

# WEEK 2

# DAY 9

## 애플리케이션에 ActiveX 컨트롤을 끼워 넣자

오늘날의 애플리케이션 개발 시장을 살펴보면, 이미 하나의 부품으로 만들어져서 개발중인 프로젝트에 끼워 넣기만 하면 애플리케이션의 기능을 바로 확장시킬 수 있는 컴포넌트들이 지천에 널려 있음을 느낄 수 있다. 이러한 프로그래밍 방식은 원래 비주얼 베이직 프로그래머의 영역으로 생각되었으나, 이제는 비주얼 C++를 비롯한 윈도우 프로그래밍 언어라면 어떤 것인든 ActiveX 컨트롤을 가지고 프로그래밍을 할 수 있게 되었다. 이번 장에서는 바로 이 ActiveX 컨트롤이란 것을 여러분의 비주얼 C++ 애플리케이션에 추가하여 원하는 기능을 직접 만들지 않고도 바로 구사할 수 있도록 해보겠다.

- ❖ ActiveX 컨트롤이란 무엇인가와 작동 매커니즘에 대하여
- ❖ 여러분의 프로젝트 워크스페이스에 ActiveX 컨트롤을 추가하는 방법
- ❖ 비주얼 C++ 애플리케이션에서 ActiveX 컨트롤을 사용하는 방법
- ❖ ActiveX 컨트롤에 속한 여러 가지 메소드 호출 방법
- ❖ ActiveX 컨트롤에 의해 발생되는 메시지 처리 방법

위에 나열한 것을 이번 장에서 배우고 나면 ActiveX 컨트롤에 대해서 많은 편안함을 느낄 수 있을 것이다.

## ■ ActiveX 컨트롤이란?

쉽게 말해서 다른 프로그램에 끼워져서 그 프로그램의 일부인 것처럼 동작하는 소프트웨어 부품, 즉 컴포넌트를 일컫는다. 스테레오 오디오 컴포넌트에 비유할 수 있겠는데, 여러분이 카세트 데크를 구입한 다음 방안에 꾸며진 오디오 세트에 물리면 바로 사용할 수 있는 것과 마찬가지로, ActiveX 컨트롤은 소프트웨어 애플리케이션에 이러한 공통 조작성을 가져온 것이라 할 수 있다.

ActiveX는 원래 OLE 2.0에서 파생된 것이다. OLE 2.0은 두 개 이상의 애플리케이션을 묶어 하나의 형태로 동작하도록(또는 동일한 애플리케이션 골격 안에서 여러 개의 애플리케이션 사이를 옮겨가면서 실행하도록) 설계된 마이크로소프트의 기술이다. 이 개념은 초기의 OLE(개체 연결 및 삽입 : Object Linking and Embedding)에서 확장된 것으로, 여러 가지의 애플리케이션을 사용해서 하나의 복합문서를 만든다는 초기 개념과는 차원을 달리하는 것이었다. 아무튼 이 OLE 2.0 기술을 분산환경(인터넷과 같은)에 적용시키면서 마이크로소프트는 이름까지 달리 지었으며, 바뀐 이 이름이 바로 ActiveX이다.

## ■ ActiveX와 IDispatch 인터페이스

ActiveX 기술은 마이크로소프트의 COM(컴포넌트 개체 모델 : Component Object Model) 기술에 기반을 두고 구축된 것이다. 즉, ActiveX 컨트롤의 통합을 매끄럽게 해주기 위해 필요한 인터페이스와 상호대화 모델이 그대로 사용되었다. COM 기술은 ActiveX 개체의 구성과 인터페이스 설계에 대한 모델을 정의한 것이며, ActiveX 기술

은 COM을 기반으로 만들어진 계층으로서 여러 가지 개체들이 지원해야 하는 인터페이스가 무엇이며, 다른 타입의 개체들이 어떻게 대화해야 하는지를 정의한 것이다.

### Note

마이크로소프트의 COM 기술은 어떻게 애플리케이션과 컴포넌트들이 인터페이스를 통해 대화할 수 있는지를 정의해 두었다. 인터페이스(Interface)는 ActiveX 컴포넌트로 호출할 수 있는 함수와 같다고 보면 된다. 하지만, COM은 이 함수가 어떻게 구축되고 호출될 수 있는지, 그리고 이 함수 호출과 함께 따라주어야 할 지원 기능이 무엇인지를 규정해 놓았다.

모든 COM 개체는 **IUnknown** 인터페이스에서 상속받는다. 이것은 요구사항으로서, 이 인터페이스는 해당 컴포넌트에 의해 지원되는 인터페이스를 알아내는 기능을 가지고 있다. 각각의 인터페이스는 특정한 함수 집합을 지원하고 있어서, 하나의 인터페이스를 통해 ActiveX 컨트롤의 시각적인 특징을 조절할 수 있거나 또 다른 인터페이스를 통해 주변 애플리케이션과의 대화 방식을 조절할 수도 있다. 이런 등등의 인터페이스가 하나의 ActiveX 컨트롤로 지원하게 할 수 있다.

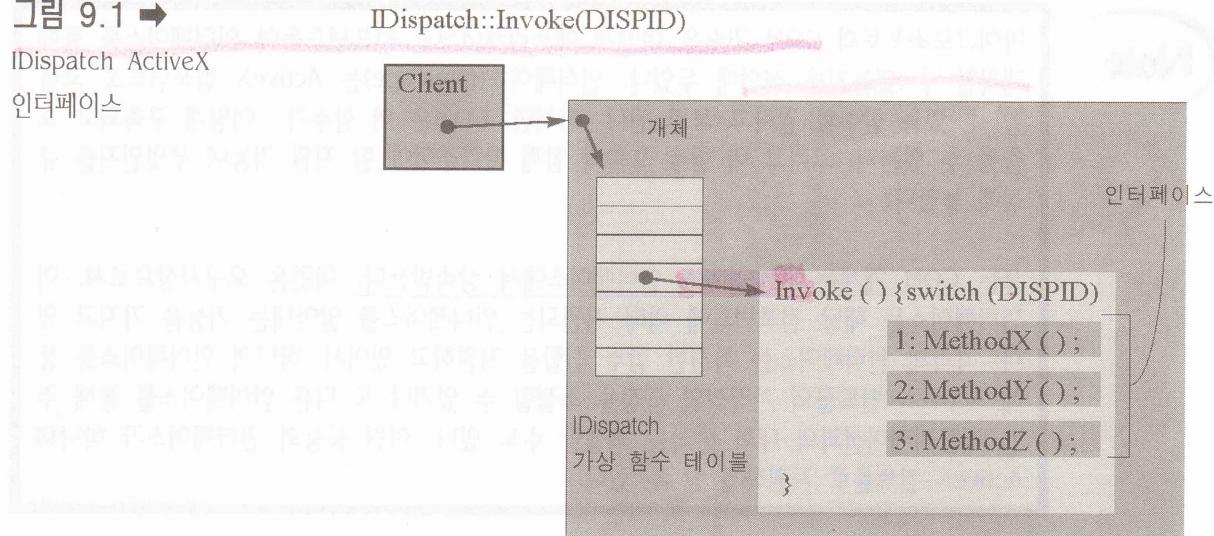
ActiveX 컨트롤을 구성하고 있는 핵심 기술 중 하나가 바로 자동화(automation)이다. 자동화는 다른 애플리케이션에 삽입된 어떤 애플리케이션으로 하여금 자기 자신을 활성화시키고 사용자 인터페이스나 도큐먼트를 변경할 수 있게 하며, 사용자가 다른 부분으로 옮기면 자신을 종료시키는, 쉽게 말하면 어떤 애플리케이션에서 다른 애플리케이션의 기능을 빌려서 사용하는 것이다.

워드 도큐먼트에 삽입된 엑셀 스프레드시트의 예를 들면 쉽게 이해가 갈지도 모르겠다. 여러분이 이 스프레드시트를 클릭하면 엑셀이 활성화되어, 워드를 사용하는 도중에도 이 활성화된 엑셀을 통해 스프레드시트를 편집할 수 있게 된다. 편집을 마치고 다시 워드 사용을 계속하면 엑셀은 자동으로 종료된다.

자동화를 작동시키는 핵심 매커니즘은 **IDispatch**(디스패치 인터페이스(dispatchinterface)라고 알려져 있다)라 불리는 특수한 인터페이스에 있다고 할 수 있다. IDispatch 인터페이스는 ActiveX 컨트롤이나 삽입된 애플리케이션에서 실행할 수 있는 메소드들의 테이블을 가리키는 포인터로 구성되어 있다. 이들 메소드는 **DISPID**라고 불리는 ID 번호로 구성되어 있으며, 이 ID 번호는 특정 메소드에 대한 ID를 찾는데 사용되는 테이블에 로드된다. 일단, 어떤 메소드에 대한 DISPID를 알고 있으면 IDispatch 인터페이스의 **Invoke()** 메소드를 통해 그 메소드를 호출할 수 있다. Invoke() 메소드에 해당 메소드의 DISPID를 넘기면 되는 것이다. [그림 9.1]은 IDispatch 인터페이스

그림 9.1은 Client가 Invoke() 메소드를 사용해서 어떤 ActiveX 개체의 메소드를 호출하는 과정을 보여주고 있다.

그림 9.1 →



## ■ ActiveX 컨테이너와 서버에 대하여

하나의 ActiveX 개체를 다른 ActiveX 개체에 삽입하기 위해서는, 삽입될 개체를 ActiveX 서버(server)로 구현해 주어야 하고, 이 개체를 담는 또 하나의 개체를 ActiveX 컨테이너(container)로 만들어 주어야 한다. 다른 개체에 삽입될 수 있는 ActiveX 개체를 ActiveX 서버라고 부르며, 완전히 독립적으로 작동되는 프로그램이나 소형 ActiveX 컨트롤로 만들 수 있다. 이와 반대로, 다른 ActiveX 개체를 자신에 포함시킬 수 있는 ActiveX 개체를 ActiveX 컨테이너라고 부른다.

### Note

컨테이너와 서버라는 용어를 (그림 9.1)에 나온 클라이언트(client)라는 용어와 혼동하지 않도록 하자. 클라이언트는 다른 개체의 Idispatch 인터페이스를 통해 메소드를 호출하는 개체를 일컫는다. 이제 한 페이지 정도 더 읽고 나면 컨테이너나 서버나 모두 다른 개체의 Idispatch 인터페이스를 필요로 하는 클라이언트가 될 수 있다는 사실을 알게 될 것이다.

이들 두 ActiveX 개체의 타입은 서로 배타적인 개념이 절대 아니다. ActiveX 서버라고 해서 동시에 ActiveX 컨테이너가 되지 말란 법은 없는 것이다. 좋은 예를 들어보자면, 마이크로소프트의 인터넷 익스플로러가 있겠는데, 이 웹 브라우저는 ActiveX 컨테이너(이를테면 워드, 액셀, 파워포인트 등)에서 작동되는 ActiveX 서버로 구현되어 있으며, 동시에 다른 ActiveX 컨트롤을 내부에 가질 수 있다.

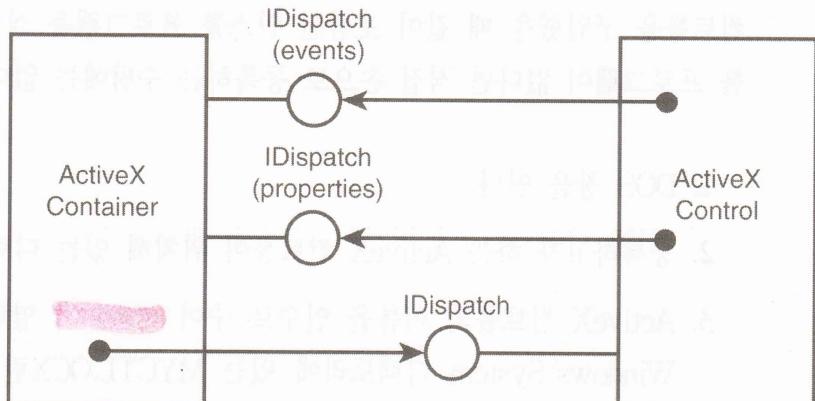
ActiveX 컨트롤은 ActiveX 서버의 특수한 예라고 할 수 있다. 어떤 ActiveX 서버 중에는 독립적으로 실행될 수 있는 애플리케이션도 있지만, ActiveX 컨트롤은 독립적으로는 실행될 수 없으며, 다른 ActiveX 컨테이너에 삽입되어야 제 기능을 발휘할 수 있다. ActiveX 컨트롤을 여러분의 비주얼 C++ 애플리케이션에서 사용하면, 여러분의 애플리케이션은 자동적으로 ActiveX 컨테이너가 된다.

ActiveX 컨테이너와 ActiveX 컨트롤 사이에 행해지는 대화의 대부분은 세 개의 IDispatch 인터페이스를 통해 이루어진다. 이 IDispatch 인터페이스 중 하나가 ActiveX 컨트롤에 위치해 있으며, 이 녀석이 컨테이너에 의해 사용되어 해당 ActiveX 컨트롤을 통해 액세스할 수 있는 여러 가지 메소드가 호출되는 것이다.

ActiveX 컨테이너는 두 개의 IDispatch 인터페이스를 ActiveX 컨트롤에 제공한다. 이 중에서 첫번째 인터페이스는 컨트롤에 의해 사용되어 컨테이너 애플리케이션에서 이벤트를 발생시킨다. 두번째는 [그림 9.2]에 나온 것과 같이 컨트롤의 프로퍼티를 설정하는데 사용된다. ActiveX 컨트롤의 프로퍼티는 실제로 컨테이너에 의해 제공되는 것이 대부분이지만, 유지 관리는 컨트롤이 맡는다. 이러한 동작 대부분은 여러분이 직접 제어할 수 있는데, 왜냐하면 비주얼 C++가 ActiveX 컨트롤에 대한 C++ 포장 클래스를 만들어 주기 때문이다. 따라서 여러분은 컨트롤의 IDispatch 인터페이스를 직접 요구하지 않고도 이 C++ 클래스에 의해 개방된 메소드를 통해 컨트롤과 대화 할 수 있게 된다.

그림 9.2 →

ActiveX 컨테이너와 컨트롤은 일차적으로 IDispatch 인터페이스를 통해 대화한다.



## 프로젝트에 ActiveX 컨트롤을 추가하자

“너무나 사용하기 쉽다고 하는” ActiveX 컨트롤의 숨겨진 내막을 조금이라도 여러분 것으로 만드는데에는 직접 여러분의 애플리케이션에 넣어 보는 것이 그만이다. 비주얼 C++를 사용하면 ActiveX 컨트롤을 애플리케이션에 추가하는 일이 더욱 쉬워진다. 그렇다고 해도 ActiveX 컨트롤을 끼울 대상이 없으면 아무 소용이 없는 법. 애플리케이션 골격을 만들도록 하자.

1. ActiveX란 이름의 MFC AppWizard 프로젝트를 새로 만든다.
2. 지금까지 해왔듯이 모든 애플리케이션 위저드의 디폴트 설정을 그대로 받아들이고, 두번째 단계에서 애플리케이션 타이틀을 ActiveX Controls로 하자.
3. 애플리케이션 골격이 다 만들어졌으면, 디자인 창상의 모든 컨트롤을 삭제하고 버튼 하나만을 추가하자.
4. 이 버튼의 ID를 IDC\_EXIT로 하고, 캡션은 E&xit로 한다.
5. 방금 만든 버튼에 함수를 연결하고, 이 함수에서는 OnOK()를 호출하도록 하자.

## 컨트롤을 등록하자

ActiveX 컨트롤을 디자인 창상에 추가하기 전에 필요한 일은 바로 이 컨트롤을 비주얼 C++와 윈도우 운영체제에 등록하는 일이다. ActiveX 컨트롤을 윈도우 운영체제에 등록하는 방법에는 두 가지가 있는데, 그 중에서 첫번째는 ActiveX 컨트롤을 구입했을 때 같이 포함된 인스톨 프로그램을 실행시키는 것이다. 만일 인스톨 프로그램이 없다면 직접 손으로 등록하는 수밖에 없다. 다음의 단계를 따르자.

1. DOS 창을 연다.
2. 등록하고자 하는 ActiveX 컨트롤이 위치해 있는 디렉토리로 옮긴다.
3. ActiveX 컨트롤의 이름을 인수로 주어 **regsvr32** 명령을 실행시킨다. 예를 들어, Windows\System 디렉토리에 있는 MYCTL.OCX란 이름의 컨트롤을 등록하고자 한다면 다음과 같이 하는 것이다.

```
C:\WINDOWS\cd system
C:\WINDOWS\SYSTEM\regsvr32 MYCTL.OCX
```

**Caution**

컨트롤 인스톨 프로그램이 있다면 되도록 이 프로그램을 사용하는 것이 좋다. 왜냐하면, 손으로 직접 등록하는 과정은 ActiveX 컨트롤을 애플리케이션 개발용으로 만들어 주지 않을 수도 있기 때문이다. ActiveX 컨트롤은 개발용과 배포용으로 라이센스될 수 있다. 만일 어떤 컨트롤이 배포용일 경우에는 비주얼 C++ 애플리케이션에 끼워서 사용할 수 없다. 컨트롤에 대한 개발 라이센스를 구매하도록 요구함으로써 컨트롤 개발자들의 생계를 보호하기 위한 장치라고 생각하면 된다.

**Note**

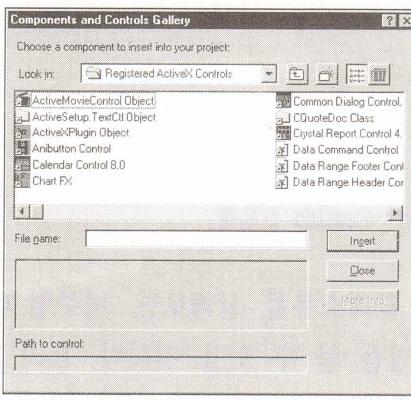
COM과 ActiveX 개체는 윈도우 레지스트리 데이터베이스에 상당한 양의 정보를 저장한다. 어떤 애플리케이션이 ActiveX 개체를 사용할 때마다, 윈도우 운영체제는 윈도우 레지스트리에 저장된 정보를 참조해서 해당 개체를 찾아낸 다음 사용 여부를 결정짓는다. ActiveX 컨트롤 등록하는데 사용되는 regsvr32.exe 유ти리티가 바로 윈도우 레지스트리에 해당 컨트롤에 대한 정보를 저장하는 구실을 하는 것이다. 하지만, ActiveX 컨트롤이 제대로 동작하게 하려면 컨트롤에 대한 추가적인 정보가 레지스트리에 저장되어야만 한다.

이제 여러분이 사용하게 될 ActiveX 컨트롤이 운영체제에 등록되었으므로, 비주얼 C++에 등록한 다음, 프로젝트에 추가하는 단계가 남았다. 다음의 단계를 따르자.

1. 비주얼 C++의 메뉴에서 **Project | Add to Project | Component and Controls**를 선택한다.
2. Components and Controls Galleryダイ얼로그에서 Registered ActiveX Controls 폴더로 옮긴다([그림 9.3] 참조).

**그림 9.3** ➔

프로젝트에 추가할 수 있는 ActiveX 컨트롤들

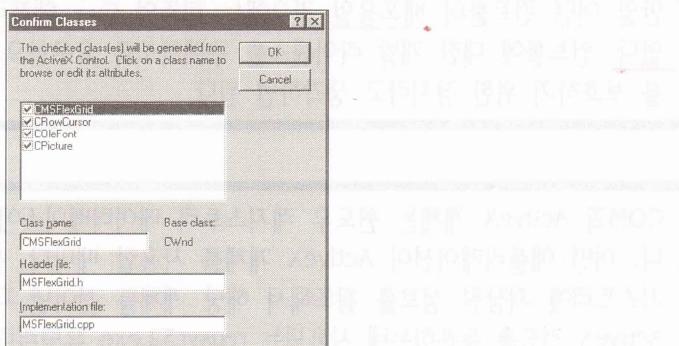


3. 등록하고자 하는 컨트롤을 선택하고(여기서는 Microsoft FlexGrid를 선택하자), Insert 버튼을 클릭한다.
4. 이 컨트롤을 여러분의 프로젝트에 추가하겠냐는 메시지 박스에서 OK를 클릭한다.

5. Confirm Classesダイアル로그에서 OK를 클릭하여 C++ 클래스를 추가한다([그림 9.4] 참조).

그림 9.4 →

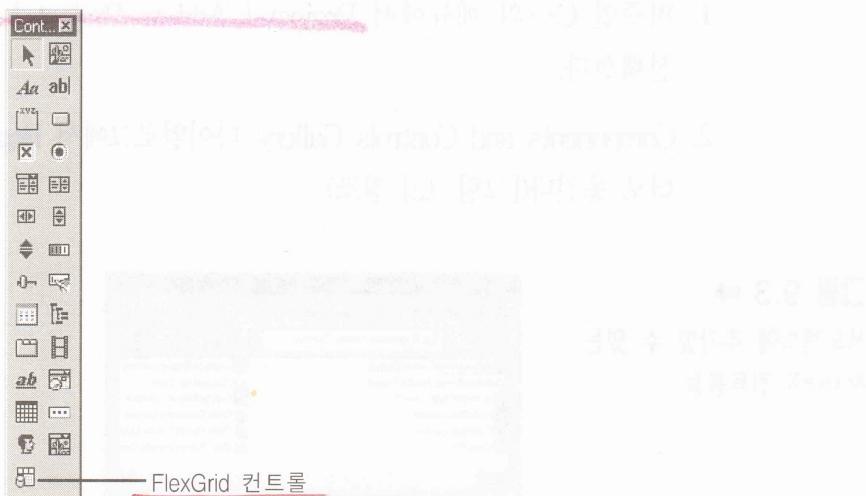
어떤 C++ 클래스가 여러분의 프로젝트에 추가될 것인지를 알려주고 있다.



6. Components and Controls Galleryダイ얼로그 상의 Close 버튼을 클릭해서 컨트롤 추가를 마친다.
7. 이제 FlexGrid 컨트롤이 Controls 도구바에 들어가 있음을 확인할 수 있을 것이다. [그림 9.5]와 같이 말이다.

그림 9.5 →

FlexGrid란 ActiveX 컨트롤이 Controls 도구바에 추가되어ダイ얼로그 윈도우에 붙일 수 있게 설정되었다.



프로젝트 워크스페이스의 클래스뷰를 살펴보면, 비주얼 C++가 여러분의 프로젝트에 추가시킨 클래스가 네 개임을 알 수 있을 것이다. 트리 구조를 펼쳐서 이들 클래스가 개방하고 있는 메소드가 무엇이 있는지 보도록 하자. 비주얼 C++가 여러분이 방금 추가한 ActiveX 컨트롤을 조사해서 적합한 클래스와 메소드를 생성해 주었고, 이 컨트롤의 IDispatch 인터페이스에 들어있는 각각의 메소드를 호출할 수 있도록 한 것이다.

**Note**

만일 비주얼 C++ 애플리케이션에 오래된 ActiveX 컨트롤을 사용한다면, 비주얼 C++가 이 컨트롤에 대한 클래스와 메소드를 생성하는 작업이 안될 수도 있다는 사실을 알아두자. 컨트롤에 대한 클래스와 메소드를 만들어 내기 위해 비주얼 C++에 제공되는 정보는 최신 버전의 ActiveX 스페에 나타나 있기 때문에, 오래된 버전의 컨트롤에는 이 정보가 들어있지 않을 가능성이 매우 높으며, 결과적으로 비주얼 C++에서 사용하기가 조금 힘들다.

## ■ 여러분의 디자인에 컨트롤을 추가하자

FlexGrid 컨트롤을 프로젝트에 추가하는 일은 끝났고, 이젠 이 컨트롤을 디자인 윈도우에 추가하여 사용할 수 있을 것이다. 이 컨트롤의 프로퍼티 설정은 [표 9.1]을 참고하자.

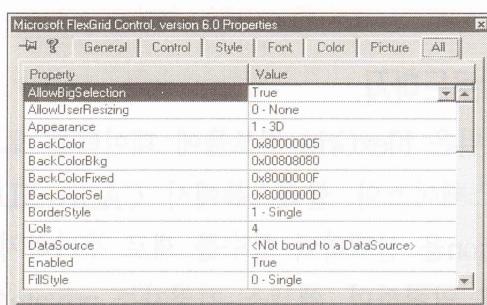
[표 9.1] 컨트롤의 프로퍼티 설정

컨트롤	프로퍼티	설정
FlexGrid	ID	IDC_MSGRID
	Rows	20
	Cols	4
	MergeCells	2 - Restrict Rows
	Format	Format   < Region   < Product   < Employee   >Sales (서식화문자열)

여러분의 디자인 윈도우에 이 컨트롤을 추가한 다음, 프로퍼티 디자인 윈도우를 띄우면 탭이 몇 개 더 붙어 있음을 알 수 있다([그림 9.6] 참조). 이 탭을 사용해서 컨트롤의 모든 프로퍼티를 설정할 수 있으며, 여느 표준 컨트롤처럼 다른 탭도 사용할 수 있을 것이다.

그림 9.6 ➔

ActiveX 컨트롤은 모든 컨트롤 프로퍼티가 포함된 프로퍼티 탭을 가지고 있다.



추가시킨 컨트롤의 프로퍼티 설정이 모두 끝났으면, 여러분의 프로그램 코드에서 이 컨트롤과 대화하기 위해 이 컨트롤에다가 변수를 하나 물려 두어야 한다. 클래스위저드의 Member Variable 탭을 선택하고 이 컨트롤에 대한 변수를 추가하자. 여러분은 ActiveX 컨트롤에 대해 변수를 물려두는 것이므로 단 하나의 컨트롤 변수만을 가지게 되며, 다른 것은 볼 필요 없이 변수 이름만 잘 지어주면 된다. 우리의 예제 애플리케이션에서는 m\_ctlFGrid라고 짓자.

## 애플리케이션에서 ActiveX 컨트롤을 사용하자

일단 비주얼 C++가 FlexGrid ActiveX 컨트롤을 포장하는 클래스를 모두 만들어 준 상태이기 때문에, 이 컨트롤로 해가는 작업은 어느 표준 컨트롤과 다를 바 없이 메소드를 호출하거나 컨트롤 이벤트를 처리하는 일 뿐이다. 우리는 우선 이 컨트롤의 메소드를 사용해서 컨트롤의 정보를 알아내고 수정해 보도록 하는 것으로 시작하자. 그 다음, 비주얼 C++를 사용해서 컨트롤 이벤트를 처리하는 방법에 대해서도 공부하게 될 것이다.

### 컨트롤과 대화하자

이번 장에서 만들고 있는 애플리케이션은 네 명의 판매원을 가진 다섯 개의 판매 지역에서 상품 판매 내역을 표시하도록 할 것이다. 애플리케이션을 사용해서 지역과 상품별로 정렬된 데이터 화면을 스크롤해 가면서 각 판매원이 각 상품을 얼마나 팔았는지 비교할 수 있을 것이다.

이 프로젝트를 만들려면 그리드를 구성하는 각 셀에 로드될 값들의 배열을 만들어 두어야 한다. 그리드는 오름차순으로 정렬되는데, FlexGrid 컨트롤의 내부 정렬 기능이 사용된다.

#### 컨트롤에 데이터를 로드하자.

일단 먼저 해야 할 일은 FlexGrid 컨트롤에 데이터를 로드하는 함수를 만드는 것이다. 프로젝트 워크스페이스의 클래스뷰에서 CActiveXDlg를 오른쪽 클릭한 다음, Add Member Function을 선택하여 함수를 새로 추가하자. 함수의 타입은 void, 선언 형식은 LoadData()로 입력하며, 액세스 지정자는 private으로 하자. OK 버튼을 클릭하면 함수를 편집할 수 있게 될 것이다. [리스트 9.1]을 입력하도록 하자.

### 리스트 9.1 LoadData() 함수

```

1: void CActiveXDlg::LoadData()
2: {
3:     int iCount;           // 그리드의 행 카운트
4:     CString strAmount;   // 판매량
5:
6:     // 난수 발생기를 초기화한다
7:     srand((unsigned)time(NULL));
8:     // 컨트롤에 배열을 생성한다
9:     for (iCount = m_ctlFGrid.GetFixedRows();
10:          iCount < m_ctlFGrid.GetRows(); iCount++)
11:     {
12:         // 첫번째 컬럼(지역) 값을 생성
13:         m_ctlFGrid.SetTextArray(GenID(iCount, 0), RandomStringValue(0));
14:         // 두번째 컬럼(제품) 값을 생성
15:         m_ctlFGrid.SetTextArray(GenID(iCount, 1), RandomStringValue(1));
16:         // 세번째 컬럼(판매원)값 생성
17:         m_ctlFGrid.SetTextArray(GenID(iCount, 2), RandomStringValue(2));
18:         // 판매량 값 생성
19:         strAmount.Format("%5d.00", rand());
20:         // 네번째 컬럼에 값을 채움
21:         m_ctlFGrid.SetTextArray(GenID(iCount, 3), strAmount);
22:     }
23:
24:     // 공통적으로 이어지는 행을 합친다
25:     m_ctlFGrid.SetMergeCol(0, TRUE);
26:     m_ctlFGrid.SetMergeCol(1, TRUE);
27:     m_ctlFGrid.SetMergeCol(2, TRUE);
28:
29:     // 그리드를 정렬한다
30:     DoSort();
31: }

```

우선 난수 발생기를 초기화했으며, 컨트롤 내의 행 개수만큼 루프를 돌면서 각각의 셀에 데이터를 넣었다. 컨트롤의 행 개수를 알아내는데는 GetRows() 메소드를 사용했으며, 헤더의 개수는 GetFixedRows() 메소드를 사용했다. 컨트롤의 셀에 데이터를 넣는 작업은 SetTextArray() 메소드를 사용했는데, 이 메소드는 첫번째 인수로 셀 ID를, 두번째 인수로 셀에 채울 데이터를 받는다. 셀 ID와 셀에 넣을 데이터는 곧 여러분이 만들 또 하나의 함수에 의해 자동으로 만들어지니까 걱정말기 바란다.

그리드 셀에 데이터를 넣은 다음에는 SetMergeCol() 메소드를 호출해서 인접한 행이 같은 값을 가지고 있으면 처음 세 개의 컬럼에 속해 있는 셀을 합치도록 컨트롤에게 지시하였다. 마지막으로 컨트롤을 정렬(sort)했는데, 이 작업 역시 여러분이 곧 만들게 될 함수를 통해 수행될 것이다.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

## 셀 ID를 계산하자

FlexGrid 컨트롤을 구성하고 있는 셀은 왼쪽에서 오른쪽으로, 위쪽에서 아래쪽으로 번호가 매겨진다. 여러분이 추가한 컨트롤의 경우 첫번째 행은 0부터 3까지, 두번째 행은 4부터 7까지 번호가 매겨져 있을 것이다. 따라서 셀의 ID는 컬럼의 총 개수에 현재의 행 번호를 곱한 값에 현재의 컬럼 번호를 더하면 구할 수 있다. 예를 들어, 여러분의 컨트롤이 네 개의 컬럼을 가지고 있고, 세번째 컬럼과 네번째 행에 있는 셀의 ID를 구하고 싶다면  $2 + (4 \times 3) = 14$ 로 계산하면 된다. 행과 컬럼의 번호는 0부터 시작한다는 사실을 잊지 말자. 따라서 세번째 컬럼의 번호는 2이며, 네번째 행의 번호는 3이다.

이제 어떻게 셀 ID를 계산하는지는 파악했으니, 직접 이 식을 함수로 구현해 보도록 하자. CActiveXDlg 클래스에 새 함수를 추가하는데, 타입은 int, 선언 형식은 GenID(int m\_iRow, int m\_iCol)이며, 액세스 지정자는 private으로 정해주자. 여러분이 입력할 코드는 [리스트 9.2]에 준비되어 있다.

### 리스트 9.2 GenID() 함수

```

1: int CActiveXDlg::GenID(int m_iRow, int m_iCol)
2: {
3:     // 컬럼의 개수를 얻어낸다
4:     int iCols = m_ctlFGrid.GetCols();
5:
6:     // 컬럼 개수, 현재 컬럼, 현재 행을 가지고
7:     // ID를 생성한다
8:     return (m_iCol + iCols * m_iRow);
9: }
    2 + 4 * 4

```

## 난수를 발생시키자

그리드 안에 있는 처음 세 개의 컬럼에 데이터를 넣기 위해서는 데이터를 임의로 생성해낼 필요가 있다. 첫번째 컬럼에는 지역 이름을, 두번째 컬럼에는 상품 이름을, 세번째 컬럼에는 판매원 이름을 넣게 된다. 데이터를 생성해낼 컬럼을 결정하기 위해 switch 구문을 사용하고 또 하나의 switch 구문 안에서는 생성된 난수에 대해 나머지 연산을 수행함으로써, 제한된 범위의 문자열 데이터를 무작위로 선택할 수 있다.

이 함수를 구현하기 위해 CActiveXDlg 클래스에 함수를 새로 하나 더 추가하자. 타입은 CString, 선언 형식은 RandomStringValue(int m\_iColumn)이며, 액세스 지정자는 private이다. 끝났으면 [리스트 9.3]에 나온 코드를 입력하자.

### 리스트 9.3 RandomStringValue() 함수

```

52:           // 2 - Whatchamacallits
53:           strStr = "Whatchamacallits";
54:           break;
55:       case 3:
56:           // 3 - Round Tufts
57:           strStr = "Round Tufts";
58:           break;
59:       default:
60:           // 4 - Widgets
61:           strStr = "Widgets";
62:           break;
63:       }
64:   break;
65: case 2: // 세번째 컬럼 (employee)
66: // 0 과 3 사이의 난수를 생성
67: iCase = (rand() % 4);
68: // 어떤 값이 생성되었나?
69: switch (iCase)
70: {
71:     case 0:
72:         // 0 - Dore
73:         strStr = "Dore";
74:         break;
75:     case 1:
76:         // 1 - Harvey
77:         strStr = "Harvey";
78:         break;
79:     case 2:
80:         // 2 - Pogo
81:         strStr = "Pogo";
82:         break;
83:     default:
84:         // 3 - Nyra
85:         strStr = "Nyra";
86:         break;
87:     }
88:     break;
89: }
90: // 생성된 문자열을 반환한다.
91: return strStr;
92: }

```

## 컨트롤을 정렬하자

FlexGrid 컨트롤을 정렬하기 위해서 모든 컬럼을 선택한 다음, 정렬 방식을 오름차순으로 정한다. 정렬에 필요한 핵심 기능은 이미 구현되어 있으니 꺽데기 함수만 작성하면 된다. CActiveXDlg 클래스에 void 타입의 DoSort()라는 이름을 가진 함수를 하나 더 추가하자. 이 함수에 대한 코드는 [리스트 9.4]를 참고하자.

#### 리스트 9.4 DoSort() 함수

```

1: void CActiveXDlg::DoSort()
2: {
3:     // 현재의 컬럼을 0번 컬럼으로 설정
4:     m_ctlFGrid.SetCol(0);
5:     // 전체 컬럼을 선택
6:     m_ctlFGrid.SetColSel((m_ctlFGrid.GetCols() - 1));
7:     // 오름차순 정렬
8:     m_ctlFGrid.SetSort(1);
9: }
```

우선 SetCol() 메소드를 사용해서 현재의 컬럼을 첫번째 컬럼으로 설정하였다. 다음 현재 컬럼부터 마지막 컬럼까지의 범위를 SetColSel() 메소드를 사용해서 설정하였기 때문에 컨트롤 내의 모든 컬럼을 선택한 셈이 된다. 마지막으로, SetSort() 메소드를 사용해서 정렬 방식을 오름차순으로 정해주었는데, 인수로 넘긴 1이 바로 오름차순 이란 뜻이다.

이제 데이터를 컨트롤에 로드하는데 필요한 모든 함수는 다 준비되었으므로, LoadData() 함수를 OnInitDialog() 함수 안에서 호출하여 FlexGrid 컨트롤이 사용자에게 보여지기 전에 데이터를 로드하게 하면 깔끔한 끝마무리가 된다. OnInitDialog() 함수를 편집해서 [리스트 9.5]와 같이 만들자.

#### 리스트 9.5 OnInitDialog() 함수

```

1: BOOL CActiveXDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:     .
5:     .
6:     .
7:     // TODO: Add extra initialization here
8:     .
9:     ///////////////
10:    // 새로 넣을 코드가 여기서부터 시작된다
11:    ///////////////
12:    .
13:    // 그리드 컨트롤에 데이터를 로드한다
14:    LoadData();
15:    .
16:    ///////////////
17:    // 새로 넣을 코드는 여기서 끝난다
18:    ///////////////
```

```

19:
20:     return TRUE; // return TRUE unless you set the focus to a control
21: }

```

이제 애플리케이션을 컴파일하고 실행하면, [그림 9.7]과 같이 데이터가 로드되어 정렬된 그리드 컨트롤이 들어 있는 디자인로그 윈도우를 보게 될 것이다.

그림 9.7 →

데이터가 로드된 FlexGrid  
컨트롤



## ■ 컨트롤 이벤트를 처리하자

현재 작동중인 애플리케이션의 그리드 컨트롤에는 데이터를 넣어보려고 해도 그리드 컨트롤이 반응하지 않는다. 셀 하나를 클릭한 다음, 값을 바꾸려고 해보자. 아마 전혀 신경도 안 쓸 것이다. 이제 필요한 일은 컨트롤의 이벤트를 처리해서 입력을 받아들 이도록 하는 것이다. ActiveX 컨트롤은 비주얼 C++ 애플리케이션에서 사용할 수 있는 이벤트 몇 가지를 개방하고 있으며, 여러분은 클래스위저드를 사용해서 이 이벤 트들을 살펴본 다음, 어떤 이벤트에 함수를 연결하고 어떤 이벤트를 무시할 것인지를 결정할 수 있다. 대부분의 ActiveX 컨트롤은 자신이 발생시키는 이벤트에 대한 디폴트 처리 루틴을 가지고 있지 않으며, 그대신 여러분이 알아서 컨트롤에 대한 동작을 코딩해 주어야 한다.

우리는 마우스 클릭과 이동에 관한 컨트롤 이벤트 두 개에 주목하도록 하자. 우선 사용자가 헤더를 클릭해서 다른 위치를 끌어 옮길 수 있게 하는 기능을 넣을 것인데, 마우스 버튼이 눌려졌을 때와 떼어졌을 때의 컨트롤 이벤트를 잡아내어야 한다. 전자의 이벤트가 발생했을 때는 사용자가 헤더를 클릭했는지를 체크하고, 이때 사용자가 헤더를 클릭했다면 선택된 컬럼을 포착해서 처리한다. 후자의 이벤트가 발생했을 때는 선택된 컬럼을 마우스 버튼이 떼인 위치의 컬럼으로 이동시켜야 한다.

방금 설명한 기능을 구현하기 위해서는 두 이벤트가 발생했을 때 클릭된 컬럼 번호를 유지하는 변수가 클래스 멤버로 필요할 것 같다. CActiveXDlg 클래스에 새 변수를 추가하자. 타입은 int, 이름은 m\_iMouseCol로 입력하고, 액세스 지정자는 private로 해주자.

## 선택된 컬럼을 포착하자

컨트롤의 마우스 클릭 이벤트를 포착하려면 다음의 단계를 따르자.

1. 클래스위저드를 사용해서 IDC\_MSFGRID 컨트롤의 MouseDown 이벤트에 함수를 하나 추가하자.
2. [리스트 9.6]에 있는 코드를 이 함수에 입력하자.

### 리스트 9.6 OnMouseDownMsfgrid() 함수

```

1: void CActiveXDlg::OnMouseDownMsfgrid(short Button, short Shift, long x, long y)
2: {
3:     // TODO: Add your control notification handler code here
4:     //////////////////////////////
5:     // 새로 넣을 코드가 여기서부터 시작된다
6:     //////////////////////////////
7:
8:
9:     // 사용자가 헤더 행이 아닌
10:    // 데이터 행을 클릭했나?
11:    if (m_ctlFGrid.GetMouseRow() != 0)
12:    {
13:        // 그렇다면, 컬럼 변수를 0으로하고
14:        // 끝낸다
15:        m_iMouseCol = 0;
16:
17:        return;
18:    }
19:    // 클릭된 컬럼을 저장한다
20:    m_iMouseCol = m_ctlFGrid.GetMouseCol();
21:    //////////////////////////////
22:    // 새로 넣을 코드는 여기서 끝난다
23:    //////////////////////////////
24: }
```

이 함수에서는 우선 GetMouseRow() 메소드를 호출해서 클릭된 행 번호를 알아내었다. 만일 클릭된 행이 첫번째 행이면 컬럼 번호를 담고 있는 변수를 0으로 만들고 함수를 끝낸다. 그렇지 않을 경우에는 GetMouseCol() 메소드를 호출해서 클릭된 컬럼 번호를 알아낸다. 이 값을 m\_iMouseCol에 저장해 두도록 하였다.

## 마우스 버튼을 빼었을 때 컬럼을 이동시키자

선택된 컬럼 번호를 잡아내었으므로, 여러분이 필요한 것은 마우스 버튼이 빼였을 때의 컬럼 번호이다. 컨트롤의 마우스 버튼 빼임 이벤트를 잡아내기 위해 다음의 단계를 따르자.

1. 클래스위저드를 사용해서 IDC\_MSFGRID 컨트롤의 MouseUp 이벤트에 함수를 하나 추가하자.
2. [리스트 9.7]에 있는 코드를 이 함수에 입력하자.

### 리스트 9.7 OnMouseUpMsfgrid() 함수

```

1: void CActiveXDlg::OnMouseUpMsfgrid(short Button, short Shift, long x, long y)
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     ///////////////
8:
9:     // 선택된 컬럼이 첫번째 컬럼이면
10:    // 할 일이 없다
11:    if (m_iMouseCol == 0)
12:        return;
13:    // 컨트롤 다시 그리기를 해제
14:    m_ctlFGrid.SetRedraw(FALSE);
15:    // 선택된 컬럼 위치를 바꿈
16:    m_ctlFGrid.SetColPosition(m_iMouseCol, m_ctlFGrid.GetMouseCol());
17:    // 그리드를 다시 정렬
18:    DoSort();
19:    // 컨트롤 다시 그리기를 복구
20:    m_ctlFGrid.SetRedraw(TRUE);
21:
22:    /////////////////
23:    // 새로 넣을 코드는 여기서 끝난다
24:    ///////////////
25: }
```

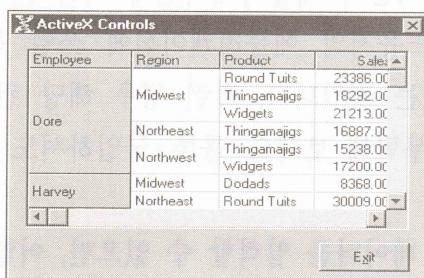
일단 이동시킬 만한 선택된 컬럼이 있는지 체크한다. 만일 선택된 컬럼이 없다면 아무런 할 일이 없으므로 함수를 끝낸다. 선택된 컬럼이 있다면 SetRedraw()를 호출해서 컨트롤의 다시 그리기 기능을 해제하여 어떠한 이동 상태도 사용자에게 보이지 않게 한다. 다음, SetColPosition() 메소드를 사용해서 마우스 버튼이 빼인 컬럼에 선택된 컬럼을 이동시킨다. 일단 컬럼을 이동시켰으면 DoSort()를 호출해서 그리드를

다시 정렬해야 한다. 마지막으로, 컨트롤의 다시 그리기 기능을 복구해서 이동된 컬럼을 다시 사용자가 볼 수 있도록 한다.

이제 애플리케이션을 컴파일하고 실행시키면 [그림 9.8]과 같이 컬럼 헤더를 마우스로 끌어서 이동시킬 수 있게 될 것이다.

**그림 9.8**

다시 정렬된 상태의  
FlexGrid 컨트롤



## 요약

이번 장에서는 ActiveX 컨트롤을 원래 만들고 있던 애플리케이션에 끼워 그대로 애플리케이션의 기능을 확장하는 방법을 공부하였다. ActiveX 컨트롤이 그 컨트롤을 포함하는 애플리케이션과 어떤 상호작용을 하는지에 대한 기본 지식을 먼저 닦았으며, 프로젝트에 ActiveX 컨트롤을 추가하는 방법을 실습해 보았다. 이때 비주얼 C++ 가 프로젝트에 추가된 ActiveX 컨트롤의 기능을 감싸는 C++ 클래스를 만들어 내는 것을 확인하였고, 이 클래스를 사용하여 ActiveX 컨트롤과 대화할 수 있었다. 마지막으로, 이 ActiveX 컨트롤이 발생시키는 이벤트를 잡아내어 애플리케이션에서 처리하는 방법을 공부하였다.

## Q&A

어떤 ActiveX 컨트롤을 사용할 때 말이죠. 이 컨트롤이 어떤 메소드를 가지고 있는지에 대해 알아내려면 어떻게 하나요?

비주얼 C++가 컨트롤을 감싸서 만든 C++ 클래스를 살펴보면 메소드 목록이 나오는데, 일차적으로 이 정보를 사용할 수 있습니다. 만일 여러분이 구입하신 컨트롤에 대한 문서가 있으시면 이 C++ 클래스의 정보와 비교해 보셔도 좋겠죠. 그리고 어떤 컨트롤이 발생시키는 이벤트에 대해 알고 싶으시면 클래스위저드를 열어 보세요.

**?** 다른 애플리케이션에서 사용되는 것으로 제 컴퓨터에 설치된 ActiveX 컨트롤을 사용하려면 어떻게 하나요?

**!** 그 컨트롤이 어떻게 라이센스되었는가와 어떤 애플리케이션이 그 컨트롤을 설치했는가에 따라 다릅니다. 만일 다른 애플리케이션 개발 도구로 설치되었다면, 여러분도 사용하실 수 있는 가능성이 있습니다. 즉, 개발 라이센스가 있을 수 있다는 뜻이죠 (이 경우 이 컨트롤을 여러분의 애플리케이션에 끼워서 사용하실 줄 알아야 겠지요). 반면에, 워드와 같은 엔드유저의 애플리케이션에 쓰일 목적으로 라이센스되었다면 런타임 라이센스 밖에 없는 것입니다. 이런 경우 해당 컨트롤의 개발자에게 전화를 거셔서 개발 라이센스를 부탁해 보세요. 물론, 구입하셔도 되구요.

**?** FlexGrid 컨트롤에 직접 데이터를 입력할 수 없으면, 어떻게 스프레드시트를 사용하는 느낌이 들도록 하나요?

**!** 원도우에 에디트 박스를 하나 생기게 해야 합니다. 여러분의 코드에서는 어떤 셀을 편집할 것인가를 결정한 다음 해당 셀 위에 그 에디트 박스를 띄우는 것이죠. 즉, 사용자에게 셀에 직접 데이터를 입력한다는 느낌을 주게 하는 것입니다. 또 다른 방법은 그리드 바깥쪽에 데이터 필드를 만드는 것인데요. 액셀에서 사용하는 방법입니다. 사용자들이 그리드 안의 셀을 돌아다닐 때 셀을 반전시킨다든지, 밝게 만들 어준다든지 하면 더욱 확실한 효과가 되겠지요.

## 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 있으며, 미리 보면 나쁜 사람이다.

### 퀴즈

1. ActiveX 컨테이너는 어떻게 ActiveX 컨트롤이 가진 메소드를 호출할까?
2. ActiveX 컨트롤은 어떻게 컨테이너 애플리케이션 내에서 이벤트를 발생시킬까?
3. 비주얼 C++ 애플리케이션에서 ActiveX 컨트롤이 제대로 작동되게 하기 위해서는 애플리케이션 위저드에서 어떤 옵션을 선택해야 할까?

4. 비주얼 C++는 ActiveX 컨트롤의 사용을 쉽게 하기 위해 어떤 동작을 할까?
5. 오래된 컨트롤은 왜 비주얼 C++에서 사용하기 어려울까?

## ■ 연습문제

이번 장에서 만든 애플리케이션을 수정해서 컬럼 헤더를 더블클릭하면 그 컬럼이 그 리드의 첫번째 컬럼이 되도록 만들자.