new 연산자 등으로 힙(heap)에 메모리를 할당하여 리턴하는 방식이다.

```
TCHAR *CreateName(int iIndex)
{
  TCHAR *pszName = new TCHAR[100];
  _tcscpy(pszName, _T("Max")); // TCHAR용 strcpy() 함수 호출
  return pszName;
}
```

메모리를 할당해서 넘기면 받은 쪽에서 delete 연산자를 사용해 반드시 해제해야 한다. 이 때문에 함수 이름이 'Create'로 시작하는 경우가 많다. 리턴받은 포인터는 반드시 해제해야 메모리 누수가 발생하지 않는다. 이 방식은 메모리 해제를 해야 하는 부담이 너무 크기 때문에 사용하기 불편하다. 대신 C++에서는 메모리를 관리해 주는 클래스를 사용하는 방식이 있다. MFC의 경우 CString 클래스를 제공한다. 다음과 같이 사용하면 그만이다.

```
CString GetName(int iIndex)
{
  CString str = _T("Max");
  return str;
}
```

CString 클래스는 내부적으로 new 연산자를 사용하여 문자열에 필요한 메모리를 힙에 할당하기 때문에 중도에 소멸되는 문제가 없다. 또한 CString 객체가 소멸할 때 소멸자가 할당된 메모리를 해제하기 때문에 메모리 문제도 없다. 이 때문에 MFC에서 문자열은 보통 CString으로 처리한다. 만약 MFC를 사용하지 않는 환경에서 별도의 클래스를 구현하거나 메모리 할당을 사용하고 싶지 않다면, 다음과 같이 간단히 static 변수를 사용하는 방법이 있다.

```
TCHAR *GetName(int iIndex)
{
  static TCHAR szName[100];
  _tcscpy(szName, _T("Max")); // TCHAR용 strcpy() 함수 호출
  return szName;
}
```

정적(Static) 변수를 위한 메모리는 프로그램이 시작할 때 할당되어 종료할 때까지 소멸되지 않으므로 그대로 리턴해도 아무런 문제가 없다. 단, 정적 변수는 하나의 메모리만 확보된 상황이다. 따라서 이 함수가 다시 호출되어 값이 바뀌면 보관하고 있는 문자열 값도 바뀌므로 주의해야 한다.

## 요약 / 연습문제

### 1 대화 상자

다양한 컨트롤을 포함하고 있는 일종의 윈도우로, 사용자의 입력을 받거나 정보를 출력하는 용도로 사용한다.

### 2 대화 상자의 구분

대화 상자는 동작 방식에 따라 모드형 대화 상자와 비모드형 대화 상자로 구분한다. 모드형 대화 상자는 대화 상자를 닫아야 응용 프로그램이 다른 작업을 할 수 있지만, 비모드형 대화 상자는 대화 상자를 닫지 않아도 응용 프로그램이 다른 작업을 할 수 있다.

### 3 대화 상자 템플릿

대화 상자 자체와 대화 상자에 포함된 컨트롤에 대한 모든 정보를 가진 이진 데이터다.

### 4 모드형 대화 상자 만드는 방법

모드형 대화 상자는 대화 상자 리소스 작성 → CDialog의 파생 클래스 정의 및 객체 생성 → CDialog::DoModal( ) 함수 호출 절차를 통해 생성한다.

### 5 DDX/DDV

- DDX : 컨트롤과 멤버 변수 간의 데이터 교환을 자동화한다.
- DDV : 컨트롤에 입력한 데이터의 타당성을 검사한다.

### 6 비모드형 대화 상자 만드는 방법

MFC로 비모드형 대화 상자를 작성할 때는 아래의 규칙을 따른다.

- CDialog::DoModal( ) 대신 CDialog::Create( ) 함수를 이용해 대화 상자를 생성한다.
- CDialog::EndDialog( ) 대신 CWnd::DestroyWindow( ) 함수를 이용해 대화 상자를 닫는다.

### 7 대화 상자 기반 응용 프로그램

대화 상자가 메인 윈도우 역할을 하는 응용 프로그램을 말한다.

### 8 공용 대화 상자

응용 프로그램에서 흔히 필요한 대화 상자를 윈도우 운영체제 수준에서 제공하는 것으로, MFC에서는 CCommonDialog의 다양한 파생 클래스로 기능을 제공한다.

## 연습문제

**1** ModalDialog1 예제에 "안녕하세요!" 글자가 대화 상자 중앙에 있는 정적 컨트롤을 추가하시오.

HINT 리소스 편집기 사용

**2** ModalDialog2 예제에 편집 컨트롤을 추가하여 글자 크기를 지정할 수 있도록 하시오. 단, 편집 컨트롤은 숫자만 입력 가능하도록 스타일을 설정한다.

HINT 글자 크기 지정은 CFont::CreatePointFont( ) 함수의 첫 번째 인자 이용

**3** ModalDialog2 예제에 콤보 박스 컨트롤을 추가하여 드랍다운 형태로 글자 색상(빨간색, 초록색, 파란색)을 지정할 수 있도록 하시오.

HINT 글자 색상 지정은 CDC::SetTextColor( ) 함수 이용

**4** ModeDialog2 예제에 체크 박스 컨트롤을 추가하여 폰트의 볼드체를 ON/OFF로 처리할 수 있도록 하시오.

HINT 볼드체 폰트 생성은 CFont::CreateFont( ) 함수 이용

**5** ModeDialog3 예제에 편집 컨트롤을 추가하여 글자 크기를 지정할 수 있도록 하시오. 단, 편집 컨트롤은 숫자만 입력 가능하도록 스타일을 설정하고, DDV를 사용하여 글자 크기를 10부터 40까지 제한한다.

HINT 글자 크기 지정은 CFont::CreatePointFont( ) 함수의 첫 번째 인자 사용

**6** ModelessDialog 예제를 다음과 같이 수정하시오. 먼저 대화 상자에 "최상위"라는 체크 박스 컨트롤을 추가하고, 체크 박스를 체크하면 대화 상자가 최상위 윈도우가 되고 체크를 해제하면 보통 상태 윈도우가 된다.

HINT 최상위 윈도우 지정과 해제는 CWnd::SetWindowPos( ) 함수의 첫 번째 인자 이용

## 심화문제

**1** DialogBase 예제를 다음과 같이 수정하시오. 대화 상자를 마우스 왼쪽 버튼으로 더블 클릭하면 파일 대화 상자(CFileDialog)가 나타난다. 그리고 대화 상자 배경을 파일 대화 상자에서 선택한 비트맵 파일로 변경한다. 단, 파일 열기 대화 상자에서 비트맵 파일(*.bmp)을 선택하기 위한 필터가 반드시 설정되어 있어야 한다.

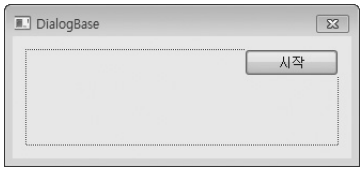
HINT 윈도우 배경은 WM_ERASEBKGND 메시지 처리, 배경 출력은 CDC::BitBlt( ) 함수 이용, 비트맵 파일로부터 CBitmap 클래스 객체를 생성하는 함수 제작(예 코드구루 (http://www.code guru.com) 프로그래밍 사이트의 LoadBMPImage( ) 함수), BITMAPFILEHEADER 구조체와 BITMAPINFOHEADER 구조체 이용, DIB(Device Independent Bitmap)를 DDB(Device Dependent Bitmap)로 바꿔주는 ::CreateDIBitmap( ) API 함수 이용

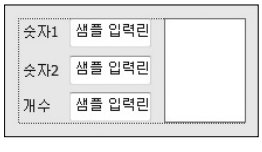
**2** 대화 상자에서 콤보 박스 컨트롤에서 폰트를 변경하면 메인 윈도우의 글자에 바로 반영되도록 ModalDialog2 예제를 수정하시오. 단 대화 상자에서 〈확인〉 버튼을 누르면 폰트가 변경되지만, 〈취소〉 버튼을 누르면 원래 폰트로 설정된다.

HINT 윈도우 메시지 전달은 CWnd::SendMessage( ) 또는 CWnd::PostMessage( ) 함수 이용,
사용자 메시지 수신은 메시지 맵에서 ON_MESSAGE( ) 매크로 이용

**3** DialogBase 예제를 다음과 같이 수정하시오. 새 대화 상자 리소스를 추가하고 그림과 같이 〈시작〉 버튼을 제외한 나머지 요소를 새 대화 상자 리소스로 이동시킨다. 새 대화 상자의 속성은 테두리 없는 자식 속성(Style : Child, Border : None)으로 변경한다. 그리고 자식 윈도우를 생성하여 본래 DialogBase 예제와 동일한 모습으로 동작하게 만든다.

HINT 대화 상자 기반 자식 윈도우 생성은 CDialog::Create( ) 함수, 자식 윈도우 위치 지정은 CWnd::MoveWindow( ) 함수, 화면에 나타나게 하는 것은 CWnd::ShowWindow( ) 함수 이용



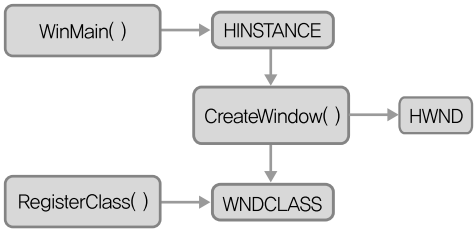
현장의 목소리 #09

# HWND = HINSTANCE + WNDCLASS

윈도우 프로그래밍을 하다 보면 윈도우 핸들(이후 HWND)을 자주 본다(물론 MFC에서는 CWnd로 포장해 놓았다). 윈도우는 크고 작은 부모 및 자식 윈도우로 구성되며, 각각 고유의 핸들을 갖는다. 그리고 개발자는 핸들을 통해 윈도우를 관리하거나 소멸시킨다. 콘솔 프로그램에는 이런 개념이 없고, 단지 main() 함수로 시작하고 끝나는 인스턴스만 존재한다. 물론 윈도우에도 WinMain() 함수로 시작하고 끝나는 인스턴스가 있는데, 이 인스턴스도 고유 핸들(이후 HINSTANCE)로 관리된다. 정리하면 다음과 같다.

"콘솔 프로그램과 윈도우 프로그램 모두 메인 함수가 있고 메인 함수가 종료될 때까지 인스턴스가 유지된다. 다만 윈도우 프로그램은 윈도우를 띄운 후, 윈도우가 종료되기 전까지 인스턴스가 종료되지 않도록 '메시지 루프'를 돌린다."

그렇다면 윈도우 프로그래밍에서 윈도우를 생성할 때 등록하는 윈도우 클래스(이후 WNDCLASS)는 무엇인가? 모든 프로그램이 항상 이 일을 해야 하는가? 이 질문에 답하려면 윈도우 운영체제에서 윈도우 프로그래밍을 하는 한, 다음 HWND, HINSTANCE, WNDCLASS의 관계를 이해해야 한다.

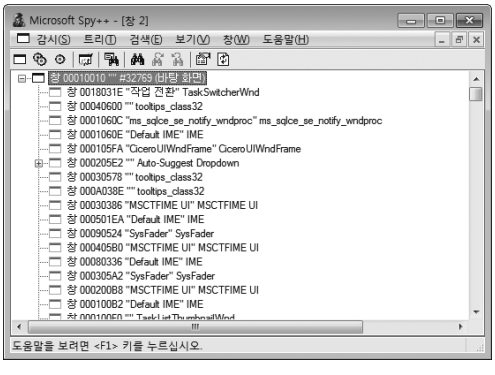



▲ HWND, HINSTANCE, WNDCLASS 관계

HINSTANCE는 ::WinMain() 함수의 인자로 제공되고, WNDCLASS는 ::RegisterClass() API 함수가 리턴한다. 그리고 HINSTANCE와 WNDCLASS를 ::CreateWindow() API 함수에 입력하면, 비로소 HWND를 얻는다. 따라서 HINSTANCE와 WNDCLASS는 직접 관련은 없고 HWND에 대해 다음 등식이 성립한다.

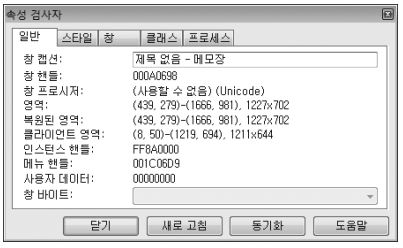
```
HWND = HINSTANCE + WNDCLASS
```

위의 등식을 반대로 해석하면, HWND를 알면 HINSTANCE와 WNDCLASS도 알 수 있다는 의미다. 이 내용을 스파이(Spy++) 유틸리티로 직접 확인해 보자. 스파이 유틸리티에서 [감시(S)]-[창(W)] 메뉴를 선택하면 시스템에 존재하는 윈도우 정보가 모두 표시된다. 여기서 트리 항목은 윈도우를, 트리 수준은 윈도우 레벨을 의미한다. 레벨이 가장 높은 윈도우는 바탕 화면 윈도우(Desktop Window)고 이것이 맨 바닥에 위치한다. 그리고 이 윈도우를 기준으로 하위 레벨 윈도우가 차례로 쌓인다. 윈도우를 이렇게 열거하기 위해 ::GetWindow()와 ::GetNextWindow() API 함수가 제공된다.




▲ 스파이 유틸리티를 통한 윈도우 목록 확인

트리 항목 중 하나를 더블 클릭하면 속성 페이지가 나타난다. [일반] 탭의 '창 핸들' 항목을 통해 HWND를, '인스턴스 핸들' 항목을 통해 HINSTANCE를 확인할 수 있다. 그리고 [클래스] 탭에서 WNDCLASS에 대한 정보를 얻을 수 있다.




▲ 속성 페이지를 통한 HWND, HINSTANCE, WNDCLASS 정보 확인

이렇게 특정 윈도우, 즉 HWND를 지정함으로써 HINSTANCE와 WNDCLASS 정보를 확인할 수 있다. Visual Studio 설명서를 통해 API 함수를 살펴보면, 다음과 같이 HWND로부터 HINSTANCE를 얻을 수 있는 ::GetWindowLong() API 함수를 발견할 수 있다.

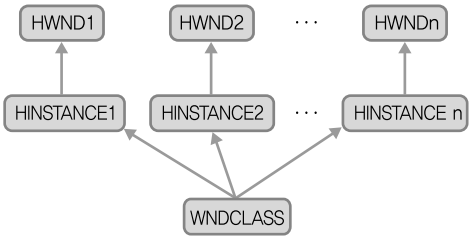
```
// hWnd는 HWND 타입 변수로, 특정 윈도우를 가리킨다.
HINSTANCE hInstance = ::GetWindowLong(hWnd,
  GWL_HINSTANCE);
```

그러나 WNDCLASS 정보는 HWND로부터 직접 얻을 수 없고 다음과 같이 ::GetClassName()과 ::GetClassInfo() API 함수를 연이어 거쳐야 한다. 단, WNDCLASS의 개별 정보만 얻거나 수정할 목적이라면, HWND만으로 ::GetClassLong()/::SetClassLong() API 함수를 사용할 수 있다.

```
// 클래스 이름 얻기
TCHAR szClassName[256];
::GetClassName(hWnd, szClassName,
  sizeof(szClassName));

// 클래스 정보 얻기
WNDCLASS WndClass;
::GetClassInfo(hInstance, szClassName,
  WndClass);
```

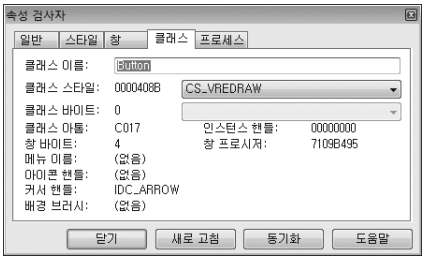
위 코드에서 다룬 내용을 주의 깊게 살펴보자. HWND에서 WNDCLASS 정보를 얻으려면 HINSTANCE를 경유함을 알 수 있다. 즉, 다음과 같이 정리된다.




▲ WNDCLASS를 공유하는 HINSTANCE와 HWND

WNDCLASS 정보는 HINSTANCE에 저장되고, 윈도우 클래스 이름만 HWND에 저장된다고 기억하면 쉽다. 이는 WNDCLASS는 여러 개 인스턴스에서 사용할 수 있고, 여러 개 윈도우에서 공용으로 사용할 수 있다는 의미다.

실제로 WNDCLASS를 공유하는 것을 예제를 통해 확인해 보자. 비주얼 C++ 프로젝트에서 IDD_ABOUTBOX 대화 상자 템플릿을 열어 보자. 그리고 대화 상자에 있는 〈확인〉 버튼을 스파이를 통해 확인하면 다음과 같은 클래스 정보를 얻을 수 있다. 단, 스파이 이후 실행한 윈도우는 F5 키를 사용하거나 [창(W)]-[새로 고침(R)] 메뉴를 선택하여 스파이를 갱신해야 한다.

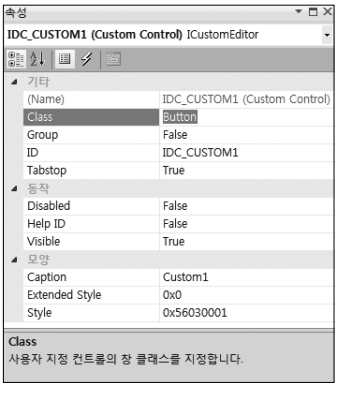



▲ 버튼 윈도우 클래스의 이름 확인하기

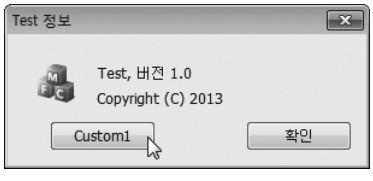
'클래스 이름' 항목이 Button인 것은 버튼 컨트롤을 의미한다. 이를 통해 대화 상자 템플릿 편집기도 실제 컨트롤을 사용하는 것을 확인할 수 있다.

이제 도구 상자에서 사용자 컨트롤(Custom Control, )을 대화 상자 템플릿에 드래그 & 드롭한다. 그리고 그 사용자 컨트롤의 속성을 보면 다음과 같이 구성된다.

그 중 'Class' 항목이 공란으로 되어 있다. 여기에 윈도우 클래스 이름으로 "Button"을 입력하고, 'Style' 항목은 스파이의 [스타일] 탭에 있는 값을 참고하여 "0x56030001"을 입력한다. 이제 실행해 보면 사용자 컨트롤은 마치 버튼 컨트롤을 사용한 것처럼 동작할 것이다.




▲ 사용자 컨트롤의 속성 창




▲ 사용자 컨트롤이 버튼 컨트롤로 동작하는 화면

이처럼 WNDCLASS는 공유할 수 있으며, 실제 표준 컨트롤과 공통 컨트롤도 WNDCLASS를 등록해서 공유한다. 이 때문에 컨트롤을 사용할 때는 WNDCLASS를 등록할 필요가 없다. 게다가 누구나 같은 방식으로 WNDCLASS를 직접 등록하여 컨트롤을 제작하고 공유할 수 있다. WNDCLASS가 이렇게 공용으로 사용될 수 있어 여기에 정의된 윈도우 프로시저 함수도 필요에 따라 분기할 수 있도록 HWND가 제공된다. 정리하면 다음과 같다.

"윈도우 프로그램도 콘솔 프로그램처럼 HINSTANCE를 갖는다. 그러나 윈도우 프로그램은 윈도우 외형과 동작 등을 지정하기 위해 WNDCLASS를 사용하고, 이것은 같은 윈도우 특성을 공유할 수 있도록 설계되어 있다. 그리고 윈도우를 구분하기 위해 HINSTANCE와 WNDCLASS를 조합한 HWND 개념이 소개되었다."