

WEEK 1

DAY 2

애플리케이션에서 컨트롤 사용하기

다는 아니겠지만 대부분의 윈도우 애플리케이션에서 쉽게 찾을 수 있는 것들을 꼽아보라면 버튼, 체크 박스, 텍스트 박스, 드롭다운 리스트 박스 등이 생각날 것이다. 이런 것들은 컨트롤(Control)이라고 알려져 있으며, 이들 컨트롤은 이미 운영체제 차원에서 만들어진 것들이며, 윈도우 애플리케이션에서 컨트롤 사용하기는 비주얼 C++를 통하여 아주 쉬운 일이다. 다음에 나온 것들을 차근히 공부해서 여러분 것으로 만드는 것을 이번 장의 목표로 삼자.

- ❖ 비주얼 C++에서 사용하는 기본적인 컨트롤에 대해
- ❖ 컨트롤에 변수를 선언해서 물려두는 방법

- ❖ 컨트롤과 물려진 변수 사이에서 값을 교환하는 방법
- ❖ 애플리케이션 윈도우 내에서 컨트롤들의 탭 순서를 설정하는 방법
- ❖ 컨트롤에 대해 동작 설정하기
- ❖ 애플리케이션이 실행되는 도중에 컨트롤의 외양을 조작하고 변경하기

기본적인 윈도우 컨트롤

표준 컨트롤 몇 가지는 슬라이더, 트리 컨트롤과 리스트 컨트롤, 프로그레스바와 함께 윈도우 운영체제에 내장되어 있다. 하지만 이번 장에서는 거의 모든 윈도우 애플리케이션에서 사용되는 핵심 컨트롤만 가지고 공부해 보도록 하겠다.

- ❖ 정적 텍스트(Static box)
- ❖ 에디트 박스(Edit box)
- ❖ 푸시 버튼(Push button, 명령 버튼이라고도 한다)
- ❖ 체크 박스(Check box)
- ❖ 라디오 버튼(Radio button)
- ❖ 드롭다운 리스트 박스(Drop-down list box : 콤보 박스로도 알려져 있다)

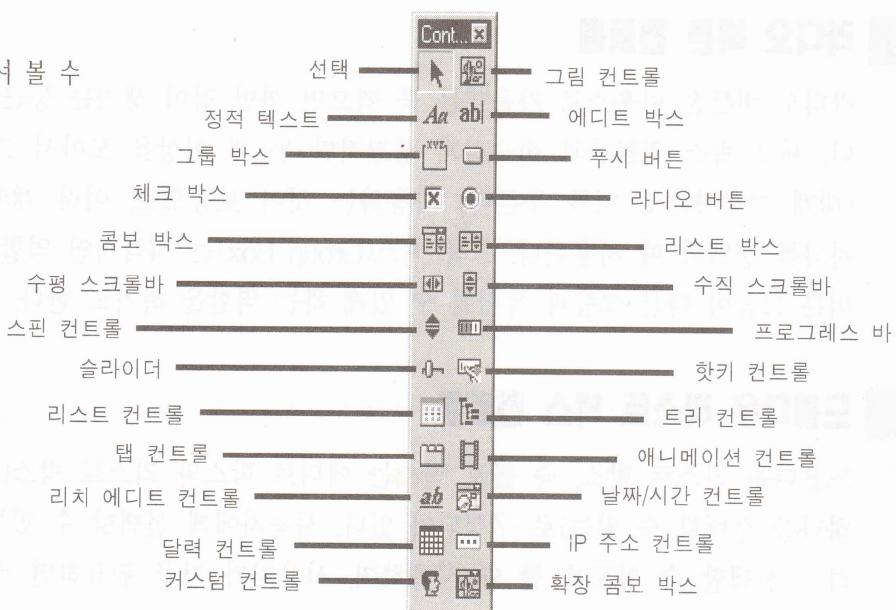
위 나열한 것을 포함해서, 윈도우 운영체제에서 지원하는 표준 컨트롤은 모두 비주얼 C++ 애플리케이션에서 사용할 수 있으며, 디벨로퍼 스튜디오의 다이얼로그 에디터와 같이 나오는 Controls 도구바에서([그림 2.1] 참조) 찾을 수 있을 것이다.

정적 텍스트 컨트롤

정적 텍스트 컨트롤은 사용자에게 단순히 텍스트를 보여주기 위해 사용하며, 실행 도중에 사용자가 텍스트를 직접 바꾼다면 하는 일은 할 수 없다. 단지 읽기 위한 용도이지만, 준비된 윈도우용 함수를 사용하면 텍스트를 바꿀 수도 있다.

그림 2.1 →

Controls 도구바에서 볼 수 있는 표준 컨트롤



■ 에디트 박스 컨트롤

에디트 박스는 사용자로 하여금 텍스트를 입력하거나 바꿀 수 있도록 하는 컨트롤이며, 사용자로부터 어떠한 정보를 얻어내기 위해 주로 쓰이는 것이기도 하다. 사용자에게 주어진 만큼의 텍스트를 입력하게 한 다음, 이것을 잡아내어 원하는 목적에 사용하면 된다. 에디트 박스는 단순한 텍스트만을 받아들일 수 있으며, 서식을 준다든지 하는 것은 불가능하다.

■ 푸시 버튼 컨트롤

푸시 버튼은 사용자가 “눌러서” 어떤 동작이 일어나게 하는 컨트롤을 말한다. 푸시 버튼의 가운데에는 “이 버튼이 눌리면 어떤 역할을 합니다”를 알려주는 레이블(캡션)이 달려 있다. 푸시 버튼은 이미지도 가질 수 있어서 좀더 사용자에게 시각적인 정보를 전하는데 도움이 된다.

■ 체크 박스 컨트롤

체크 박스는 네모와 레이블을 가지고 있는 형식이며, 사용자는 네모를 클릭해서 체크 표시를 없애거나 놓거나 할 수 있다. 특정한 기능을 끄거나 켜 때 사용하며, 기본적으로 “그렇다/아니다”的 두 가지 상태만 가지고 있으나 경우에 따라서는 “그럴걸?”이란 중간 상태도 있을 수 있다. 보통 체크 박스를 사용할 때는 그렇다/아니다 타입을 쓰게 된다.

■ 라디오 버튼 컨트롤

라디오 버튼은 마우스로 가운데를 콕 찍으면 까만 점이 생기는 등근 원으로 되어 있다. 체크 박스 컨트롤과 비슷하게 생겼지만 두 개 이상을 모아서 그룹을 지은 다음 (대개 그룹박스로 띠를 두른다) 사용하는 것이 보통으로, 여러 개의 선택 사항 중 하나를 선택할 때 사용한다. 그룹 박스(Group Box)는 시각적인 역할 외에 각 라디오 버튼 그룹이 다른 그룹과 독립될 수 있게 하는 역할을 하기도 한다.

■ 드롭다운 리스트 박스 컨트롤

드롭다운 리스트 박스, 즉 콤보 박스는 에디트 박스와 리스트 박스(여러 개의 값 중 하나를 선택할 수 있는)로 구성되어 있다. 사용자에게 선택할 수 있는 리스트를 제시하고 선택할 수 있도록 할 때 사용하며, 사용자가 가끔 필요하면 직접 입력하게 할 수도 있다.

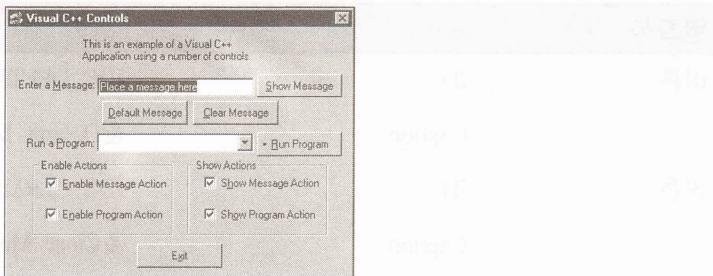
윈도우에 컨트롤 추가하기

이번 장에서 만들어 볼 애플리케이션은 하나의 다이얼로그 윈도우에 방금 설명한 컨트롤을 덕지덕지 달아본 후 각 기능을 시험해 보는 애플리케이션이다([그림 2.2] 참조). 이 윈도우의 위쪽에는 에디트 박스가 있으며, 여기에 써넣은 텍스트를 메시지 박스로 표시하게 하는 버튼이 바로 옆에 붙어 있다. 에디트 박스의 아래에는 버튼이 두 개 있는데, 하나는 에디트 박스를 디폴트 메시지로 채우며, 또 하나는 에디트 박스를 지운다. 이를 버튼 아래에는 드롭다운 리스트 박스가 있는데, 표준 윈도우 프로그램의 리스트가 나열되어 있다. 사용자가 이들 중 하나를 선택해서 드롭다운 리스트 박스의 옆에 있는 버튼을 누르면 선택된 프로그램이 실행될 것이다. 다음에 있는 두 개의 체크 박스 그룹은 다이얼로그에 추가한 컨트롤에 영향을 주기 위해 붙였는데, 사용자 메시지를 표시하는 컨트롤과 프로그램을 실행하는 컨트롤을 맡고 있다.

왼쪽에 있는 체크 박스는 각 컨트롤 그룹을 활성화하거나 비활성화시키며, 오른쪽에 있는 체크 박스는 각 컨트롤 그룹을 보이거나 숨긴다. 마지막으로, 다이얼로그 윈도우의 최하단에 있는 버튼은 애플리케이션을 끝내는 구실을 한다.

그림 2.2 →

표준 컨트롤을 달고 있는 이번 장의 애플리케이션



■ 애플리케이션 골격을 만들고ダイ얼로그 박스를 꾸미기

1장에서 배운 것을 그대로 사용하면, 애플리케이션 골격을 만들고ダイ얼로그 박스를 꾸미는 일은 그리 어렵지 않을 것이다.

1. 새 프로젝트의 이름을 Day2로 하고 애플리케이션 위저드를 시작하자.
2. 1장에서 해주었던 똑같은 설정을 사용하자.ダイ얼로그 윈도우의 타이틀은 Visual C++ Controls로 정한다.
3. 애플리케이션 골격을 만들었으면, 이제 메인ダイ얼로그 박스를 [그림 2.2]에 나온 대로 꾸미자.
4. [표 2.1]을 참고해서 각 컨트롤의 프로퍼티를 설정한다.

*ID C
control*

[표 2.1] 애플리케이션ダイ얼로그에 놓을 컨트롤들의 프로퍼티 설정

컨트롤	ID	설정
정적 텍스트	IDC_STATIC	
	Caption	This is an example of a Visual C++ Application using a number of controls.
정적 텍스트	IDC_STATICMSG	
	Caption	Enter a & Message:
정적 텍스트	IDC_STATICPGM	
	Caption	Run a & Program:
에디트 박스	IDC_MSG	
버튼	IDC_SHWMSG	
	Caption	& Show Message

컨트롤	ID	설정
버튼	ID	IDC_DFLTMSG
	Caption	&Default Message
버튼	ID	IDC_CLRMSG
	Caption	&Clear Message
버튼	ID	IDC_RUNPGM
	Caption	&Run Program
버튼	ID	IDC_EXIT
	Caption	E&xit
콤보 박스	ID	IDC_PROGTORUN
그룹 박스	ID	IDC_STATIC
	Caption	Enable Actions
그룹 박스	ID	IDC_STATIC
	Caption	Show Actions
체크 박스	ID	IDC_CKENBLMSG
	Caption	&Enable Message Action
체크 박스	ID	IDC_CKENBLPGM
	Caption	E&nable Program Action
체크 박스	ID	IDC_CKSHWMSG
	Caption	S&how Message Action
체크 박스	ID	IDC_CKSHWPGM
	Caption	Sh&ow Program Action



콤보 박스 컨트롤을 윈도우에 놓을 때 중요한 점은 드롭다운 리스트가 펼쳐질 크기까지 감안해서 크기 조정을 해두어야 한다는 점이다. 일단 윈도우 위에 컨트롤을 놓은 후 컨트롤의 폭을 조절해서 평상시의 크기를 맞추고, 드롭다운 화살표를 눌렸을 때 펼쳐질 크기까지 조절해 줄 수 있을 것이다.

5. 다이얼로그 윈도우에 컨트롤들을 모두 놓고 프로퍼티를 조정한 후, 콤보 박스에 대해 프로퍼티 다이얼로그 박스를 다시 열도록 하자. Data 탭을 선택하고 다음의 값들을 입력하는데, 한 개 입력한 후마다 Ctrl + Enter를 눌러서 아래 칸으로 진행하도록 하자. 완성된 후의 모습은 [그림 2.3]과 같다.

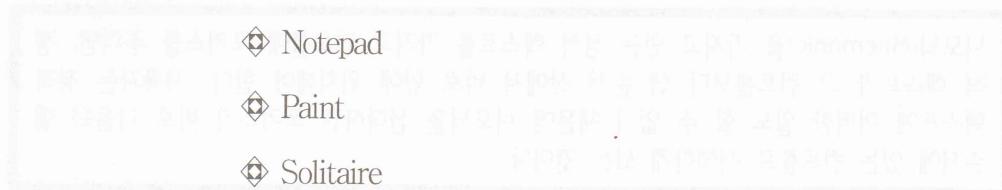
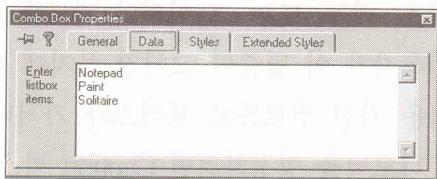


그림 2.3 →

콤보 박스의 드롭다운
리스트에 엔트리를 넣기 위해
프로퍼티 다이얼로그의 Data
탭을 사용하고 있다.



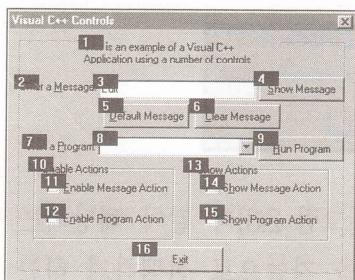
■ 컨트롤의 탭 순서 설정하기

윈도우에 컨트롤들을 모두 놓은 후에는 탭 순서(Tab order)라는 것을 설정해줄 수 있다. 탭 순서란 사용자가 Tab 키를 눌렀을 때 각 컨트롤에 포커스가 위치하는 순서를 말한다. 다음의 단계를 따라 탭 순서를 설정해 보기로 하자.

1. 디벨로퍼 스튜디오의 에디터 영역에서 다이얼로그 윈도우 또는 원도우 상의 아무 컨트롤이나 선택한다.
2. 메뉴에서 Layout | Tab Order를 선택한다. 탭 순서 설정 모드에 들어가면 윈도우상의 각 컨트롤 옆에 번호가 붙어있는 것이다. 이 번호가 바로 Tab 키를 눌렀을 때 포커스가 이동하는 순서이다. [그림 2.4]를 보도록 하자.

그림 2.4 →

Tab Order 기능을
작동시키면 다이얼로그 내의
컨트롤에 포커스가 주어지는
순서가 나타난다.



3. 마우스를 사용해서, Tab 키를 눌렀을 때 포커스가 이동했으면 하는 순서로 각 번호를 클릭해 가자. 컨트롤에 붙은 번호가 여러분이 클릭한 순서대로 변할 것이다.
4. 탭 순서 설정이 끝났으면 Layout | Tab Order를 다시 선택해서 보통의 다이얼로그 에디터로 돌아온다.

Note

니모닉(Mnemonic)을 가지고 있는 정적 텍스트를 가지고 컨트롤에 포커스를 주려면, 정적 텍스트가 그 컨트롤보다 템 순서 상에서 바로 앞에 위치해야 한다. 사용자는 정적 텍스트에 어떠한 일도 할 수 없기 때문에 니모닉을 선택하면 포커스가 바로 다음의 템 순서에 있는 컨트롤로 직행하게 되는 것이다.

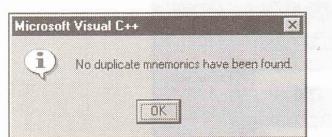
니모닉은 버튼, 체크 박스, 메뉴, 또는 그 이외의 컨트롤 레이블(캡션)에서 밑줄이 그어진 문자를 말한다. 사용자가 이 밑줄이 그어진 문자에 해당하는 키와 Alt키를 동시에 누르면 해당 니모닉을 가진 컨트롤로 포커스가 가거나 컨트롤을 클릭한 것과 똑같은 효과가 발생한다. 니모닉을 설정하려면 Caption 박스에 써진 캡션에서 니모닉으로 삼을 문자 앞에 &를 붙여주면 된다. 이때 주의할 것이 있는데, 한 윈도우 또는 메뉴 안에서 중복된 니모닉은 설정하지 않도록 하자. 사용자가 혼동할 수도 있기 때문이다.

이제 애플리케이션의 코드를 작성하기까지 딱 한 가지만 더 하면 된다. 바로 니모닉을 점검해서 중복된 것이 없는지를 확인해 보는 것이다. 다음의 단계를 따르자.

1. 디벨로퍼 스튜디오의 에디터 영역에서 다이얼로그 윈도우 또는 윈도우 상의 아무 컨트롤이나 선택한 다음 오른쪽 클릭해서 팝업 메뉴를 띄우고 Check Mnemonic을 선택한다.
2. 설정된 니모닉들간의 충돌이 없으면 비주얼 C++는 “충돌이 없네요”란 메시지를 메시지 박스를 통해서 내보낸다([그림 2.5] 참조).

그림 2.5 →

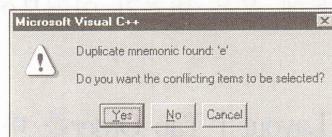
니모닉 체커는 니모닉 간의 충돌이 있는지 점검한다.



3. 충돌이 있을 경우에는 [그림 2.6]과 같이 해당되는 문자와 함께 충돌된 니모닉을 가지고 있는 컨트롤을 자동으로 선택하게 하는 옵션이 메시지 박스와 함께 제시된다.

그림 2.6 →

중복된 니모닉은 자동으로 선택된다.



컨트롤에 변수를 물려두기

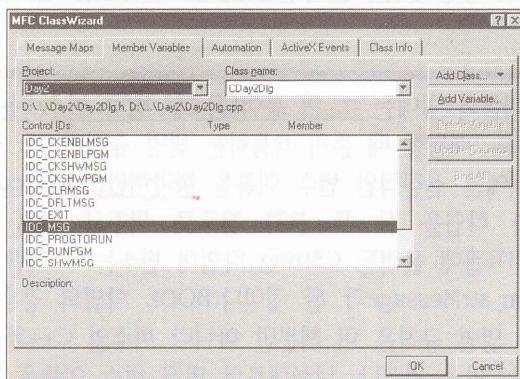
이 시점에서, 만약 여러분이 비주얼 베이직이나 파워빌더로 프로그래밍하고 있었다면 “이제 코드를 [결] 준비가 다 되었구나.”라고 생각할지 모르겠지만 비주얼 C++의 경우는 한 단계가 더 남아있다(결코 나쁘단 뜻이 아니다). 코드를 작성하기 전에, 텍스트 컨트롤과 푸시 버튼을 제외한 모든 컨트롤 값을 가질 필요가 있는 변수를 선언해서 물려 두어야 하는 것이다. 왜냐하면, 실제 코딩 시에는 컨트롤을 사용할 때 물려 둔 변수를 사용하기 때문이다. 사용자가 화면상의 컨트롤을 안에다가 써둔 값은 애플리케이션 코드 내에서는 해당 컨트롤에 물려둔 변수를 통해 알아내고 이용하는 것이다. 그 반대의 경우도 가능하다. 즉, 애플리케이션 코드 내에서 설정한 변수의 값은 그 변수가 물고 있는 컨트롤로 전송시켜 사용자에게 보일 수도 있다.

그렇다면, 어떻게 컨트롤의 값을 나타내는 변수를 선언하고 물려둘 수 있을까? 간단하다. 이것 때문에 이 책이 있는 것 아니겠는가?

1. 1장에서 배운 대로 클래스위저드를 연다.
2. [그림 2.7]과 같이 Member Variables 탭을 선택한다.

그림 2.7 →

클래스위저드의 Member Variables 탭은 여러분이 컨트롤에 변수를 물려두는 곳이다.

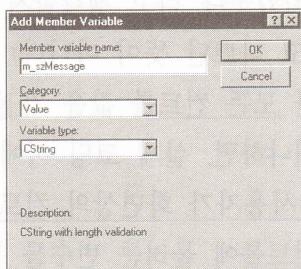


3. 변수를 물려둘 컨트롤 중 하나의 ID(IDC_MSG와 같은)를 선택한다.
4. Add Variable 버튼을 클릭한다.
5. Add Member Variableダイ얼로그 박스에서 [그림 2.8]과 같이 변수 이름과 카테고리(Category), 그리고 변수 타입(Variable type)을 설정한 다음 OK를 클릭 한다.

6. 변수를 물려줄 필요가 있는 다른 모든 컨트롤들에 대해 단계 3과 단계 5를 반복하자. 아직까지 열렬열해 있다면 [표 2.2]를 참고하자.

그림 2.8 ➔

변수를 컨트롤에
물려주는 순간



[표 2.2] 애플리케이션의 컨트롤에 물려줄 변수

컨트롤	이름	카테고리	타입
IDC_MSG	m_strMessage	Value	CString
IDC_PROGTORUN	m_strProgToRun	Value	CString
IDC_CKENBLMSG	m_bEnableMsg	Value	BOOL
IDC_CKENBLPGM	m_bEnablePgm	Value	BOOL
IDC_CKSHWMSG	m_bShowMsg	Value	BOOL
IDC_CKSHWPGM	m_bShowPgm	Value	BOOL

Tip

모든 변수 이름의 앞에는 m_이 붙어있는데, 멤버 변수라는 것을 나타내기 위해서이다. MFC에서 프로그래밍할 때 흔히 사용하는 명명 규칙이기도 하다. 이외에 조금 더 설명하자면, m_ 뒤에는 실질적인 변수 이름을 헝가리안(Hungarian) 표기를 사용해서 짓는데, 먼저 변수의 타입을 한 두 문자 정도로 써주고 나서 이름을 붙여준다. 이를테면 IDC_MSG 컨트롤에 물려둔 CString 타입의 변수는 str이란 타입 명칭과 Message란 이름이 붙어 m_strMessage가 된 것이다(BOOL 타입의 경우에는 str 대신 b가 붙는다). 이러한 변수 명명 규칙은 이 책뿐만 아니라 비주얼 C++와 MFC를 다룬 다른 책에서도 술술하게 볼 수 있을 것이다. 나름대로의 멋진 변수 이름을 지을 수 있는 철학이 없으면 이 규칙을 따르는 것이 다른 사람으로 하여금 자기 코드를 읽기 쉽게 할 것이다. 반대로 다른 사람의 코드를 읽을 때도 쉽게 읽을 수 있다는 이점도 있을 것이다.

7. 필요한 모든 변수를 추가했으면 OK를 클릭해서 클래스위저드를 닫는다.

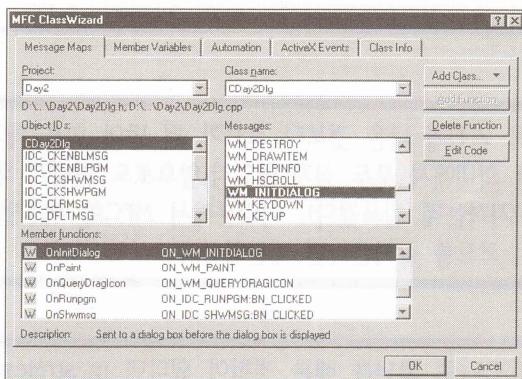
컨트롤에 함수를 연결하기

여러분의 애플리케이션 윈도우가 가지고 있는 컨트롤에 코드를 연결하기 전에 할 일이 또 하나 있는데(미안하다 흑흑), 컨트롤에 물려둔 변수들의 값을 초기화하는 코드를 넣어주어야 한다. 경우에 따라선 필요할 수 있으니까 따라해 주기 바란다.

1. 클래스워저드에서 Messages Maps 탭을 선택하고, Messages 리스트에서 OnInitDialog 함수를 찾아서 선택한다. 선택된 함수는 Member Functions 리스트 박스에 추가될 것이다. 아니면 Object IDs 리스트에서 CDay2Dlg를 선택하고 Messages 리스트에서 WM_INITDIALOG를 선택해도 된다([그림 2.9] 참조).

그림 2.9 →

클래스워저드는 이미 있는 함수를 찾아 추가하는데도 사용할 수 있다.



2. Edit Code 버튼을 클릭해서 OnInitDialog 함수의 소스 코드로 들어가자.
3. TODO 주석문을 찾은 다음 [리스트 2.1]에 나온 코드 부분을 끼워 넣는다.

리스트 2.1 초기화 코드를 넣는데 적당한 OnInitDialog 함수

```

1: BOOL CDay2Dlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:
6:
7:
8:
9:     // TODO: Add extra initialization here
10:
11:    /////////////////////
12:    // 새로 넣을 코드가 여기서부터 시작된다
13:    ///////////////////

```

```

15: // 디폴트 메시지를 설정한다
16: m_strMessage = "Place a message here.";
17:
18: // 모든 체크 박스에 체크해 둔다
19: m_bShowMsg = TRUE;
20: m_bShowPgm = TRUE;
21: m_bEnableMsg = TRUE;
22: m_bEnablePgm = TRUE;
23:
24: // 파일로그의 컨트롤에 값을 넣는다
25: UpdateData(FALSE);
26:
27: /////////////
28: // 새로 넣을 코드는 여기서 끝난다
29: /////////////
30:
31: return TRUE; // return TRUE unless you set the focus to a control
32: }

```

Note

아마 (리스트 2.1)에 나온 것보다 코드가 더 많이 있을 테지만 수정하고 추가할 코드에만 집중하란 의미에서 모두 싣지 않았다(앞으로도 이렇게 해서 책의 부피를 줄이고 국민 경제에 이바지하는데 힘쓰겠다). 여러분께서 MFC와 비주얼 C++에 대한 이해를 높이기 위해 나머지 코드를 살펴본다면 대환영이다.

Tip

이전에 C나 C++로 코딩을 해본 경험이 있다면 `m_strMessage` 변수에 값을 넣는 과정이 참 이상하게 느껴졌을 것이다. 마치 비주얼 베이직나 파워빌더에서 문자열 타입 변수에 값을 넣는 것과 비슷하지 않은가? 이것은 이 변수가 `CString` 타입의 변수이기 때문이다. `CString` 클래스는 다른 프로그래밍 언어에서 경험했었던 편리한 문자열 처리를 할 수 있도록 도와주는 클래스이다. 그래도 엄연히 C++ 언어를 사용하기 때문에 각 문장의 끝에 세미콜론(;)을 꼭 붙여주어야 한다.

하여간 초기화 코드는 꽤 간단해 보인다. 에디트 박스를 통해 사용자에게 보일 초기 메시지를 설정하였고, 모든 체크 박스에 체크하였다. 그리고 마지막 줄이 조금 주목할 필요가 있다.

UpdateData() 함수는 비주얼 C++에서 컨트롤에 물려둔 변수를 다루는 열쇠이다. 이 함수는 컨트롤에 물려둔 변수에 들어있는 값을 화면상의 컨트롤로 보내어 컨트롤을 갱신하는 역할 또는 컨트롤에 있는 값을 이 컨트롤에 물려둔 변수로 보내어 변수를 갱신하는 역할을 맡고 있다. 이 두 가지는 `UpdateData()`에 들어가는 `BOOL` 타입의 인수로 결정되는데, FALSE일 경우에는 변수에 있는 값이 컨트롤로 보내어지며 TRUE일 경우에는 컨트롤에 있는 값이 변수로 보내어진다. 따라서, 이 함수에 전달

되는 값이 갱신시키고자 하는 방향을 좌우한다. 한 개 이상의 변수 값을 새로 설정한 후 컨트롤로 보내고 싶으면 UpdateData() 함수에 FALSE 인수를 넘겨서 호출하고, 컨트롤의 현재 값을 읽어내고 싶으면 TRUE 인수를 넘기면 된다. 처음에 모두 이해할 필요는 없고 애플리케이션에 코드를 좀 더 추가하다 보면 자연히 몸에 배일 것이다.

■ 애플리케이션 종료하기

뭐니뭐니해도 첫번째로 살펴보아야 할 일은 사용자가 여러분의 프로그램을 종료시킬 수 있는가를 점검하는 것이다. OK와 Cancel 버튼을 지우고 애플리케이션을 종료시키는 용도의 다른 버튼(Exit)을 추가하였기 때문에, Exit 버튼을 눌렀을 때 호출되는 함수에다가 애플리케이션을 종료시키는 코드를 넣어 두어야 한다. 다음의 단계를 따르자.

1. 클래스위저드를 사용해서 IDC_EXIT 컨트롤의 BN_CLICKED 메시지에 대한 함수를 추가한다. 방법은 이미 1장에서 다루었다.
2. Edit Code 버튼을 클릭해서 방금 추가한 함수의 몸체로 진행하자.
3. [리스트 2.2]에 나온 코드를 입력하자.

리스트 2.2 OnExit 함수

```

1: void CDay2Dlg::OnExit()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 프로그램을 종료한다.
10:    OnOK();
11:
12:    /////////////////
13:    // 새로 넣을 코드는 여기서 끝난다
14:    ///////////////
15:
```

여기에 추가한 단 한 줄의 코드로 인해 윈도우가 닫히고 애플리케이션이 종료된다. 그런데 대체 OnOK() 함수는 어디서 왔으며, 1장에서는 이 중요한 함수를 호출하지

않았을까? `OnOK()`와 `OnCancel()` 함수는 `CDay2Dlg` 클래스가 상속한 `CDialog` 클래스에서 이미 가지고 있는 함수이다. `CDialog` 클래스안의 메시지 맵이 이미 `OK`와 `Cancel` 버튼에 대한 `ID(IDOK, IDCANCEL)`와 `OnOK()`, `OnCancel()`을 연결시키는 엔트리를 가지고 있기 때문에 이 ID를 가지고 있는 버튼이 다이얼로그 윈도우 위에 있으면 자동으로 해당 함수를 호출하게 되는 것이다. 만약 `Exit` 버튼의 ID를 `IDOK`로 만들었다면, 기본 클래스의 `OnOK()` 함수를 재정의하지 않는 이상, 함수를 추가할 필요가 없었을 것이다.

사용자에게 메시지 띄우기

사용자가 에디트 박스에 써넣은 텍스트를 메시지 박스를 통해 띄우는 일은 매우 간단하다. 이미 비슷한 일은 1장에서 해 보았기 때문이다. `Show Message` 버튼에 대한 함수를 추가한 다음, [리스트 2.3]과 같이 `MessageBox` 함수를 호출하도록 하자.

리스트 2.3 사용자 메시지를 표시하는 `OnShwmsg` 함수

```

1: void CDay2Dlg::OnShwmsg()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     ///////////////////
8:
9:     // 사용자에게 메시지를 출력한다.
10:    MessageBox(m_strMessage);
11:
12:    ///////////////////
13:    // 새로 넣을 코드는 여기서 끝난다
14:    ///////////////////
15:
```

여기서 바로 애플리케이션을 컴파일하고 실행시키면 뭔가 잘못되어 간다는 느낌을 지울 수 없다. 에디트 박스에 입력한 텍스트와 전혀 무관한 메시지가 출력되는 것이다. 즉, `OnInitDialog()` 함수에서 초기화된 `m_strMessage` 값이 사용되었다. 왜 그런고 하니, 윈도우 상의 컨트롤이 가진 내용을 아직 컨트롤에 물려둔 변수로 보내지 않았기 때문이다. 따라서 `TRUE` 값을 인수로 해서 `UpdateData()`를 호출한 후에 `MessageBox()`를 써서 메시지를 띄워야 한다. `OnShwmsg()` 함수를 [리스트 2.4]와 같이 바꾸도록 하자.

리스트 2.4 바뀐 OnShwmsg() 함수

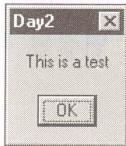
```

1: void CDay2Dlg::OnShwmsg()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 사용자가 입력한 텍스트로 메시지 변수를 갱신한다
10:    UpdateData(TRUE);
11:
12:    // 메시지를 띄운다.
13:    MessageBox(m_strMessage);
14:
15:    /////////////////
16:    // 새로 넣을 코드는 여기서 끝난다
17:    /////////////////
18: }
```

이제 애플리케이션을 컴파일하고 실행시키면, 에디트 박스에 입력한 텍스트를 메시지 박스를 통해 확인할 수 있을 것이다([그림 2.10] 참조).

그림 2.10 →

에디트 박스에 입력한
메시지가 사용자에게
띄워진다.



■ 사용자가 입력한 메시지를 지우기

사용자가 에디트 박스에 텍스트를 써넣기 전에 무엇인가가 지저분하게 써있다면 좋아하지 않는 분들도 분명히 있을 것이다. 그래서 준비한 버튼이 Clear Message 버튼으로, 에디트 박스의 내용을 지우는 역할을 한다. 이 버튼에 대해 함수를 추가하는 일은 언제나처럼 클래스위저드를 사용하고, 코딩 또한 단순하다. m_strMessage에 빈 문자열을 대입한 다음 컨트롤에 보내면 그만인 것이다. [리스트 2.5]를 보도록 하자.

리스트 2.5 OnClrmsg() 함수

```

1: void CDay2Dlg::OnClrmsg()
2: {
3:     // TODO: Add your control notification handler code here
4:
```

```

5:   /////////////////
6:   // 새로 넣을 코드가 여기서부터 시작된다
7:   ///////////////////
8:
9:   // 메시지를 지운다
10:  m_strMessage = "";
11:
12:  // 화면을 갱신한다.
13:  UpdateData(FALSE);
14:
15:  /////////////////
16:  // 새로 넣을 코드는 여기서 끝난다
17:  ///////////////////
18: }

```

■ 메시지 컨트롤을 비활성화하고 감추기

메시지 표시에 관련된 컨트롤에 마지막으로 해줄 일은 Enable Message Action과 Show Message Action 체크 박스에 원하는 동작을 연결하는 것이다. 첫번째 체크 박스는 사용자 메시지의 표시에 관련된 컨트롤을 활성화시키거나 비활성화시키는데 사용한다. 이 체크 박스가 체크된 상태에 있을 때는 관련된 컨트롤이 모두 활성화되지만, 체크가 되지 않은 상태에 있을 때는 관련된 컨트롤이 모두 비활성화된다. 두번째 체크 박스도 비슷한 동작을 취하며, 단지 메시지 표시에 관련된 컨트롤을 보이거나 숨기는 것만 다르다. 이 두 개의 체크 박스에 연결된 각각의 함수 코드를 [리스트 2.6]에 실었다.

리스트 2.6

Enable Message Actions 체크 박스와
Show Message Actions 체크 박스에 대한 함수

```

1: void CDay2Dlg::OnCkenablemsg()
2: {
3:   // TODO: Add your control notification handler code here
4:
5:   /////////////////
6:   // 새로 넣을 코드가 여기서부터 시작된다
7:   ///////////////////
8:
9:   // 화면상의 데이터를 읽어서 변수로 보낸다. - 여백
10:  UpdateData(TRUE);
11:
12:  // Enable Message Action 체크 박스가 체크된 상태인가?
13:  if (m_bEnableMsg == TRUE)
14:  {
15:    // 그렇소. 그러니 사용자 메시지 출력에 관계된
16:    // 컨트롤을 모두 활성화시키시지.

```

```

17:     GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
18:     GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
19:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
20:     GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
21:     GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
22: }
23: else
24: {
25:     // 아니오. 그러니 사용자 메시지 출력에 관계된
26:     // 컨트롤을 모두 비활성화시키시지.
27:     GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
28:     GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
29:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
30:     GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
31:     GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
32: }
33: ///////////////////////////////
34: // 새로 넣을 코드는 여기서 끝난다
35: //////////////////////////////
36: //////////////////////////////
37: }
38:
39: void CDay2Dlg::OnCkshwmsg()
40: {
41:     // TODO: Add your control notification handler code here
42:     //////////////////////////////
43:     // 새로 넣을 코드가 여기서부터 시작된다
44:     //////////////////////////////
45:     //////////////////////////////
46:     // 화면상의 데이터를 읽어서 변수로 보낸다.
47:     UpdateData(TRUE);
48:
49:     // Show Message Action 체크 박스가 체크된 상태인가?
50:     if (m_bShowMsg == TRUE)
51:     {
52:         // 그렇소. 그러니 사용자 메시지 출력에 관계된
53:         // 컨트롤을 모두 보이시지.
54:         GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
55:         GetDlgItem(IDC_SHWMSG)->ShowWindow(TRUE);
56:         GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
57:         GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
58:         GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
59:     }
60:
61:     else
62:     {
63:         // 아니오. 그러니 사용자 메시지 출력에 관계된
64:         // 컨트롤을 모두 숨기시지.
65:         GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
}

```

```

66:     GetDlgItem(IDC_SHWMSG)->ShowWindow(FALSE);
67:     GetDlgItem(IDC_DFLTMSG)->ShowWindow(FALSE);
68:     GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
69:     GetDlgItem(IDC_STATICMSG)->ShowWindow(FALSE);
70: }
71: /////////////////////
72: // 새로 넣을 코드는 여기서 끝난다
73: /////////////////////
74: /////////////////////
75: }

```

이 함수들의 쉬운 부분부터 살펴보기로 하자. 우선, 윈도우 상의 컨트롤이 현재 가지고 있는 값을 읽어서 컨트롤에 물려둔 변수로 보냈다. 다음, 해당 체크 박스에 대한 변수가 불리언 값을 검사해서 TRUE이면 컨트롤을 활성화시키거나 보이고, FALSE이면 컨트롤을 비활성화시키거나 숨기는 부분이 이어지는 것 같다.

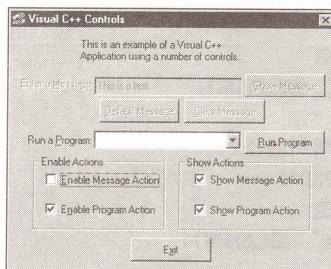
이제부터 어려운 코드가 나오기 시작한다. `GetDlgItem()` 함수가 첫번째 주자로서, 바꾸고자 하는 컨트롤의 ID를 받아 그 컨트롤에 해당하는 포인터(`CWnd*` 타입)를 반환한다. 여러분은 애플리케이션이 수행되는 도중에 이 함수를 사용해서 윈도우 상의 어떤 컨트롤이라도 얻어낼 수가 있다. 자, 이제 컨트롤의 개체 포인터를 얻어낸 후에 그 개체의 멤버 함수를 호출하고 있다. 즉, 앞의 문장에서 두번째 함수는 첫번째 함수에서 반환된 개체의 멤버 함수인 것이다. 알동 말동 하거나 아예 모르겠다고 생각하시는 분은 부록 A, “C++도 다시 보자”를 읽고 오기 바란다.

아무튼 “두번째” 함수란 것을 보기로 하자. `EnableWindow()`와 `ShowWindow()`는 윈도우 차원에서 사용되는 함수이지, 컨트롤 차원에서 사용되는 함수는 아닌 것 같다. 음, 사실 그렇다. 아주 먼 옛날, 이 함수는 어쩌다가 `CDialog` 클래스(`CDay2Dlg` 클래스가 상속한)의 기본 클래스인 `CWnd`(윈도우의 조작을 담당하는 클래스)의 멤버 함수로 들어가게 되었다. 윈도우 운영체제에서 모든 컨트롤은 그 자체가 윈도우이기 때문에, 자신이 놓인 윈도우와 독립적인 윈도우로 취급되어 동작할 수 있다. 따라서 윈도우용 함수(`CWnd`의 멤버 함수)를 호출해도 상관이 없다. 사실 모든 컨트롤 클래스는 `CWnd`에서 상속받은 것이기 때문에 윈도우의 성질을 그대로 가지고 있다.

이제 애플리케이션을 컴파일하고 실행한 다음 Enable Message Action 체크 박스와 Show Message Action 체크 박스에 클릭해서 시험해 보자. [그림 2.11]처럼 제대로 돌아가는지 확인하자.

그림 2.11 →

사용자 메시지 출력에
관련된 컨트롤은 현재
비활성화되었다.



■ 다른 애플리케이션 실행하기

여러분의 애플리케이션에서 구현해야 할 주요한 마지막 한 가지는 다른 프로그램을 실행시키기 위한 컨트롤을 셋업하는 것이다. 기억력이 아직 죽지 않았다면 콤보 박스에 세 가지의 윈도우 표준 애플리케이션의 이름을 써넣었던 사실이 생각날 것이다. 여러분의 애플리케이션을 실행시켰을 때, 드롭다운 리스트에서 이 애플리케이션의 이름들이 나온 것도 확인이 되는지 모르겠다. 이 이름 중 하나를 선택하면 콤보 박스에 붙어있는 에디트 박스에 선택한 이름이 설정될 것이다. 자, 콤보 박스는 제대로 되는 것 같으니 여러분이 할 일은 Run Program 버튼에 대한 함수를 추가한 다음, 콤보 박스의 값(프로그램 이름)을 읽어 내고 해당되는 프로그램을 실행시키는 일 뿐이다. 클래스위저드를 사용해서 Run Program 버튼에 대해 함수를 추가했으면 이제 [리스트 2.7]에 나온 코드를 추가하도록 하자.

리스트 2.7 다른 윈도우 애플리케이션을 실행시키는 OnRunpgm() 함수

```

1: void CDay2Dlg::OnRunpgm()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     ///////////////////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     //////////////////////////////
8:
9:     // 화면상의 데이터를 읽어서 변수로 보낸다
10:    UpdateData(TRUE);
11:
12:    // 프로그램 이름을 담을 지역 변수를 하나 선언한다
13:    CString strPgmName;
14:
15:    // 현재 선택된 프로그램 이름을 지역변수에 대입한다
16:    strPgmName = m_strProgToRun;
17:
18:    // 프로그램 이름을 모두 대문자로 바꾼다
19:    strPgmName.MakeUpper();
20:
21:    // 그림판 프로그램을 선택하셨소?

```

```

22:     if (strPgmName == "PAINT")
23:         // 그럴소. 실행하시오
24:         WinExec("pbrush.exe", SW_SHOW);
25:
26:     // 메모장 프로그램을 선택하셨소?
27:     if (strPgmName == "NOTEPAD")
28:         // 그럴소. 실행하시오
29:         WinExec("notepad.exe", SW_SHOW);
30:
31:     // 카드놀이 프로그램을 실행하셨소?
32:     if (strPgmName == "SOLITAIRE")
33:         // 그럴소. 실행하시오
34:         WinExec("sol.exe", SW_SHOW);
35:
36:     /////////////
37:     // 새로 넣을 코드는 여기서 끝난다
38:     /////////////
39: }
```

예상했다시피, 이 함수에서 먼저 한 일은 UpdateData()를 호출해서 컨트롤에 있는 값을 변수로 옮긴 일이다. 하지만 그 다음에 해준 일은 조금 유두리가 없어보이긴 하다. CString 타입의 변수를 선언해서 콤보 박스에서 선택된 값을 대입하였는데, 사실 콤보 박스에서 선택된 값 자체가 CString인데 이렇게 할 필요가 있을까? 이것은 여러분이 결정할 문제이다. 다음 코드는 CString 클래스의 멤버 함수인 MakeUpper()를 호출해서 문자열을 모두 대문자로 바꾸는 부분이다. 콤보 박스에 CString 변수를 물려둔 상태에서 UpdateData()에 FALSE 인수를 주어 호출하면, 콤보 박스에서 현재 선택된 값이 대문자로 설정된다. 이따금씩 이런 일이 발생하는 것은 그리 좋은 동작이 아니다. 이 함수 안에 CString 변수를 넣어둔 이유가 바로 이 때문이다.

일단 문자열을 모두 대문자로 바꾸고 난 다음에는 여러 개의 if 구문을 통해 프로그램 이름과 선택된 이름을 비교한다. 일치하는 것이 있을 경우 WinExec() 함수가 호출되어 애플리케이션을 실행시키게 된다. 정리하면, 드롭다운 리스트에서 프로그램 이름을 선택한 다음, Run Program 버튼을 클릭하면 해당 프로그램이 실행된다.

Caution

C와 C++ 프로그래밍에 있어서 = 기호를 하나 쓸 때(=)와 두 개 쓸 때(==)의 차이를 이해해 두는 것이 중요하다. 단일 = 기호는 오른쪽에 있는 값을 왼쪽에 있는 값으로 대입시키라는 뜻으로, 만약 왼쪽에 상수가 위치해 있으면 프로그램이 컴파일되지 않고 “오른쪽에 있는 값을 왼쪽에 넣을 수 없습니다”라는 멋진 에러 메시지와 인사해야 한다. 이 중 = 기호(==)는 비교 수식에 사용된다. 따라서 두 개의 값을 비교할 때 단일 = 기호를 사용하면 왼쪽에 있는 값을 바꿀 수 있으므로 항상 주의하기 바란다. C/C++ 프로그램에서 논리적인 버그를 발생시키는 큰 요인 중 하나이다.

Note

WinExec() 함수는 현재 윈도우 운영체제의 버전에서는 쓸모가 없어진 윈도우 함수이다. Win32에서는 **CreateProcess()**란 함수를 사용해야 한다. 하지만 CreateProcess 함수 자체가 받아들이는 인수가 너무 많고 비주얼 C++를 처음 프로그래밍하는데 이것을 이해시키기란 좀 무리이다. 또한 WinExec()은 아직도 사용이 가능하며 실제로는 CreateProcess() 함수를 호출하는 매크로로 되어 있기 때문에 다른 애플리케이션을 실행시키는 용도로는 아직 사용해도 무방하다.

참고로, 다른 애플리케이션을 실행시키는데 사용되는 API 함수는 **ShellExecute()**도 있다. 이 함수는 원래 파일을 열거나 인쇄하는 용도로 의도된 것이지만, 다른 프로그램을 실행시킬 때도 사용될 수 있다.

요약

2장에서 배운 것은 비주얼 C++ 애플리케이션에서 표준 윈도우 컨트롤을 어떻게 사용하느냐에 관한 것이었다. 각각의 컨트롤의 값을 읽어내고 쓰기 위한 변수를 선언하고 물려두는 방법과 컨트롤과 그 변수간의 값을 일치시키는 방법을 공부하였고, 컨트롤의 ID를 사용해서 해당 컨트롤의 개체 포인터를 얻어내고 컨트롤을 윈도우처럼 다루어서 조작하는 방법도 아울러 알아보았다. 마지막으로, 여러분의 애플리케이션 윈도우 상의 컨트롤에 원하는 동작을 수행하는 함수를 연결해서, 여러 가지 컨트롤에 대한 사용자 동작을 처리할 수 있도록 해 보았다. 보너스로, 여러분의 애플리케이션 내에서 다른 윈도우 애플리케이션을 실행하는 방법까지 덤으로 구현하였다.

Q&A

? 윈도우 상의 컨트롤에 ID를 연결했을 때, 세 개의 컨트롤이 IDC_STATIC이란 똑같은 ID를 가지더라구요. 하나는 윈도우의 상단에 있는 정적 텍스트이고, 두 개는 그룹 박스이죠. 에디트 박스와 콤보 막스의 왼쪽에 붙은 두 개의 정적 텍스트 컨트롤도, ID를 바꾸기 전에는 항상 똑같은 ID로 시작했던군요. 어떻게 이런 수가 있지요? 그리고, 왜 두 개의 정적 텍스트 컨트롤의 경우에는 ID를 바꾸어야 할까요?

! 정적 텍스트와 그룹 박스와 같은, 보통의 사용자 동작을 받아들이지 않는 모든 컨트롤은 기본적으로 동일한 컨트롤 ID를 가집니다. 여러분의 애플리케이션이 이들 컨트롤에 대해 특별한 동작을 취해주지 않는 이상 가만히 두어도 별 문제는 없지요. 그런데 이 컨트롤들 중 하나에 대해 무언가를 하고 싶으시면 그 컨트롤에 대해선 유일

한 ID를 붙여주어야 합니다. 그래야 GetDlgItem()과 같은 함수로 컨트롤의 개체 포인터를 얻어내어서 비활성화시킨다든가, 숨긴다든가를 할 수 있겠죠. 정적 텍스트 컨트롤이 가진 텍스트를 프로그램 내에서 바꾸기 위해 변수를 물려 두어야 할 때도 유일한 ID를 붙여 주어야 합니다.

동일한 ID를 가지는 정적 컨트롤을 프로그램 내에서 조작하려고 하면 예상치 못한 동작을 일으킬 수도 있음을 항상 염두에 두세요. 일반적인 법칙은, 정적 컨트롤에 특별한 일을 하지 않을 때는 그 컨트롤의 ID를 바꾸지 않되, 어떤 특별한 동작을 처리하고 싶을 때는 유일한 ID를 붙여 주어야 한다는 것입니다.

② ID를 사용해서 컨트롤 개체를 얻어내는 방법 외에 컨트롤을 조작할 수 있는 다른 방법은 없나요?

! 카테고리가 Control인 변수를 선언합시다. 이렇게 하면 해당 컨트롤의 MFC 클래스 타입의 개체가 멤버 변수로 생기고, 컨트롤을 정말 직접 다룰 수 있게 되지요. 그 변수를 사용해서 CWnd 클래스의 멤버 함수를 모두 호출할 수 있고, 컨트롤 클래스만이 가진 특수한 동작을 수행하는 멤버 함수를 호출할 수도 있지요. 예를 들어, 콤보박스 컨트롤에 Control 카테고리를 가지는 변수를 물려두면, 이 변수를 사용해서 드롭다운 리스트에 직접 항목을 추가한다든지 하는 일이 가능해집니다.

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 감추어진 대로 사는게 매력 아니겠는가?

퀴즈

- 애플리케이션 윈도우 상의 컨트롤에 대해 텁 순서를 설정해야 하는 까닭은?
- 에디트 컨트롤이나 콤보 박스로 바로 옮겨갈 수 있도록 정적 텍스트 컨트롤에 니모닉을 넣고 싶다. 어떻게 해야 하는가?

3. 에디트 박스와 콤보 박스 앞에 있는 정적 텍스트 컨트롤에 유일한 ID를 넣어야 하는 이유는 뭘까?
4. 컨트롤의 값을 점검하기 전에 UpdateData()를 호출해야 하는 이유는 무엇인가?

■ 연습문제

1. 에디트 박스에 “Enter a Message here”란 텍스트를 써넣을 수 있도록 Default Message 버튼에 코드를 붙여보자.
2. 다른 윈도우 애플리케이션을 선택하고 실행시키는데 관련된 컨트롤을 숨기거나 보이고, 비활성화시키거나 활성화시킬 수 있는 코드를 작성해 보자.
3. OnRunpgm() 함수를 좀더 고쳐서 사용자로 하여금 직접 프로그램의 이름을 넣고 실행할 수 있도록 해보자.