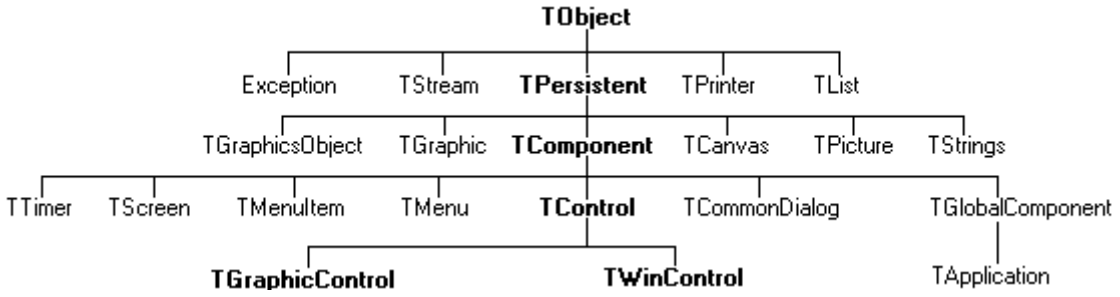
비주얼 컴포넌트 라이브러리의 이해

(Understandings of Visual Component Library)

델파이의 컴포넌트는 OOP 의 관점에서 보면 진정한 객체라고 말할 수 있다. 델파이의 컴포넌트는 데이터의 세트와 데이터 접근 함수를 캡슐화 하며, 조상 컴포넌트의 기능과 데이터를 상속하고, 동일한 조상에서 상속받은 서로 다른 객체 들이 다양하게 동작하는 다형성을 완벽하게 지원한다.
이번 장에서 델파이의 VCL 라이브러리의 구조에 대해서 알아 보고, 이를 활용하여 어플리케이션을 제작하는 방법에 대해서 알아본다. 기본적인 어플리케이션의 제작 방법은 이미 간단히 알아본 바 있으므로 구체적인 컴포넌트의 사용 방법은 독자들이 도움말과 데모 어플리케이션을 이용해서 익히기 바란다.
그리고, 이 장의 후반부에는 비교적 고급스러운 주제인 드래그-드롭(drag-and-drop)을 구현하는 방법과 델파이 4 에서 새롭게 지원되는 드래그-도크(drag-and-dock)의 구현, 액션 리스트를 이용하는 방법에 대해서 알아본다.

VCL 객체 계층 구조

VCL 은 Visual Component Library 의 약자로서, 컴포넌트라고 부르는 객체들로 구성된 라이브러리를 가리킨다. VCL 은 델파이의 오브젝트 파스칼로 작성되었으며, 개별적인 파일이 아니라 하나의 라이브러리에 저장되어 있다. 이들은 나중에 어플리케이션 실행 파일의 일부분으로 사용된다.
VCL 에 대해 잘 이해하려면 먼저 VCL 의 전체적인 계층구조에 대해서 알아야 한다. VCL 의 계층 구조의 핵심 부분은 다음과 같다.



여기서 주의해서 볼 부분은 가운데의 축을 이루는 TObject, TPersistent, TComponent, TControl 클래스이다. 이들은 델파이 VCL 계층 구조에서 가장 핵심을 이루는 클래스 들이다. 이들의 특징에 대해서 이번 장에서 알아볼 것이다.

간략하게 설명하면 TObject 클래스는 델파이의 모든 객체의 기초가 되는 클래스이다. 즉, 모든 델파이 객체는 TObject 클래스를 상속한다. TPersistent 클래스는 TObject 클래스에 추가적인 데이터를 저장할 때 사용자 정의 메소드를 사용하는 것을 허용한다. 다시 말해 값을 저장하고, 불러오는 등의 지속성(persistency)을 지원한다.
TComponent 클래스는 메소드와 프로퍼티를 지원하며 컴포넌트 팔레트에 나타나고, 다른 컴포넌트를 소유할 수 있는 등의 델파이에서 사용하는 각종 컴포넌트의 기본적인 행동을 정의하는 클래스이다.
델파이의 컴포넌트는 시각적인 컴포넌트와 비시각적인 컴포넌트로 나눌 수 있는데, 여기에서 시각적인 컴포넌트를 컨트롤이라고 부른다. 이런 컨트롤의 기본적인 특징을 정의하는 클래스가 TControl 이다. 그래서, TControl 이 아닌 TComponent 를 계승한 컴포넌트를 특히 비시각적 컴포넌트라고 한다.
컨트롤은 폼 위에 존재하면서 사용자에게 정보를 보여 주거나 또는 폼과 상호작용하는 컴포넌트이다. 특별한 컨트롤로는 윈도우 컨트롤이 있다. 윈도우 컨트롤에는 윈도우 핸들이 있는데, 윈도우는 핸들을 이용하여 윈도우 컨트롤에 메시지를 전달한다. 만약 윈도우 핸들을 가지고 있지 않은 컨트롤을 그래픽 컨트롤이라고 한다. 여기서 TWindowControl 로부터 상속받은 컨트롤을 윈도우 컨트롤이라고 하고, TGraphicControl 로부터 상속받은 컨트롤을 그래픽 컨트롤이라고 한다.

- TObject 클래스

델파이의 모든 클래스는 TObject 클래스의 서브 클래스이다. 즉, 모든 객체가 하나의 조상을 가지는 것이다. 그렇기 때문에, 개발자는 시스템의 어떤 클래스의 데이터 형이든 TObject 데이터 형으로 대체하여 사용할 수 있다. 가장 흔히 TObject 를 구경할 수 있는 것은 이벤트 핸들러의 파라미터로 사용하는 'Sender' 이다. 이 파라미터는 보통 TObject 형으로 선언되어 있는데, 이는 어떤 클래스의 객체이든 Sender 객체로 사용할 수 있다는 것을 의미한다.
이렇게 TObject 를 사용할 경우 객체에 대한 작업을 할 때 그 데이터 형을 알아내야 하는 경우가 생긴다. 즉, TObject 형의 변수가 있으면 이 변수는 TObject 에 의해 정의된 메소드와 프로퍼티만 변경할 수 있다. 예를 들어, 이 변수가 TComponent 와 같이 TObject 의 자식 클래스 객체를 참조하게 되면 TObject 클래스에 없는 프로퍼티나 메소드에는 직접 접근할 수가 없다.
이를 해결하기 위해서는 RTTI 연산자인 as 와 is 를 사용하여 데이터 형을 검사하고 데이터 변환을 해야 한다. 예를 들어 TObject 형의 Sender 파라미터가 에디트 박스를 가리킨다면 다음과 같이 쓸 수 있다.

```
(Sender as TEdit).Text := '재밌다 !';
```

이렇게 형변환을 해서 사용하는 방법은 형변환이 실패하면 델파이가 예외를 발생시키므로, 적절한 예외 처리가 가능하다.

> 참고: 런타임 타입 정보(Run-Time Type Information, RTTI)에 대하여
>
> 델파이는 클래스의 가상 메소드 테이블(Virtual Method Table, VMT)내에 모든 클래스에 대한 몇 가지 데이터 형 정보를 저장한다. VMT 와 published 타입 정보는 실행시 이용 가능하다. 그래서 이를 런타임 타입 정보라고 한다.
> 각 클래스는 자신만의 고유한 VMT 를 가지고 있으며, 델파이는 클래스 VMT 포인터 들을 검사하여 클래스들을 구분한다. 한 클래스의 모든 인스턴스들은 해당 클래스의 VMT 를 공유하게 된다.

- TPersistent 클래스

TPersistent 클래스는 다른 객체에 대입될 수 있는 모든 행동을 캡슐화한 클래스이다. 그리고 이런 속성을 스트림에 저장하고, 읽을 수 있어야 한다. 이를 위해 TPersistnet 클래스는 다음과 같은 특징을 가지는 메소드 들을 지원하며, 이들을 적절히 오버라이드하여 사용해야 한다.

- publish 되지 않은 데이터를 스트림에 저장하고, 읽을 수 있는 프로시저를 정의한다.
- 값을 프로퍼티에 대입할 수 있는 방법을 제공한다.
- 하나의 객체의 내용을 다른 객체에 대입할 수 있는 방법을 제공한다.

TPersistent 클래스는 컴포넌트가 아닌 객체를 기초 클래스로 선언하는데, 값을 스트림에 저장하거나 대입 기능을 추가할 필요가 있을 때 가장 적합한 기초 클래스가 된다.
또한 TPersistent 클래스는 published 클래스를 생성시킬 때 많이 사용된다. 예를 들어, 가장 대표적으로 TComponent 는 TPersistent 를 상속한다. 따라서 모든 컴포넌트는 published 되며, 하나의 published 섹션을 가질 수 있다. 또한 모든 컨트롤과 폼은 TComponent 와 TPersistent 로부터 상속받기 때문에 published 섹션을 가질 수 있다.
델파이는 TPersistent 로부터 계승된 여러 개의 클래스를 가지고 있다. 따라서, 그 클래스들은 프로퍼티 데이터 형으로 이용될 수 있다. 예를 들어 TStrings, TFont, TBrush 등이 여기에 해당된다. 반면에 TList 는 TObject 로부터 계승되며 published RTTI 를 가지지 않기 때문에 published 프로퍼티의 데이터 형으로 사용할 수 없다.

- TComponent 클래스

TComponent 클래스는 델파이에서 사용하는 모든 컴포넌트의 기본적인 특징을 지원하는 기초 클래스이다. TComponent 클래스가 지원하는 기능은 다음과 같다.

- 컴포넌트 팔레트에 나타날 수 있으며, 폼 디자이너에서 조작할 수 있다.
- 다른 컴포넌트를 소유하고 다룰 수 있다.
- 향상된 스트림 지원과 파일 조작 기능
- 액티브 X 컨트롤 등의 인터페이스를 구현하는 객체에 대한 wrapper 로서의 기능

비시각적 컴포넌트는 일반적으로 TComponent 로부터 직접 상속받는다. 이러한 컴포넌트는 설계시에는 사용할 수 있도록 보이지만, 실행시에는 보이지 않으면서 백그라운드에서 유용한 서비스들을 수행한다. TTimer 는 비시각적 컴포넌트이고, TField 와 같은 데이터베이스 관련 컴포넌트들도 대부분 비시각적 컴포넌트이다.

- TControl 클래스

컨트롤이란 비주얼 컴포넌트를 의미하는 것으로 사용자가 런타임에서 이들을 직접 보고, 조작할 수 있다. 모든 컨트롤들은 공통적인 프로퍼티, 메소드, 이벤트를 가지고 있으며, 여기에는 컨트롤의 위치와 커서, 힌트 등에 대한 프로퍼티와 컨트롤을 움직이거나 칠하는 메소드와 마우스 행동에 반응하는 이벤트 등이 있다.
컨트롤에는 윈도우 컨트롤과 그래픽 컨트롤이 있는데 이들의 차이점은 다음과 같다.

1. TWinControl

TWinControl 은 입력 포커스를 받아야 하거나, 키보드 입력이 있어야 할 경우에 사용하는 컨트롤이다. 대표적인 컴포넌트로는 TEdit 컨트롤을 들 수 있다. 또한 윈도우 컨트롤은 다른 윈도우나 컨트롤을 담을 수 있다.

2. TGraphicControl

TGraphicControl 컴포넌트는 객체는 포커스를 가지지 않으며, 키보드에 반응하지 않는다. 그렇기 때문에 윈도우에 대한 부담이 줄어들기 때문에 가볍다. 대표적인 그래픽 컨트롤로는 TImage 컴포넌트를 들 수 있다. 또한 TSpeedButton 컨트롤도 그래픽 컨트롤인데, 스피드 버튼은 마우스로만 접근하며 특별히 포커스를 가지지도 않기 때문에 어찌 보면 당연한 것이다.

그래픽 컨트롤과 윈도우 컨트롤 사이의 차이점은 컴포넌트에게 메시지를 전송하고자 할 때 매우 중요하다. 어떤 객체에게나 그 객체의 디스패치(dispatch) 메소드를 호출하여 메시지를 보낼 수 있다. 그러나 윈도우 컨트롤에게는 SendMessage, PostMessage 를 사용한다. 따라서 메시지를 받기 위해서는 윈도우 핸들이 필요하다.

VCL 의 공통적인 프로퍼티

모든 비주얼 컴포넌트에는 공통적으로 사용하는 많은 수의 프로퍼티들이 있다. 한번 비주얼 컴포넌트에 대해서 이해를 하면 많은 수의 프로퍼티가 비슷하다는 것을 쉽게 알 수 있다. 그러면, 이들 중에서 몇가지에 대해 알아보도록 하자.

- Name 프로퍼티

모든 델파이 컴포넌트들은 적절한 이름을 가지고 있다. 이름은 owner 컴포넌트 안에서 각각 유일한 것이어야 하는데, 여기서 owner 컴포넌트는 일반적으로는 컴포넌트를 올려놓는 폼을 말한다. 이것은 어플리케이션이 두 개의 서로 다른 폼을 가질 수 있으며, 그 각 폼들은 같은 이름을 가지는 컴포넌트를 가질 수 있다는 것을 의미한다.
컴포넌트의 Name 프로퍼티 값은 폼 클래스의 선언 안에서 객체의 이름을 정의하는데 사용된다. 이것은 객체를 가리키기 위해 일반적으로 코드 안에서 사용하게 되는 이름이다. 그러므로, 이 값은 파스칼의 변수 이름 규칙에 맞아야 한다.

- 위치와 크기 프로퍼티

폼 위에 있는 컨트롤의 크기와 위치를 정의하기 위해 사용되는 프로퍼티에는 다음의 4 가지가 있다.

1. 수직 크기를 나타내는 Height
2. 수평 크기를 나타내는 Width
3. 위쪽 끝에서의 위치를 나타내는 Top
4. 좌측 끝에서의 위치를 나타내는 Left

이들 프로퍼티는 비시각적 컴포넌트에서는 제공되지 않는다. 그리고, 대부분 폼 디자이너에서 마우스를 가지고 조절할 때 자동으로 설정되지만 경우에 따라서 런타임에서 직접 값을 대입하기도 한다.

- 활성화(Activation), 시각화(Visibility) 속성

사용자들로 하여금 컴포넌트를 활성화하거나 감추도록 하는데 사용할 수 있는 2 가지 기본 프로퍼티가 있다. Enabled 프로퍼티는 입력을 하지 못하게 한다고 생각하면 된다. 이 프로퍼티를 False 로 설정하면 디자인 시에는 그 변화가 눈에 보이지 않지만, 런타임에서는 흐리게 표시된다. 어떤 경우에는 아예 컨트롤을 눈에 보이지 않게 해버리고 싶을 때가 있는데, 이럴 때에는 Hide 메소드를 사용하거나 Visible 프로퍼티를 False 로 설정한다.
여기서 주의할 것은 Visible 프로퍼티를 가지고 그 컨트롤이 보이는지를 판단하면 안된다는 것이다. 만약 컨트롤의 컨테이너가 감추어지면, 그 컨트롤의 Visible 프로퍼티의 내용에 상관없이 그 내용을 볼 수 없다. 이럴 때에는 Showing 프로퍼티를 사용하여 현재 그 컨트롤이 보이는지 알 수 있다.

- 디스플레이 프로퍼티

컨트롤의 전반적인 형태를 결정하는 프로퍼티에는 다음과 같은 것들이 있다.

1. 컨트롤의 경계부위의 형을 설정하는 BorderStyle
2. 컨트롤의 배경색을 결정하는 Color
3. 컨트롤이 3D 형태를 가질 것인지를 결정하는 Ctrl3D
4. 폰트의 경우 색상, 이름, 크기 등을 color, name, style 을 통해 모두 설정할 수 있다.

- Parent, Owner 프로퍼티

어플리케이션의 컨트롤의 모양을 컨트롤을 담고 있는 컨테이너의 전반적인 색상, 폰트 등과 일치를 시키기 위해서는 ParentColor, ParentFont, ParentCtrl3D 등의 프로퍼티를 True 로 설정하면 된다. 보통 기본적으로 이들 프로퍼티는 True 로 설정되어 있는데 Font, Color, Ctrl3D 프로퍼티를 설정하게 되면 이런 Parent 프로퍼티 들의 설정은 False 로 바뀌게 된다.
그리고 프로퍼티 중에 컴포넌트의 Parent 와 Owner 를 결정하는 Parent, Owner 프로퍼티도 있다.
Parent 프로퍼티는 비시각적 컴포넌트 이외의 컨트롤 사이의 시각적 관계를 나타낸다. 델파이의 개발 환경은 Parent 컨트롤을 TPanel, TGroupBox 와 같은 특별히 Parent 컨트롤로 사용하려고 만든 컨트롤들로 제한하고 있다. TPanel 컨트롤 위에 컴포넌트를 놓으면 델파이는 자동으로 새 컴포넌트의 Parent 프로퍼티를 TPanel 로 설정한다. 그리고, Parent 컨트롤은 Child 컨트롤 들을 .DFM 파일에 기록한다.
Owner 프로퍼티는 일반적으로 메모리 관리에 사용된다. 시각적이든 비시각적이든 모든 컴포넌트는 Owner 프로퍼티를 가지고 있다. 컴포넌트가 자신의 Free 메소드 호출에 의해서

제거될 때, 자동으로 컴포넌트의 Owner 가 된다. 따라서 폼이 제거되면 폼 안의 모든 컴포넌트도 같이 제거된다. 보통 대부분의 컴포넌트의 Owner 는 폼이 된다.

- 드래그-드롭(Drag-and-drop) 프로퍼티

드래그-드롭을 지원하는 프로퍼티는 드래그가 어떻게 시작되는지 결정하는 DragMode, 드래그할 때 도킹을 지원할 것인지를 결정하는 DragKind, 드래그할 때 커서의 모양을 결정하는 DragCursor 등이 있다. 드래그-드롭에 대해서는 이 장의 후반부에서 자세히 다룰 것이다.

- 드래그-도크(Drag-and-dock) 프로퍼티

DockSite, DragKind, DragMode, FloatingDockSiteClass 등의 컴포넌트 프로퍼티는 드래그-도크를 지원하기 위해 사용되는 프로퍼티이다. 이들에 대해서는 이 장의 후반부에 자세히 다룰 것이다.

- Canvas 프로퍼티

Canvas 프로퍼티는 그래픽 컨트롤의 가장 중요한 프로퍼티로, 윈도우 디바이스 컨텍스트를 캡슐화한다. 이 프로퍼티를 이용하면 저수준의 그리기 함수를 이용할 수 있다. 여기에 대해서는 다른 장에서 더욱 자세하게 알아볼 것이다.

델파이 컴포넌트의 개괄 (Delphi Components Overview)

델파이 컴포넌트 팔레트는 어플리케이션을 작성할 때 사용할 수 있는 많은 컴포넌트로 이루어져 있다. 컴포넌트를 개발자 입맛대로 팔레트에 추가, 삭제, 재배열할 수 있으며 컴포넌트 템플릿을 이용하여 많은 수의 컴포넌트를 추가할 수도 있다.
컴포넌트는 비슷한 기능에 따라 그룹이 나누어진다. 다음에 컴포넌트 팔레트의 페이지 별로 어떤 종류의 컴포넌트를 가지고 있는지 나열해 보았다.

페이지 이름	내 용
Standard	메뉴 등의 표준 윈도우 컨트롤
Additional	비트맵 버튼 등의 추가적인 컨트롤
Win32	윈도우 95/NT 4.0 에서부터 지원하는 공통 컨트롤, 윈도우 98 용 컨트롤
System	DDE, 파일 시스템, 멀티 미디어, 타이머 등의 시스템 레벨의 컴포넌트
Internet	클라이언트/서버 어플리케이션을 관리하는데 도움을 주는 컴포넌트, 다양한 인터넷

	통신 프로토콜 컴포넌트, 저수준 TCP/IP 를 지원하는 소켓 관리 컴포넌트와 웹서버 어플리케이션을 위한 컴포넌트
Data Access	데이터베이스, 테이블, 쿼리, 리포트 등에 대한 비시각적 컴포넌트
Data Controls	시각적인 데이터 컨트롤
Decision Cube	데이터베이스의 정보를 요약하고 이를 다양한 시각으로 보여줄 수 있는 컨트롤
QReport	쉽게 리포트를 작성하는 것을 도와주는 Quick Report 컴포넌트
Dialogs	윈도우 공통 대화상자 컴포넌트
Win 3.1	델파이 1.0 프로젝트와의 호환성을 위한 컴포넌트
Samples	게이지, 컬러 그리드, 스핀 버튼 등의 샘플 사용자 정의 컴포넌트
ActiveX	샘플 액티브 X 컨트롤
MIDAS	멀티-tiered 데이터베이스 어플리케이션을 작성할 때 사용하는 Midas 컴포넌트

이들 각각에 대해서 어떤 컴포넌트가 있는지 알아보도록 하자. 그렇지만 이 책의 성격상 이들에 대한 자세한 사용 방법을 소개하지는 않는다. 자세한 사항은 델파이가 제공하는 도움말을 참고하기 바라며, 델파이 4 에서 새롭게 제공되는 일부 컴포넌트나 사용방법이 많이 알려지지 않은 클래스의 사용방법과 테크닉에 대해서는 이장의 후반부에서 다루게 될 것이다.

- Standard 탭의 컴포넌트들

여기서는 가장 일반적으로 사용되는 컴포넌트 들을 찾아볼 수 있다. 여기에는 다음 그림에서 보듯이 15 개의 컴포넌트들로 구성된다.

그러면, 각각의 컴포넌트의 역할에 대해서 간단하게 알아보도록 하자.

컴포넌트	설 명
TMainMenu	폼의 메뉴바나 드롭다운 메뉴를 작성하는데 사용되는 비시각적 컴포넌트이다.
TPopupMenu	사용자가 오른쪽 마우스 버튼을 눌렀을 때 나타나는 팝업 메뉴를 작성하고자 할 때 사용하는 비시각적 컴포넌트이다.
TLabel	라벨 컴포넌트는 폼이나 다른 컨테이너에 문자열을 나타내도록 하는데 사용한다. 사용자들은 이를 변경할 수 없다. 시각적 컴포넌트이다.
TEdit	Edit 컴포넌트는 사용자로부터 한 줄의 문자열을 입력받는데 사용한다. Edit

	컴포넌트는 문자열을 화면에 나타내는데 사용하기도 한다. 시각적 컴포넌트이다.
TMemo	여러 줄의 텍스트를 입력받고 화면에 나타내기 위하여 사용하는 시각적 컴포넌트이다.
TButton	버튼 컴포넌트는 버튼을 생성하는데 사용하며, 어플리케이션에서 어떤 항목을 선택할 때 사용한다. 시각적 컴포넌트이다.
TCheckBox	사용자가 체크박스를 선택하거나 선택을 취소하는 것이 가능하다. 시각적 컴포넌트이다.
TRadioButton	일련의 선택 항목들을 제시하고, 이들 중에서 하나 만을 선택하도록 할 때 사용한다. 이들 항목들의 수는 제한이 없으며, 폼, 패널 등을 컨테이너로 사용하는 시각적 컴포넌트이다.
TListBox	표준 윈도우 리스트 박스로서, 항목들의 리스트를 만들어서 사용자가 이들 중에 하나를 선택할 수 있는 시각적 컴포넌트이다.
TComboBox	리스트 박스와 유사하지만, 여기에 Edit 컴포넌트의 장점을 추가한 형태이다. 콤보 박스 컴포넌트는 사용자가 하나의 항목을 선택할 수도 있지만, 원하는 텍스트를 직접 입력할 수도 있는 시각적 컴포넌트이다.
TScrollBar	표준 윈도우 스크롤바로서 폼이나 컨트롤들을 스크롤하는데 사용하는 시각적 컴포넌트이다.
TGroupBox	라디오 버튼, 체크 박스 등의 컨테이너로서 관련된 컨트롤들을 하나의 그룹으로 구성하는데 사용되는 시각적 컴포넌트이다.
TRadioGroup	그룹 박스와 라디오 버튼이 조합된 컴포넌트로서 라디오 버튼의 그룹을 생성하고자 할 때 사용한다. 여러 개의 라디오 버튼들을 사용할 수는 있지만, 다른 컨트롤을 사용할 수는 없다. 시각적 컴포넌트이다.
TPanel	컨트롤이나 컨테이너들을 하나의 그룹으로 만들어주는 또 하나의 컨테이너 컴포넌트이다. 시각적 컴포넌트이다.
TActionList	컴포넌트와 컨트롤에 의해서 사용되는 메뉴 아이템과 버튼과 같은 실제 액션(action)의 리스트를 생성하고 관리하는 비시각적 컴포넌트이다.

- Additional 탭의 컴포넌트들

Additional 탭은 다음 그림과 같은 14 개의 컴포넌트들로 이루어져 있다. 이들 컴포넌트의 역할은 다음과 같다.

컴포넌트	설 명
TBitBtn	비트맵 그림을 포함하는 버튼을 만들 때 사용하는 시각적 컴포넌트이다.
TSpeedButton	패널 컴포넌트에 대하여 사용할 수 있도록 만든 특수한 버튼이다. 스피드 버튼은 툴바 등의 특별한 버튼의 그룹에 대해 사용할 수 있다. 시각적 컴포넌트이다.
TMaskEdit	특별한 형식의 데이터나 적당한 문자의 입력을 위해 사용하는 시각적 컴포넌트이다.
TStringGrid	행과 열에 문자열 데이터를 나타내는데 사용하는 시각적 컴포넌트이다.
TDrawGrid	행과 열에 텍스트 이외의 정보를 나타내고자 할 때 사용하는 시각적 컴포넌트이다.
TImage	아이콘, 비트맵, 메타파일 등의 그림을 나타내기 위해 사용하는 시각적 컴포넌트이다.
TShape	사각형, 원 등의 도형을 그리는데 사용하는 시각적 컴포넌트이다.
TBevel	3 차원으로 들어가거나 튀어나온 모양을 그리는데 사용되는 시각적 컴포넌트이다.
TScrollBox	스크롤이 가능한 화면표시 영역을 생성하기 위해 사용하는 시각적 컴포넌트이다.
TCheckListBox	ListBox 와 CheckBox 의 기능을 합쳐놓은 시각적 컴포넌트이다.
TSplitter	어플리케이션에서 사용자가 크기를 조절할 수 있도록 만들어 주는 패널을 만드는데 사용된다. 시각적 컴포넌트이다.
TStaticText	Label 컴포넌트와 유사하지만, 테두리 유형을 설정할 수 있는 등의 추가적인 기능을 가지고 있다. 시각적 컴포넌트이다.
TControlBar	툴바 컴포넌트 들의 형태를 유지하는데 역할을 한다. 툴바 컴포넌트의 도킹 site 로 사용되는 시각적 컴포넌트이다.
TChart	TeeChart 에서 제공되는 컴포넌트로, 차트나 그래프를 생성할 때 사용되는 시각적 컴포넌트이다.

- Win32 탭

Win32 탭에는 윈도우 95/98/NT 4.0 과 유사한 어플리케이션을 작성하는데 필요한 18 개의 컴포넌트가 제공된다. 이들 중의 몇 가지는 Win3.1 탭의 컴포넌트와 유사하다. 이들 컴포넌트의 역할은 다음과 같다.

컴포넌트	설 명
TTabControl	윈도우 95 형식의 탭 컴포넌트로서 폼에 사용자가 선택할 수 있는 탭들을 추가하기 위해서 사용하는 시각적 컴포넌트이다.
TPageControl	윈도우 95 형식의 컴포넌트로서, 탭이나 다른 컨트롤로 바꿀 수도 있으며, 화면의 공간을 절약하여 사용할 수 있다. 시각적 컴포넌트이다.
TImageList	이미지의 리스트를 관리하기 위한 새로운 객체이다.
TRichEdit	윈도우 95 형식의 컴포넌트로서, 여러가지 색깔이나 폰트, 텍스트 검색 등을 지원하는 향상된 메모 컴포넌트라고 생각하면 된다. 시각적 컴포넌트이다.
TTrackBar	윈도우 95 형식의 슬라이더(silder) 컨트롤이다. 시각적 컴포넌트이다.
TProgressBar	윈도우 95 형식의 진행상태를 나타내는 바로서 시각적 컴포넌트이다.
TUpDown	윈도우 96 형식의 스핀 버튼 컨트롤이다. 시각적 컴포넌트이다.
THotKey	어플리케이션에 추가적인 단축 키를 지원하도록 하기 위해 사용하는 시각적 컴포넌트이다.
TAnimate	윈도우 95 에서 파일을 복사 또는 이동할 때 나타나는 것과 같은 AVI 동영상 클립을 재생시켜 주기 위하여 사용하는 시각적 컴포넌트이다.
TDataTimePicker	ComboBox 와 유사한 컴포넌트로서, 달력을 나타내어 데이터를 선택하도록 하는 시각적 컴포넌트이다.
TMonthCalendar	사용자가 날짜나 날짜의 범위를 선택할 수 있는 달력 컴포넌트이다. 시각적 컴포넌트이다.
TTreeView	윈도우 95 형식의 컴포넌트로서, 데이터를 계층 구조의 형식으로 나타내는데 사용되는 시각적 컴포넌트이다.
TListView	윈도우 95 형식의 컴포넌트로서, 리스트 들을 이미지를 가진 열(column)로 나타내는데 사용하는 시각적 컴포넌트이다.
THeaderControl	윈도우 95 형식의 컴포넌트로서, 여러 개의 이동이 가능한 헤더를 생성하기 위해 사용하는 시각적 컴포넌트이다.
TStatusBar	윈도우 95 형식의 컴포넌트로서, 상황에 대한 정보를 여러 개의 패널에 나누어 나타나도록 하는데 사용하는 시각적 컴포넌트이다.
TToolBar	어플리케이션에서 자주 사용되는 기능을 빠르게 사용할 수 있도록 하기 위한 툴바를 생성하기 위해 사용되는 시각적 컴포넌트이다.
TCoolBar	사용자가 크기 조정을 할 수 있도록 밴드를 컴포넌트에 추가한 시각적 컴포넌트이다.
TPageScroller	툴바와 같이 좁은 윈도우 영역의 디스플레이 방법을 정의하는 시각적 컴포넌트이다. 윈도우가 디스플레이 영역보다 큰 경우, 화살표를 윈도우의 끝 부분에 표시하여 디스플레이 영역을 스크롤하여 볼 수 있도록 해준다.

- VCL 의 System 탭

System 탭에는 윈도우의 장점을 최대한으로 이용하는데 도움을 주는 다음 그림과 같은 8 개의 컴포넌트로 이루어져 있다. 이들 컴포넌트의 역할은 다음과 같다.

컴포넌트	설 명
TTimer	일정한 시간 간격으로 프로시저나 함수, 이벤트 들을 사용하는데 사용되는 비시각적 컴포넌트이다.
TPaintBox	그림을 그릴 수 있는 폼의 영역에서 그림을 그리게 위해 사용하게 되는 시각적 컴포넌트이다.
TMediaPalyer	VCR 과 같은 형태의 패널를 생성하여, 소리나 비디오 파일 들을 재생하는데 사용하는 시각적 컴포넌트이다.
TOleContainer	OLE 클라이언트 영역을 생성하는데 사용하는 시각적 컴포넌트이다.
TDDEClientConv	DDE 서버와의 통신을 설정하는데 사용되는 비시각적 컴포넌트이다.
TDDEClinetItem	DDE 통신 중에 DDE 서버에게 전송하고자하는 클라이언트 데이터를 지정하는데 사용하는 비시각적 컴포넌트이다.
TDDEServerConv	DDE 서버 어플리케이션이 DDE 클라이언트와 통신을 하는데 사용되는 비시각적 컴포넌트이다.
TDDEServerItem	통신 중에 DDE 클라이언트에 전송하고자하는 데이터를 지정하는데 사용하는 비시각적 컴포넌트이다.

- Internet 탭

인터넷 탭에는 인터넷이나 TCP/IP 네트워크 환경의 어플리케이션을 작성하기 위한 작업을 도와주는 26 개의 컴포넌트 들이 포함되어 있다. 이들은 다음과 같은 기능을 한다.

컴포넌트	설 명
TClientSocket	네트워크 상의 다른 기계와 연결을 생성하기 위하여 사용된다. 소켓 프로그래밍을 저수준으로 할 수 있도록 지원하는 비시각적 컴포넌트이다.
TServerSocket	네트워크 상의 다른 기계로부터의 클라이언트 요청에 응답하기 위해 사용

	된다. 소켓 프로그래밍을 지원하는 비시각적 컴포넌트이다.
TWebDispatcher	일반적인 데이터 모듈을 웹 모듈로 변환시키는데 사용되는 비시각적 컴포넌트이다.
TPageProducer	웹 브라우저나 다른 HTML 뷰어 어플리케이션에서 볼 수 있도록, HTML 템플릿을 HTML 문자열 코드로 변환시키는데 사용되는 비시각적 컴포넌트이다.
TQueryTableProducer	TQuery 객체로부터 HTML 테이블을 생성하는데 사용되는 비시각적 컴포넌트이다.
TDataSetTableProducer	TDataSet 객체의 레코드들로부터 HTML 테이블을 생성하는데 사용된다.
TDataSetPageProducer	필드 데이터를 포함한 HTML 템플릿에 기초한 HTML 문자열 코드를 생성하는데 사용되는 비시각적 컴포넌트이다.
TNMDatTime	인터넷 daytime 서버에서 날짜와 시간을 얻어오는데 사용되는 비시각적 컴포넌트이다.
TNMEcho	인터넷 echo 서버에 텍스트를 전송하는데 사용되는 컴포넌트로, 전송하는 텍스트는 다시 돌아온다. 주로 네트워크 무결성과 속도를 측정하기 위해 사용되는 비시각적 컴포넌트이다.
TNMFinger	사용자에 대한 정보를 인터넷 finger 서버에서 얻어오는데 사용되는 비시각적 컴포넌트이다.
TNMFTP	파일을 인터넷 FTP 서버에 FTP 프로토콜을 이용하여 전송할 때 사용되는 비시각적 컴포넌트이다.
TNMHTTP	인터넷을 통해 HTTP 전송을 수행하는데 이용되는 비시각적 컴포넌트이다.
TNMMsg	인터넷에 TCP/IP 프로토콜을 이용해서 간단한 ASCII 텍스트 메시지를 전송하는데 사용되는 비시각적 컴포넌트이다.
TNMMsgServ	TNMMsg 컴포넌트가 전송한 메시지를 받아서 이를 사용할 수 있도록 해주는 비시각적 컴포넌트이다.
TNMNNTP	인터넷 뉴스 기사를 뉴스 서버에서 읽거나, 전송하는데 사용되는 비시각적 컴포넌트이다.
TNMPOP3	인터넷 E-mail 을 POP3 서버에서 받아올 때 사용되는 비시각적 컴포넌트이다.
TNMUUProcessor	MIME 또는 UUEncode 된 파일을 코딩화 하거나 해독화할 때 사용되는 비시각적 컴포넌트이다.
TNMSMTP	인터넷 메일 서버에 E-mail 을 전송할 때 사용되는 비시각적 컴포넌트이다.
TNMStrm	인터넷을 통해 스트림 서버에 스트림을 전송하는데 사용되는 비시각적 컴포넌트이다.

TNMStrmServ	TNMStrm 컴포넌트에서 전송한 스트림을 받아서 사용할 수 있도록 해주는 스트림 서버 컴포넌트이다. 비시각적 컴포넌트이다.
TNMTime	TNMDateTime 컴포넌트와 비슷하나, 시간을 얻어올 뿐이다.
TNMUDP	UDP 프로토콜을 이용하여 데이터 그램 패킷을 인터넷에 전송할 수 있도록 구현된 비시각적 컴포넌트이다.
TPowerSock	Internet 탭에 있는 많은 컴포넌트의 기초가 되는 클래스로, 구현되지 않은 다른 프로토콜이나 사용자 정의 프로토콜을 구현할 때 사용할 수 있는 비시각적 컴포넌트이다.
TNMGeneralServer	멀티 쓰레드 인터넷 서버를 개발하기 위한 기초 클래스로 제공되는 컴포넌트이다. RFC 표준을 따르는 서버이든, 사용자가 나름대로 서비스를 제공하는 서버이든 이를 구현하는데 유용하게 사용할 수 있는 비시각적 컴포넌트이다.
THTML	웹 브라우저 등에서 사용하는 HTML 코드의 내용을 HTML 페이지로 나타내어 주는데 사용되는 시각적 컴포넌트이다.
TNMURL	URL 데이터를 읽기 쉬운 문자열로 해독하거나, 표준 문자열을 URL 데이터 포맷으로 코딩화하는 작업을 하는 비시각적 컴포넌트이다.

- Data Access 탭

Data Access 탭은 다음과 같이 데이터베이스 연결하고 통신하는데 사용되는 9 개의 컴포넌트로 이루어져 있다. 이들의 역할은 다음과 같다.

컴포넌트	설 명
TDataSource	테이블이나 쿼리 컴포넌트들을 데이터 컨트롤과 연결시키기 위하여 사용하게 된다. 비시각적 컴포넌트이다.
TTable	데이터베이스 테이블을 어플리케이션에 연결시켜 주기 위하여 사용하는 비시각적 컴포넌트이다.
TQuery	원격 SQL 서버나 지역 데이터베이스에 대하여 SQL 쿼리를 실행시키거나 생성하는데 사용하는 비시각적 컴포넌트이다.
TStoredProc	SQL 서버에 저장되어 있는 프로시저를 실행시키기 위해 사용하는 비시각적 컴포넌트이다.
TDatabase	원격 데이터베이스 서버와 연결을 하기 위하여 사용하는 비시각적 컴포넌

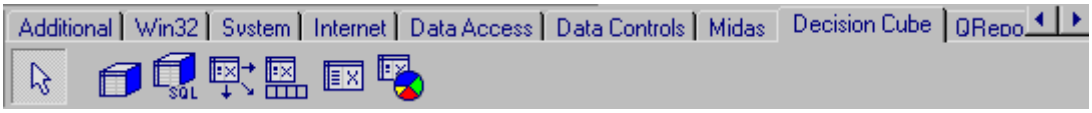
	트이다.
TSession	어플리케이션의 데이터베이스 연결에 대한 전반적인 제어를 위하여 사용하는 비시각적 컴포넌트이다.
TBatchMove	지역적으로 작업한 레코드와 테이블 들로 서버로 올려서 서버의 정보를 갱신하는데 사용하는 비시각적 컴포넌트이다.
TUpdateSQL	SQL 데이터베이스를 갱신하기 위해 사용되는 비시각적 컴포넌트이다.
TNestedTable	오라클 8 에서부터 지원되는 중첩된 테이블을 지원하는 비시각적 컴포넌트이다.

- Data Controls 탭

DataControls 탭은 15 개의 데이터 컨트롤을 모아 놓은 페이지이다. 이들 대부분은 Standard 나 Additional 탭에서 볼 수 있는 컴포넌트들을 데이터베이스와 함께 사용할 수 있도록 수정한 것이다. 특별히 설명할 것이 없으므로 이들에 대한 설명은 생략하겠다.

- Decision Cube 탭

Decision Cube 탭은 데이터 분석에 유용하게 사용할 수 있는 6 개의 다차원 차트와 그래프 컴포넌트를 포함하고 있다. 이들에 대해서 간단히 알아보면 다음과 같다.



컴포넌트	설 명
TDecisionCube	다차원의 데이터를 저장하는 저장소로서, 데이터 집합으로부터 데이터를 불러들이기 위해 사용되는 비시각적 컴포넌트이다.
TDecisionQuery	TQuery 컴포넌트를 DecisionCube 에서 사용할 수 있도록 수정한 비시각적 컴포넌트이다.
TDecisionSource	DecisionGrid 나 DecisionGraph 컴포넌트의 현재 피봇(pivot) 상태를 정의하기 위해 사용하는 비시각적 컴포넌트이다.
TDecisionPivot	버튼을 통하여, DecisionCube 의 차원이나 필드를 열거나 닫도록 하는데 사용되는 시각적 컴포넌트이다.

TDecisionGrid	DecisionCube 컴포넌트의 데이터를 그리드의 형식으로 나타내는데 사용하는 시각적 컴포넌트이다.
TDecisionGraph	DecisionCube 컴포넌트의 데이터를 그래프로 나타내는데 사용되는 시각적 컴포넌트이다.

- Midas 탭

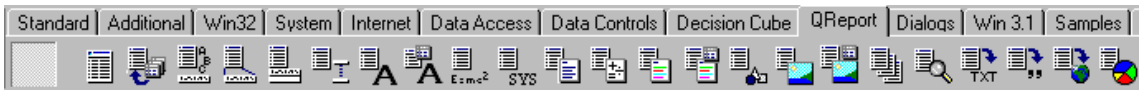
Midas 탭의 컴포넌트 들은 멀티-tiered 데이터베이스 어플리케이션 작성에 필요한 10 개의 컴포넌트로 구성되어 있다. 이들의 역할에 대한 간단한 설명을 하면 다음과 같다.

컴포넌트	설 명
TClientDataSet	데이터베이스에 독립적인 데이터 세트로 1-tiered 어플리케이션을 작성하거나, 멀티-tiered 데이터베이스 어플리케이션을 사용할 때 사용되는 비시각적 컴포넌트이다.
TDCOMConnection	멀티-tiered 데이터베이스 어플리케이션에서 DCOM 원격 서버에 접속하는 방법을 제공하는 비시각적 컴포넌트이다.
TCorbaConnection	멀티-tiered 데이터베이스 어플리케이션에서 CORBA 원격 서버에 접속하는 방법을 제공하는 비시각적 컴포넌트이다.
TSocketConnection	멀티-tiered 데이터베이스 어플리케이션에서 TCP/IP 원격 서버에 접속하는 방법을 제공하는 비시각적 컴포넌트이다.
TOLEnterpriseConnection	멀티-tiered 데이터베이스 어플리케이션에서 OLEnterprise 원격 서버에 접속하는 방법을 제공하는 비시각적 컴포넌트이다.
TDataSetProvider	데이터 세트의 데이터를 코딩화하여 클라이언트 어플리케이션으로 전송될 패킷을 생성하며, 클라이언트 어플리케이션의 업데이트를 반영하는 역할을 하는 비시각적 컴포넌트이다.
TProvider	원격 서버의 데이터베이스 어플리케이션 서버와 클라이언트 데이터 세트 간에 연결을 제공하는 비시각적 컴포넌트이다.
TSimpleObjectBroker	가능한 어플리케이션 서버의 리스트에서 접속 컴포넌트를 서버에 위치시키는 역할을 하는 비시각적 컴포넌트이다.
TRemoteServer	멀티-tiered 어플리케이션의 원격 서버와 DCOM 접속을 위한 비시각적 컴포넌트이다. 델파이 3 와의 호환성을 위해 사용될 뿐이다.
TMIDASConnection	멀티-tiered 어플리케이션의 원격 서버와 DCOM, TCP/IP, OLEnterprise

	접속을 위한 비시각적 컴포넌트로, 델파이 3 와의 호환성을 위해 사용될 뿐이다.

- QReport 탭

QReport 탭은 리포트의 작성을 위한 23 개의 컴포넌트 들로 구성된다. 이들 컴포넌트의 사용법에 대해서는 40 장에 대해서 자세하게 다루게 될 것이므로 설명은 생략한다.



- Dialogs 탭

Dialogs 탭은 윈도우에서 여러가지 대화상자를 생성하는데 사용되는 10 가지 컴포넌트들로 구성된다. 이들에 대해 간단히 설명하면 다음과 같다.

컴포넌트	설 명
TOpenDialog	파일 열기 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TSaveDialog	파일 저장 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TOpenPictureDialog	그림 열기 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TSavePictureDialog	그림 저장 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TFontDialog	글꼴 선택 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TColorDialog	색깔 선택 대화상자를 생성하는데 사용되는 비시각적 컴포넌트이다.
TPrintDialog	인쇄 대화상자를 생성하는데 사용하는 비시각적 컴포넌트이다.
TPrinterSetupDialog	프린터 설정 대화상자를 생성하는데 사용하는 비시각적 컴포넌트이다.
TFindDialog	찾기 대화상자를 생성하는데 사용하는 비시각적 컴포넌트이다.
TReplaceDialog	바꾸기 대화상자를 생성하는데 사용하는 비시각적 컴포넌트이다.

- Win 3.1 탭

델파이 1.0 과 2.0 간의 어플리케이션 전환을 위한 호환성을 위해 제공되고 있는 컴포넌트들이다. 더 이상 사용하지 않는 것이 좋을 것으로 생각되므로, 이 탭에 대해서는 특별히 따로 설명하지 않는다.

- Samples 탭

Samples 탭에는 6 개의 VCL 들을 포함하고 있지만, 이들은 최소한의 설명과 함께 예제로서 포함되어 있는 것이다. 이들의 소스 코드들은 Delphi 4₩Source₩Samples 디렉토리에서 찾아볼 수 있다. 이들에 대해 간단히 설명하면 다음과 같다.

컴포넌트	설 명
TGuage	사각형과 텍스트 또는 파이 형식으로 진행 정도를 나타내는 게이지를 표시해주는 시각적 컴포넌트이다.
TColorGrid	사용자가 색깔을 선택할 수 있는 그리드를 생성하는데 사용되는 시각적 컴포넌트이다.
TSpinButton	스핀 버튼을 생성하는데 사용되는 시각적 컴포넌트이다.
TSpinEdit	스핀 컨트롤과 Edit 상자 컴포넌트의 기능을 합쳐놓은 형태의 시각적 컴포넌트이다.
TDirectoryOutline	선택된 드라이브의 디렉토리 구조를 나타내는데 사용되는 시각적 컴포넌트이다.
TCalendar	날짜 정보를 나타내는 달력을 화면에 표시하기 위해 사용되는 시각적 컴포넌트이다.
TIBEventAlerter	이벤트 경고 컴포넌트이다. 비시각적 컴포넌트이다.

- ActiveX 탭

ActiveX 탭에는 5 가지 ocx 예제 컨트롤들이 포함되어 있다. 그렇지만, 델파이 프로젝트에는 VCL 컨트롤을 사용하는 것이 장점이 많으며, 거의 사용될 일이 없으므로 이 탭에 대해서는 특별히 따로 설명하지 않는다.

드래그-드롭(Drag-and-Drop)의 구현

드래그-드롭은 일반적인 컨트롤에서는 대부분이 지원하게 되는 특징이다. 먼저 간단하게 드래그-드롭을 구현하는 방법에 대해서 알아보고, 실제 예제를 하나 작성해 보도록 하자.

- 드래그-드롭의 구현 방법

1. 드래그 작업의 시작

모든 컨트롤은 DragMode 라는 프로퍼티를 가지고 있는데, 이 프로퍼티는 사용자가 컴포넌트를 런타임에서 드래그를 시작할 때 어떤 식으로 반응할 지를 결정한다. dmAutomatic 인 경우 끌기는 사용자가 컨트롤 위에서 마우스 버튼을 누르는 순간 자동적으로 시작된다. dmAutomatic 은 정상적인 마우스의 활동을 제한할 수 있기 때문에, 보통은 디폴트 값인 dmManual 을 사용하며 대신 OnMouseDown 이벤트를 사용하여 드래그를 시작한다.
드래그를 시작하려면 컨트롤의 BeginDrag 메소드를 호출하면 된다. BeginDrag 메소드에는 Immediate 라는 Boolean 형의 파라미터가 있는데, 이를 True 로 설정하면 드래그가 즉시 시작되므로, DragMode 를 dmAutomatic 으로 설정한 것과 별반 차이가 없는 것이 된다. 그에 비해 False 로 값을 설정한 경우 사용자가 마우스를 가지고 조금이라도 끌기 전에는 d 드래그가 시작되지 않는다. 그러므로, BeginDrag(False)를 호출하는 것이 일반적인 마우스의 클릭을 드래그 작업으로 간주하지 않게 되므로 더 좋다.
드래그를 시작하기 전에 어떤 버튼이 사용자에 의해 눌려 졌는지 검사함으로써 드래그를 시작할 지 결정하게 되는 경우도 많은데, 이럴 때에는 OnMouseDown 이벤트에서 Button 파라미터를 검사하여 적절한 버튼이라면 드래그를 시작하도록 설정할 수 있다.
다음의 코드는 파일 리스트 박스의 OnMouseDown 이벤트에서 좌측 마우스 버튼이 눌린 경우에만 드래그를 시작하도록 하는 코드이다.

```
procedure TForm1.FileListBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then                          {좌측 버튼일 때에만 drag}
    with Sender as TFileListBox do
    begin
      if ItemAtPos(Point(X, Y), True) >= 0 then    {위치에 Item 이 있으면 ... }
        BeginDrag(False);
    end;
end;
```

2. 드래그된 아이템을 받아들일지 여부 결정

사용자가 아이템을 드래그하여 어떤 컨트롤의 위를 지날 때에 그 컨트롤은 OnDragOver 이벤트를 발생시킨다. 이때 이 컨트롤이 드롭을 지원할 지를 결정하게되는데, 델파이는 컨트롤이 드롭을 지원하는지 여부를 드래그 커서의 모양을 바꿔서 사용자에게 알리게 된다.

드래그를 허용한다면 컨트롤의 OnDragOver 이벤트의 이벤트 핸들러를 작성해야 한다. OnDragOver 이벤트에는 Accept 라는 variable 파라미터가 있는데, 이 값을 True 로 설정하면 아이템을 드롭할 수 있도록 허용한다는 의미가 된다. 만약 이 값을 False 로 설정되면 이 컨트롤이 현재의 아이템에 대한 드롭을 허용하지 않겠다는 의미가 된다.
OnDragOver 이벤트에는 이런 판단을 하는데 도움을 주는 드래그의 source, 마우스 커서의 현재 위치 등의 몇 가지 파라미터가 있다. 다음 코드는 디렉토리 트리뷰에 FileListBox 에서 온 아이템의 드롭 만을 허용하게 된다.

```
procedure TForm.DirectoryOutline1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  if Source is TFileListBox then
    Accept := True;
  else
    Accept := False;
end;
```

3. 아이템의 드롭

일단 컨트롤이 드롭을 허용하면, 실제로 아이템이 드롭되었을 때 이를 처리하는 것은 OnDragDrop 이벤트이다. OnDragOver 이벤트와 마찬가지로 OnDragDrop 이벤트 역시 드래그된 아이템의 source 와 마우스 커서의 위치를 이용하여 여러가지 작업을 하게 된다.

4. 드래그 작업의 종료

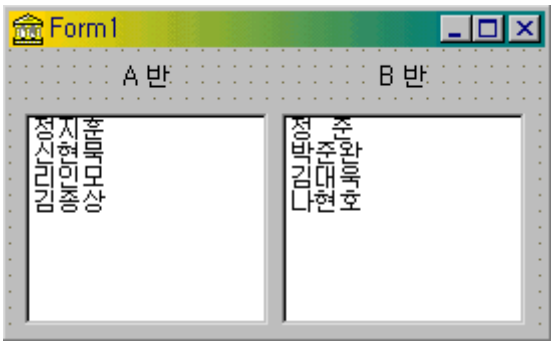
드래그 작업이 끝나면, 드래그가 시작된 컴포넌트에 OnEndDrag 이벤트가 발생한다. OnEndDrag 이벤트의 파라미터 중에서 가장 중요한 것은 Target 을 여기에는 어떤 컨트롤이 드롭을 받았는지가 전달된다. 이 값이 nil 인 경우 드롭을 받아들인 컴포넌트가 없다는 의미이다. 그리고, 파라미터에는 드롭이 일어난 컨트롤의 위치를 나타내는 X, Y 좌표가 전달된다.

5. 드래그 마우스 포인터 변경

드래그를 할 때 마우스의 형태를 변경하려면 해당 컴포넌트의 DragCursor 프로퍼티를 변경해주면 된다.

● 드래그-드롭을 구현한 간단한 예제

간단한 예제로 두 개의 리스트 박스 사이에 문자열을 드래그-드롭으로 이동하도록 해보자. 먼저 폼에 다음과 같이 2 개의 리스트 박스와 2 개의 라벨 컴포넌트를 올려놓고, Label1, Label2 의 Caption 프로퍼티를 각각 'A 반', 'B 반'으로 설정한다. 그리고, ListBox1 과 ListBox2 의 Items 프로퍼티를 편집하여 다음 그림과 같은 내용을 입력한다.



그리고, 코드 에디터에서 드래그-드롭으로 옮겨 다닐 문자열의 내용을 저장할 전역변수인 SelectedText 를 다음과 같이 선언하도록 한다.

```
var
  Form1: TForm1;
  SelectedText: string;
```

그럼, 이제 드래그-드롭을 구현해 보자. 디폴트로 DragMode 프로퍼티는 dmManual 로 설정되어 있으므로, 첫번째로 해야 할 일은 ListBox1 의 OnMouseDown 이벤트 핸들러를 작성하는 것이다. OmMouseDown 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Index: Integer;
begin
  SelectedText := '';                          {선택된 문자열을 초기화}
  if Button = mbLeft then                      {좌측 버튼일 때에만 drag}
    with Sender as TListBox do
    begin
      Index := ItemAtPos(Point(X, Y), True);   {커서 위치의 아이템의 인덱스를 구한다}
```

```
      if Index >= 0 then                          {커서 위치에 Item 이 있으면 ... }
      begin
        BeginDrag(False);
        SelectedText := Items.Strings[Index];     {선택된 문자열을 커서 위치의 아이템으로 설정}
      end;
    end;
end;
```

여기서 주의해서 관찰할 것은 일단 TListBox 의 ItemAtPost 메소드를 이용해서 현재 커서 위치에 아이템이 있는지를 알아본 후, 있다면(Index >= 0) BeginDrag(False) 메소드를 호출하여 드래그를 시작한다는 것이다. 그리고, 그 위치의 아이템의 문자열을 전역변수인 SelectedText 에 대입하여 이 문자열을 드롭할 때 사용하게 된다.
여기서 ItemAtPos 메소드는 첫번째 파라미터에 TPoint 형으로 위치를 전달하고 두번째 파라미터에 True 를 설정하면, 지정된 위치에 아이템이 존재하면 그 아이템의 인덱스를 반환하며, 아이템이 없으면 -1 을 반환한다.
드래그가 시작되었으면, ListBox2 에 드롭을 하게 될 것이므로 드롭을 받아들일 것인지 여부를 OnDragOver 이벤트에서 결정해 주어야 한다. ListBox2 의 OnDragOver 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.ListBox2DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  if (Source as TListBox).Name = 'ListBox1' then Accept := True
  else Accept := False;
  ListBox2.ItemIndex := ListBox2.ItemAtPos(Point(X, Y), True);
end;
```

이 코드에서 Source 파라미터는 드래그가 시작된 컨트롤을 가리킨다. 즉, 드래그가 시작된 컨트롤의 이름이 'ListBox1'인 경우에만 드롭을 받아들이겠다는 의미이다. 그러므로, ListBox1 에서 드래그를 해서 ListBox2 에 드롭하는 것은 허용하겠지만, ListBox2 에서 드래그를 시작하거나 탐색기와 같은 다른 프로그램에서 드롭을 하려고 하면 ListBox2 에서는 받아들이지 않는다는 뜻이다.
마지막 줄의 TListBox 의 ItemIndex 프로퍼티는 선택된 아이템의 위치를 지정하는 것으로, 이 코드는 커서 위치에 따라 선택된 아이템의 위치를 변경하라는 코드이다.
그러면, ListBox2 의 OnDragDrop 이벤트 핸들러에서 드롭이 되었을 때 작업을 마무리하는 이벤트 핸들러를 작성해 보자.

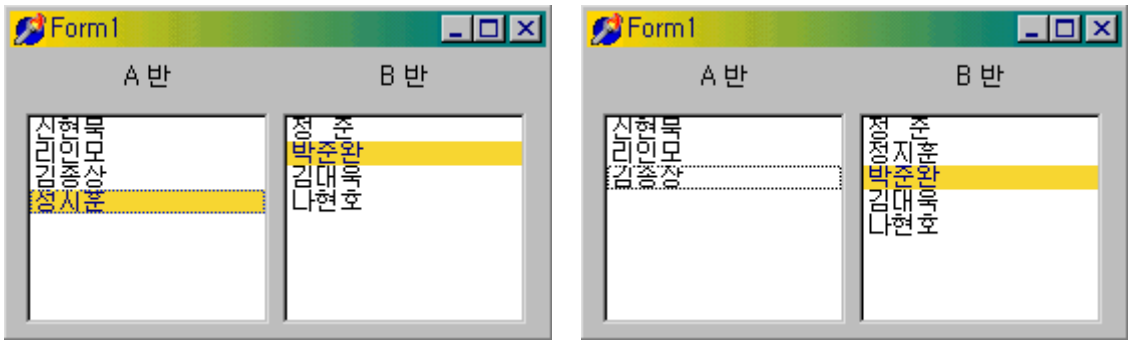
```
procedure TForm1.ListBox2DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  Index: Integer;
begin
  Index := ListBox2.ItemAtPos(Point(X, Y), True);
  if Index >= 0 then ListBox2.Items.Insert(Index, SelectedText)
  else ListBox2.Items.Add(SelectedText);
  ListBox1.Items.Delete(ListBox1.Items.IndexOf(SelectedText));
end;
```

일단 현재 드롭되려는 위치의 아이템의 위치를 ItemAtPost 메소드를 이용해서 Index 변수에 저장한다. Index 변수의 값이 0 보다 크거나 같다면 커서의 위치에 아이템이 존재한다는 의미이므로 Items.Insert 메소드를 이용해서 현재 위치에 SelectedText 변수의 문자열(드래그하는 문자열)을 삽입한다. Index 변수의 값이 0 보다 작다면 커서의 위치에 아이템이 존재하지 않는 것이므로 제일 마지막에 SelectedText 변수의 문자열을 Items.Add 메소드를 이용해서 추가하면 된다.

그리고, 이렇게 추가하고 나면 ListBox1 에서 옮겨온 아이템은 삭제해 주어야 이동이 완료된다. 리스트 박스의 각각의 메소드에 대한 자세한 사용법은 도움말을 참고하기 바란다.

이것으로 ListBox1 에서 ListBox2 로의 드래그-드롭은 모두 구현되었다. ListBox2 에서 ListBox1 으로의 드래그-드롭의 구현도 완전히 똑같은 방법으로 하면 된다. 다만, ListBox1 을 가리키는 것은 ListBox2 로 ListBox2 는 ListBox1 으로 바꾸어 주기만 하면 된다. 이벤트 역시 반대로 적용해서 구현하면 된다. 지면 관계상 같은 내용을 반복해서 설명하지는 않는다. 자세한 것은 소스 코드를 참고하기 바란다.

제대로 했을 것으로 믿고, 실행을 한번 해보자 다음과 같이 드래그 드롭할 수 있으면 제대로 실행된 것이다.



드래그-도크(Drag-and-dock)의 구현

델파이 4 의 TControl 과 TWinControl 은 기본적으로 MS 오피스 97 에서 볼 수 있는 형태의 도킹 윈도우를 지원하게 되었다. 모든 TWinControl 의 자손 컨트롤은 도킹 site 로 사용될 수 있으며, 모든 TControl 의 자손은 도킹 site 에 도킹하는 자식 윈도우로 사용될 수 있다. 예를 들어, 폼의 좌측 면을 도킹 site 로 사용하려면, 패널을 좌측으로 정렬하게 한 뒤 이 패널을 도킹 site 로 만들어 버리면 된다. 이때 도킹이 가능한 컨트롤을 패널 옆으로 끌어다 놓으면 이 윈도우는 패널의 자식 윈도우로 도킹이 된다.

- 드래그-도크의 구현 방법

1. 윈도우 컨트롤을 도킹 site 로 만든다.

윈도우 컨트롤을 도킹 site 로 만들려면 다음과 같이 한다.

- 컨트롤의 DockSite 프로퍼티를 True 로 설정한다.
- 도킹 site 객체가 도킹 클라이언트를 포함할 때에만 나타나게 하려면, AutoSize 프로퍼티를 True 로 설정한다. 이렇게 하면 도킹된 자식 컨트롤이 없을 때에 도킹 site 의 크기가 0 이 되기 때문에 언제나 도킹될 컨트롤의 크기에 맞출 수가 있다.

2. 도킹할 자식 컨트롤을 도킹이 가능하도록 설정한다.

- DragKind 프로퍼티를 dkDock 으로 설정한다. 이렇게 하면 컨트롤을 드래그할 때 도킹 site 근처에 가면 그 위치로 컨트롤이 이동해 가며, 도킹된 것을 떼어낼 때에는 플로팅 윈도우(floating window)가 된다. DragKind 프로퍼티가 dkDrag(디폴트)인 경우에는 드래그할 때 드래그-드롭 작업이 시작된다.
- DragMode 프로퍼티를 dmAutomatic 으로 설정한다. 이렇게 하면, 사용자가 컨트롤을 마우스로 드래그하기 시작하면 자동으로 드래그 작업으로 들어간다. 이 프로퍼티가 dmManual 로 설정된 경우에는 BeginDrag 메소드를 호출해야 한다.
- FloatingDockSiteClass 프로퍼티는 컨트롤이 도킹에서 떨어질 경우나 플로팅 윈도우가 되었을 때 컨트롤의 주인이 될 TWinControl 의 자손 클래스로 설정한다. 컨트롤이 도킹 site 에서 떨어져 나오면, 이 클래스의 윈도우 컨트롤이 동적으로 생성되서, 도킹 윈도우의 부모가 된다. 도킹 윈도우가 TWinControl 의 자손일 경우에는 컨트롤의 주인이 될 분리된 플로팅 도킹 site 를 생성할 필요는 없다. 이럴 때에는 컨트롤과 같은 클래스로 지정하면, 플로팅 윈도우가 될 때 특별한 부모가 없이 생성된다.

3. 컨트롤이 도킹 site 에 도킹하는 방법을 조정한다.

도킹 site 는 자식 컨트롤이 놓여질 때 자동으로 이를 받아들인다. 대부분의 컨트롤에서 보면 첫번째 자식이 클라이언트 영역을 모두 채우게 도킹된 경우, 두번째 자식은 영역을 분리시키곤 한다. 이렇게 세밀하게 도킹 site 에 자식 윈도우가 도킹하는 방법을 제어하기 위해 OnGetSiteInfo, OnDockOver, OnDockDrop 의 3 가지 이벤트가 사용된다. 이들 이벤트에 대해 잠시 알아보도록 하자.
먼저 OnGetSiteInfo 이벤트의 선언부와 역할에 대해서 알아보자. OnGetSiteInfo 이벤트의 선언부는 다음과 같다.

```
property OnGetSiteInfo: TGetSiteInfoEvent;
TGetSiteInfoEvent = procedure(Sender: TObject; DockClient: TControl; var InfluenceRect: TRect;
  var CanDock: Boolean) of object;
```

OnGetSiteInfo 이벤트는 도킹 site 에 도킹될 자식 컨트롤을 사용자가 드래그하면 발생하게 되는데, 여기에서 site 가 DockClient 파라미터에 지정된 컨트롤을 받아들일 것인지 여부와 받아들인다면 자식 컨트롤이 도킹할 위치를 지정할 수 있다. OnGetSiteInfo 이벤트가 발생하면 InfluenceRect 는 도킹 site 의 스크린 위치로 설정되며, CanDock 파라미터는 True 로 초기화된다. 만약 경우에 따라서 자식 컨트롤의 도킹을 허용하지 않으려면 CanDock 를 False 로 설정하고, InfluenceRect 의 영역을 제한함으로써 제한된 도킹을 지원하게 할 수 있다.
그러면 이번에는 OnDockOver 이벤트에 대해서 알아보도록 하자. 선언부는 다음과 같다.

```
property OnDockOver: TDockOverEvent;
TDockOverEvent = procedure(Sender: TObject; Source: TDragDockObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean) of object;
```

OnDockOver 이벤트는 사용자가 자식 컨트롤을 도킹 site 컨트롤 위로 드래그할 때 도킹 site 컨트롤에서 발생한다. 이 이벤트는 기존의 drag-and-drop 에서 사용되었던 OnDragOver 이벤트의 역할과 유사하다. 즉, 도킹을 위해 자식 컨트롤이 release 될 때 Accept 파라미터를 설정하여 이를 받아들이거나 거부한다. 만약 도킹이 가능한 컨트롤이 OnGetSiteInfo 이벤트 핸들러에서 거부된 경우에는 OnDockOver 이벤트는 발생하지 않는다.
마지막으로 OnDockDrop 이벤트에 대해 알아보자. 선언부는 다음과 같다.

```
property OnDockDrop: TDockDropEvent;
```

TDockDropEvent = procedure(Sender: TObject; Source: TDragDockObject; X, Y: Integer) of object;

OnDockDrop 이벤트는 사용지기 자식 컨트롤을 도킹 site 에 올려 놓을 때 발생한다. 정상적인 drag-and-drop 작업의 OnDragDrop 이벤트와 유사한 역할을 하는데, 이 이벤트를 이용하여 실제로 자식 컨트롤이 도킹될 때 적절한 설정을 가능하게 할 수 있다. Source 파라미터가 가리키는 컨트롤이 도킹될 자식 컨트롤이므로, 이를 이용하여 여러가지 설정이 가능하다.

4. 자식 컨트롤의 도킹이 해제될 때의 조정 방법

도킹 site 는 자식 컨트롤을 드래그하면 DragMode 프로퍼티가 dmAutomatic 이라면 자동적으로 도킹이 해제된다. 도킹 site 는 자식 컨트롤이 도킹에서 벗어나고자 할 때 여기에 반응하여 여러가지 설정을 하거나, 아예 도킹이 해제되지 못하도록 할 수도 있는데, 이럴 때에는 OnUnDock 이벤트 핸들러를 사용한다.

property OnUnDock: TUnDockEvent;
TUnDockEvent = procedure(Sender: TObject; Client: TControl; var Allow: Boolean) of object;

Client 파라미터는 도킹을 해제하려는 자식 컨트롤을 나타내며, Allow 파라미터는 도킹 해제를 허용할 것인지 여부를 결정한다. OnUnDock 이벤트 핸들러를 구현할 때에는 현재 도킹되어 있는 다른 자식 컨트롤이 어떤 것이 있는지 아는 것이 유용하다. 이러한 정보는 읽기 전용인 DockClients 프로퍼티를 통해 얻을 수 있는데, 이 프로퍼티는 인덱스를 가진 TControl 의 배열이다. 도킹되어 있는 클라이언트 컨트롤의 수는 DockClientCount 프로퍼티를 통해 얻을 수 있다.

5. 자식 컨트롤이 드래그-도크 작업에 적절하게 반응하는 방법

도킹이 가능한 자식 컨트롤은 drag-and-dock 작업을 수행하는 동안 2 개의 이벤트를 발생시킨다. 각각의 이벤트의 선언부와 역할에 대해서 잠시 알아보도록 하자.
OnStartDock 이벤트의 선언부는 다음과 같다.

property OnStartDock: TStartDockEvent;
TStartDockEvent = procedure(Sender: TObject; var DragObject: TDragDockObject) of object;

OnStartDock 이벤트는 drag-and-drop 작업의 OnStartDrag 이벤트와 유사한 역할을 한다. OnStartDrag 이벤트와 마찬가지로, 도킹 가능한 자식 컨트롤이 사용자 정의 드래그 객체

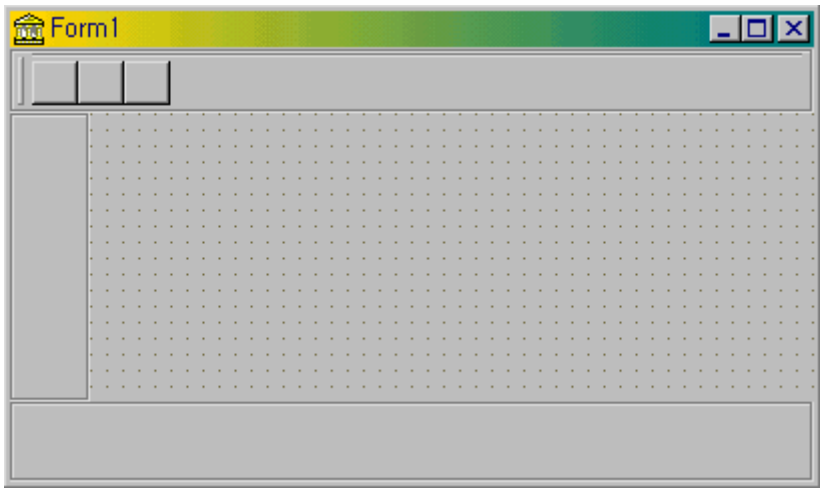
(drag object)를 생성하는 것을 허용할 것인지를 결정한다.
OnEndDock 이벤트의 선언부는 다음과 같다.

```
property OnEndDock: TEndDragEvent;
TEndDragEvent = procedure(Sender, Target: TObject; X, Y: Integer) of object;
```

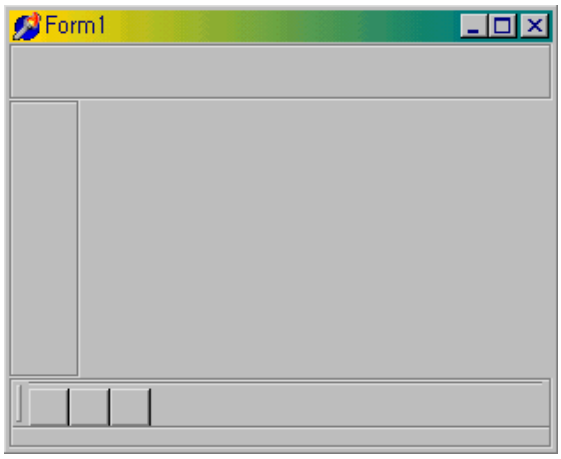
OnEndDock 이벤트는 드래그-드롭 작업의 OnEndDrag 이벤트와 비슷한 역할을 한다.

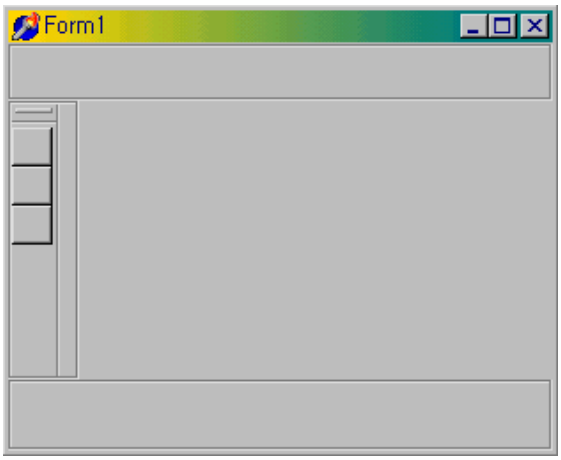
- 드래그-도크를 구현한 간단한 예제

그러면, 지금까지 설명한 내용을 바탕으로 간단하게 드래그-도크를 구현한 예제를 살펴 보자. 이를 위해서는 아무런 코딩 작업도 필요 없고 단지 컴포넌트를 올려 놓고, 프로퍼티를 변경하는 것으로 모든 것을 마칠 수 있다.
최근에 경향으로 보아 쿨바와 툴바를 이용한 인터페이스가 거의 대세를 장악하고 있는 듯하다. 그러므로, 여기에서도 쿨바와 툴바를 이용하도록 하겠다.
먼저 폼에 TCoolbar 컴포넌트 3 개를 올려 놓고, 각각의 Align 프로퍼티를 alTop, alBottom, alLeft 로 설정한다. 그리고, TToolbar 컴포넌트 하나를 alTop 으로 설정한 쿨바 컴포넌트 위에 다음과 같이 올려 놓도록 하자.

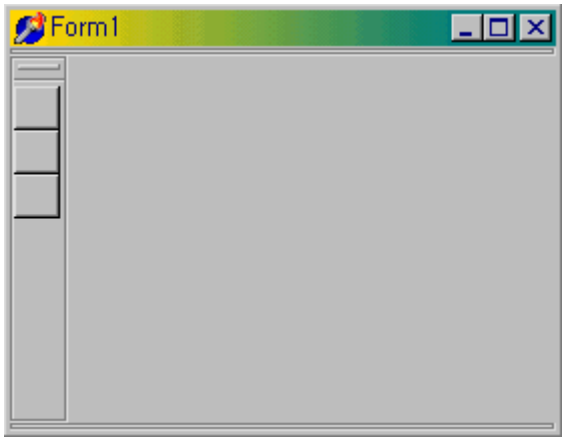


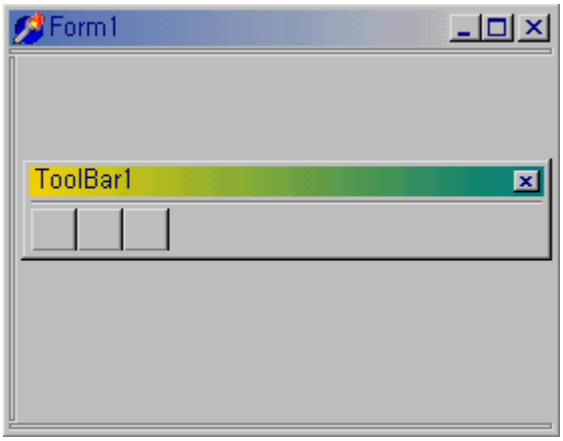
이제 이들의 프로퍼티를 각각 설정할 차례이다. 3 개의 쿨바 컴포넌트의 DockSite 프로퍼티를 모두 True 로 설정한다. 그리고, 툴바 컴포넌트 위에서 오른쪽 버튼을 클릭하고 New Button 메뉴로 툴버튼을 3 개 추가하면 앞의 그림과 같은 모양이 될 것이다.
이제 툴바 컴포넌트의 DragKind 프로퍼티를 dkDock, DragMode 프로퍼티를 dmAutomatic 으로 설정한 뒤에 컴파일하고 이를 실행하도록 하자.
그러면, 다음과 같이 툴바를 마음대로 옮겨다닐 수 있을 것이다.





그런데, 아마도 쿨바의 경계 부분이 눈에 거슬릴 것이다. 이를 제거하려면 다시 프로젝트 파일로 돌아와서 3 개의 쿨바 컴포넌트의 AutoSize 프로퍼티를 True 로 설정하도록 한다. 그리고, 다시 컴파일하고 실행을 하면 다음 그림과 같이 쉽게 드래그-도크가 구현된 윈도우를 볼 수 있을 것이다.





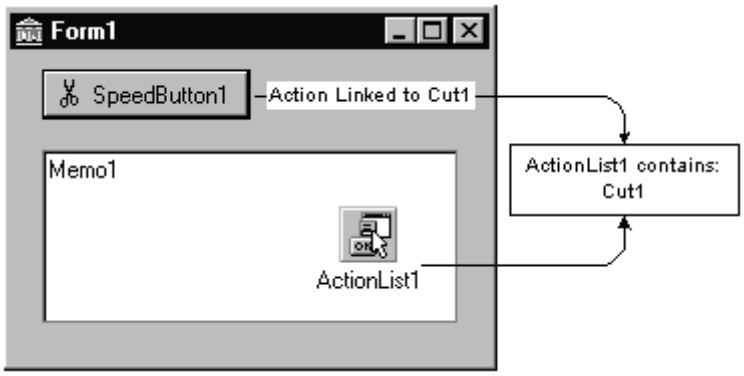
액션 리스트(Action List) 클래스의 활용

액션 리스트는 버튼과 메뉴 등의 사용자 명령어를 중앙 집중식으로 관리할 수 있도록 해준다. 액션이란 목적 객체(target object)에서 동작하는 사용자의 명령을 말한다. 액션은 액션 링크를 통해 클라이언트 컨트롤과 연결된다.

- VCL 액션 객체

액션은 클라이언트에 의해 유발되며, 보통 메뉴 아이템이나 버튼을 클릭했을 때에 시작된다. 여기에 관여하는 클래스는 기본적인 윈도우 에디트와 메뉴 명령을 구현한 TAction 클래스

에서 상속받은 클래스들로 StdActns 유닛에 포함되어 있다.
액션의 클라이언트에는 메뉴 아이템과 여러가지 버튼(TToolButton, TSpeedButton, TMenuItem, TButton, TCheckBox, TRadioButton 등)이 있으며, 액션은 클라이언트의 해당 명령에 의해 시작된다. 보통 클라이언트의 Click 과 액션의 Execute 가 연결된다.
TActionList 컴포넌트는 TAction 클래스의 리스트를 관리하는 컴포넌트이다. 또한, TActionLink 객체는 각각의 액션과 클라이언트의 접속을 유지하는 역할을 한다.
액션 목표(action target)는 보통 메모나 각종 데이터 컨트롤 등의 컨트롤이다. 예를 들어, DBActns 유닛에는 데이터 세트 컨트롤에 대한 액션을 구현한 클래스들을 포함하고 있다. 컴포넌트 제작자는 컨트롤을 제작하거나 사용할 때 필요한 액션을 직접 제작할 수 있으며, 이 유닛을 패키지에 포함하면 보다 모듈화된 어플리케이션을 제작할 수 있다.
다음의 그림은 이들 객체들의 관계를 나타낸 것이다.



여기에서 Cut1 은 액션이며, ActionList1 에는 Cut1 을 포함하고 있다. 그리고, 여기에서 SpeedButton1 이 Cut1 액션의 클라이언트가 되며, Memo1 이 목표(target)가 된다. 또한, 이 그림에서의 액션 링크는 SpeedButton1 과 Cut1 을 연관시키고 있는 것으로, ActionList 에 포함되어 있다.

- 액션의 사용

액션 리스트 컴포넌트는 컴포넌트 팔레트의 standard 페이지에 위치해 있으며, 폼이나 데이터 모듈에 추가해서 사용한다. 이 컴포넌트를 더블 클릭하면 액션 리스트 에디터가 나타나며, 여기에서 액션을 추가, 삭제, 재배열 등의 작업을 할 수 있게 된다.
오브젝트 인스펙터에서 각 액션의 프로퍼티를 설정할 수 있다. 간단하게 이해하자면 Name 프로퍼티는 액션을 가리키며 Caption, Checked, Enabled 등의 다른 프로퍼티와 이벤트는 클라이언트 컨트롤의 것이다.

1. 코드의 중앙집중화 :

델파이 4 에서는 TToolButton, TSpeedButton, TMenuItem, TButton 등의 컴포넌트의 프로퍼티에 Action 프로퍼티가 추가되었다. 이 프로퍼티를 액션 리스트에 열거된 하나의 Action 으로 설정하면 액션 객체와 연결이 된다. Name, Tag 프로퍼티를 제외한 모든 프로퍼티와 이벤트는 액션 객체와 연관되어 사용할 수 있게 되는데, 예를 들어 버튼이나 메뉴 아이템을 disable 시키는 코드를 중복하여 작성하지 않고, 이를 액션 객체에서 중앙집중식으로 관리할 수 있다. 즉, 특정 action 이 disabled 되면 해당되는 버튼이나 메뉴 아이템이 모두 disabled 된다.
또한, 버튼이나 메뉴 아이템을 클릭하면 컨트롤의 OnClick 이벤트는 자동으로 연결된 액션 아이템의 Execute 메소드를 호출하게 된다.

2. 프로퍼티의 링킹

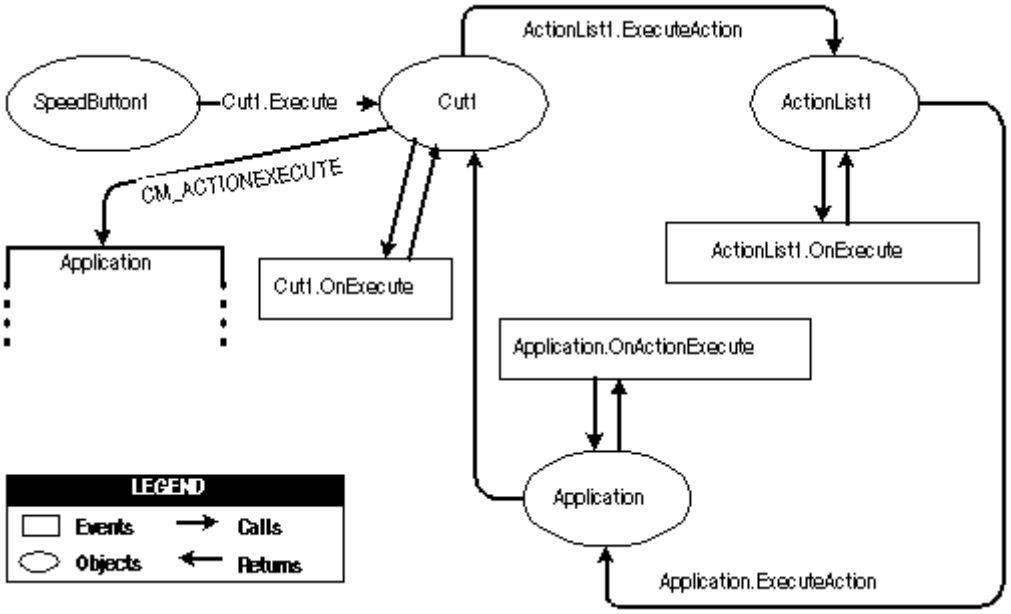
클라이언트의 액션 링크는 프로퍼티를 통해서 액션과 연결을 하게 된다. 이렇게 연결된 프로퍼티는 액션이 변경되면, 액션 링크는 클라이언트의 프로퍼티를 업데이트하게 된다. 자세한 내용은 각각의 액션 링크 클래스에 대한 도움말을 참고하기 바란다.

3. 액션의 실행

클라이언트 컴포넌트나 컨트롤이 클릭되면 연결된 액션의 OnExecute 이벤트가 발생한다. 예를 들어, 다음의 코드는 액션이 실행될 때 툴바의 visibility 를 토글한다.

```
procedure TForm1.Action1Execute(Sender: TObject);
begin
  ToolBar1.Visible := not ToolBar1.Visible;
end;
```

툴 버튼이나 메뉴 아이템을 사용하는 경우, 해당되는 메뉴 컴포넌트나 툴바의 Images 프로퍼티를 액션 리스트의 Images 프로퍼티로 설정해야 한다.
다음 그림은 Cut1 이라는 액션의 실행 사이클을 그려본 것이다. 이 그림에서 액션 클라이언트는 Speedbutton1 이며, Cut1 과 액션 링크를 통해 연결된다. Speedbutton1 의 Action 프로퍼티는 Cut1 으로 설정된다. 이렇게 되면, Speedbutton1 의 Click 메소드는 Cut1 의 Execute 메소드를 실행하게 된다.



Speedbutton1 을 클릭하면 다음과 같은 실행 사이클을 시작하게 된다.

- Speedbutton1 의 Click 메소드는 Cut1.Execute 를 실행한다.
- Cut1 액션은 액션 리스트(ActionList1)에 Execute 메소드의 처리를 맡기게 된다. 액션 리스트는 ExecuteAction 메소드를 호출하게 된다.
- ActionList1 은 ExecuteAction 메소드에 대한 OnExcute 이벤트 핸들러를 호출하게 된다. 이때 액션 리스트의 ExecuteAction 메소드는 액션 리스트에 담겨 있는 모든 액션에 적용된다. 이 이벤트 핸들러에는 Handled 라는 파라미터를 가지고 있으며, 디폴트로 False 를 반환한다. 이벤트 핸들러가 사용되고, 이벤트를 처리한 경우에는 다음과 같이 이 파라미터에 True 를 반환해야 한다.

```
procedure TForm1.ActionList1ExecuteAction(Action: TBasicAction; var Handled: Boolean);
begin
  {ActionList1 에 담긴 다른 액션의 실행을 막는다.}
  Handled := True;
end;
```

만약 이 시점에서 처리가 이루어지지 않으면, 액션 리스트 이벤트 핸들러는 계속해서 처리를 하게 된다.

- Cut1 액션은 어플리케이션 객체의 ExecuteAction 메소드를 실행하게 되고, 이로 인해 OnActionExecute 이벤트 핸들러가 실행된다. 어플리케이션 객체의 ExecuteAction 메

소드는 어플리케이션에 있는 모든 액션에 적용된다. 실행되는 순서는 액션 리스트의 ExecuteAction 메소드와 동일하다. 이 이벤트 핸들러의 Handled 파라미터 역시 False 를 디폴트로 가지며, 처리 순서는 이 시점에서 끝난다. 마찬가지로 다음과 같이 True 를 반환해야 한다.

```
procedure TForm1.ApplicationExecuteAction(Action: TBasicAction; var Handled: Boolean);
begin
    {어플리케이션에 있는 다른 액션의 실행을 막는다.}
  Handled := True;
end;
```

만약 어플리케이션 이벤트 핸들러에서 처리되지 않으면, Cut1 은 CM_ACTIONEXECUTE 메시지를 어플리케이션의 WndProc 에 전송한다. 이렇게 되면, 어플리케이션은 액션을 실행하기 위해 목표(target) 객체를 찾아서 실행하게 된다.

4. 액션의 업데이트:

어플리케이션이 idle 상태에 있을 경우 액션이 일어날 때마다 OnUpdate 이벤트가 발생한다. 이를 이용하면 어플리케이션에서 중앙집중화된 코드를 enable, disable 또는 check, uncheck 하는 작업을 할 수 있다. 다음의 코드를 살펴보자

```
procedure TForm1.Action1Update(Sender: TObject);
begin
  {툴바가 현재 보이는지 여부를 나타낸다.}
  (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

참고로 시간이 많이 걸리는 코드는 OnUpdate 이벤트 핸들러에 추가하지 않는 것이 좋다. 자주 실행되기 때문에, 어플리케이션 전체 성능에 심각한 저하를 가져오게 된다.

- 액션을 이용한 컴포넌트 제작:

델파이 4 에서는 메뉴나 버튼 컨트롤에서 프로퍼티 값을 설정하여 액션 객체에 의해 상속된 프로퍼티를 선별적으로 오버라이드할 수 있다. 이렇게 하면 액션에서의 프로퍼티는 바뀌지 않고, 버튼과 메뉴 컨트롤에만 영향을 미치게 할 수 있다.

컴포넌트 제작자는 어떤 type 의 컨트롤에든 액션 기능을 추가할 수 있게 되었다. TControl 의 protected 섹션에 선언된 Action 프로퍼티를 public 또는 published 로 섹션으로 이동하면 된다. 액션을 이용한 컴포넌트 제작에 대한 내용은 4 부에서 자세히 다루게 될 것이다.

● 액션의 등록

델파이 4 에서는 컴포넌트를 이용하지 않고, 전역 루틴을 이용해서 액션을 등록하거나 해제하는 것도 가능하다. 이때 사용되는 루틴은 다음과 같다.

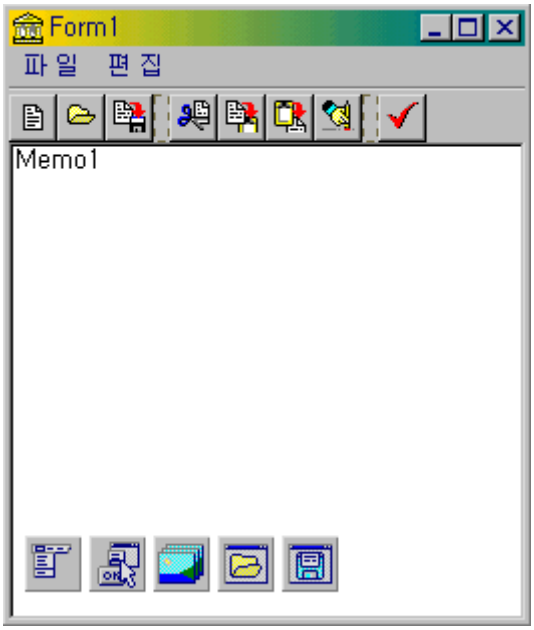
```
procedure RegisterActions(const CategoryName: string;
  const AClasses: array of TBasicActionClass);
procedure UnRegisterActions(const AClasses: array of TBasicActionClass);
```

이들 프로시저는 ActnList.pas 유닛에 선언되어 있다. 이들을 이용하여 표준 액션을 등록하는 예제 코드는 다음과 같다.

```
{ Standard action registration }
RegisterActions('', [TAction]);
RegisterActions('Edit', [TEditCut, TEditCopy, TEditPaste]);
RegisterActions('Window', [TWindowClose, TWindowCascade, TWindowTileHorizontal,
  TWindowTileVertical, TWindowMinimizeAll, TWindowArrange]);
```

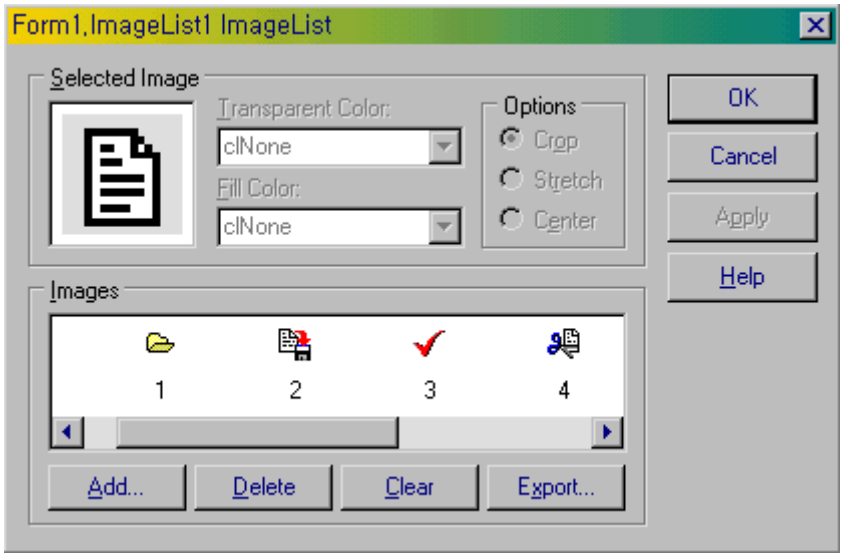
● 간단한 예제 어플리케이션의 제작

액션 리스트에 대한 예제로 아주 간단한 에디터를 하나 만들어 보자. 폼위에 TToolbar 컴포넌트를 하나 얹고, 툴바에서 오른쪽 버튼을 클릭하여 툴바 버튼을 8 개, 분리자(Separator)를 2 개 추가한다. 그리고, 메모 컴포넌트를 하나 추가하고 Align 프로퍼티를 alClient 로 설정한다. 여기에 다음 그림과 같이 파일을 읽고, 쓸 수 있도록 TOpenDialog, TSaveDialog 컴포넌트를 하나씩 얹고, 메인 메뉴를 만들기 위한 TMainMenu 컴포넌트와 TActionList, TImageList 컴포넌트를 하나씩 추가하자.

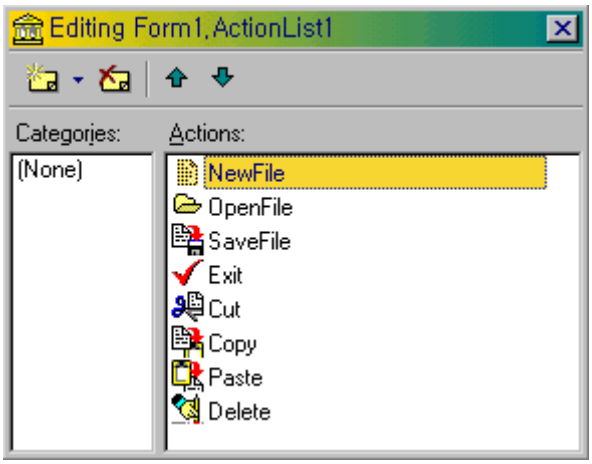


물론, 독자 들의 폼의 모양은 이와는 다를 것이다. 이것은, 아직 비트맵도 설정하지 않았고, 메뉴 아이템도 결정하지 않았기 때문이므로 계속 따라가다 보면 결국에는 이런 모습이 될 것이다.

그러면, 먼저 비트맵 이미지를 결정하도록 한다. 델파이에서 제공하는 비트맵이나 아이콘은 보통 Common Files₩Borland Shared₩Images 디렉토리에서 찾아볼 수 있다. 여기서 Buttons 디렉토리에서 앞의 그림의 8 개 비트맵을 찾아서 ImageList 에 추가하자. 이때 주의할 것은 버튼의 비트맵은 보통 2 개의 이미지로 들어가게 된다는 것이다. 뒤쪽의 이미지는 마스크로 쓰이는 것으로, 보통 버튼이 클릭되는 효과를 보이게 하기 위해 사용된다. 여기서는 마스크를 사용하지 않을 것이므로, 버튼의 비트맵을 추가한 후 마스크 비트맵은 삭제하도록 하자. 8 개의 비트맵을 모두 추가하고 나면 ImageList 의 형태가 다음과 같으면 된다.



이제는 액션 리스트를 정의할 차례이다. 먼저, ActionList1 의 Images 프로퍼티를 ImageList1 으로 설정하여 앞에서 정의한 비트맵을 사용하도록 한다. 그리고, ActionList1 컴포넌트를 더블 클릭하면 액션 에디터가 뜨는데, 여기에 다음과 같이 액션을 8 개 정의한다.



여기서 오른쪽 버튼을 클릭하면, 이와 같이 새로운 액션을 추가할 수 있게 된다. 그리고, 각각의 액션을 클릭하고 오브젝트 인스펙터에서 프로퍼티를 설정할 수 있는데, 다음과 같이 설정하면 앞의 그림과 같이 된다.

Name	Caption	ImageIndex	Name	Caption	ImageIndex
NewFile	새 파일	0	Cut	오려두기	4
OpenFile	파일열기	1	Copy	복사하기	5
SaveFile	파일저장	2	Paste	붙여넣기	6

Exit	종 료	3	Delete	삭제하기	7

이제 액션을 모두 정의했으므로, 메뉴와 툴바를 액션에 맞춰넣기만 하면 된다.
먼저 MainMenu1 컴포넌트의 Images 프로퍼티를 ImageList1 으로 설정한다. 그리고, MainMenu1 컴포넌트를 더블 클릭하여 메뉴를 편집한다. 여기서 특별히 할 것이 없다. 개발자가 할 일은 먼저 '파 일'과 '편 집'이라는 가장 큰 메뉴를 하나씩만 만들고 그 다음 메뉴 아이템 부터는 Action 프로퍼티를 ActionList1 에 정의한 NewFile 에서 부터 Delete 까지의 8 개의 액션을 하나씩 설정하기만 하면 된다. 여기서 '파 일' 섹션에 NewFile, OpenFile, SaveFile, Exit 를 Action 프로퍼티로 설정한다. SaveFile 과 Exit 사이에는 메뉴 아이템의 Caption 을 '-'로 설정하여 구분선을 하나 삽입한다. 그리고, '편 집' 섹션에 Cut, Copy, Paste, Delete 를 Action 프로퍼티로 설정한다. 이렇게만 하면 Caption 과 메뉴 아이템의 비트맵은 다음 그림과 같이 자동으로 설정되었을 것이다.

툴바 역시 마찬가지이다. Toolbar1 의 Images 프로퍼티를 ImageList1 으로 설정하고, 각각의 툴바 버튼의 Action 프로퍼티를 NewFile, OpenFile, SaveFile, Exit, Cut, Copy, Paste, Delete 로 각각 설정하면 그걸로 끝이다. 여기까지 그대로 설정했다면 앞에서 보았던 폼의 모양과 동일하게 변했을 것이다.
이제 프로퍼티의 설정이 모두 끝났다. 그러면, 실제로 돌아갈 수 있도록 이벤트 핸들러 들을 적당하게 작성해 보자. 먼저, 폼이 시작할 때 Memo1 의 내용을 깨끗이 지워주기 위해서 Form1 의 OnCreate 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;
end;
```

이제는 8 개의 액션에 대한 이벤트 핸들러를 작성해주면 된다. 오브젝트 인스펙터에서 NewFile 을 선택하고, 이벤트 페이지 탭에서 OnExecute 이벤트의 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.NewFileExecute(Sender: TObject);
begin
  Memo1.Lines.Clear;
end;
```

그리고, 파일을 읽고 쓰는 OpenFile, SaveFile 액션의 OnExecute 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.OpenFileExecute(Sender: TObject);
begin
  OpenDialog1.Filter := 'Text files (*.txt)|*.TXT';
  if OpenDialog1.Execute then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
  end;
end;

procedure TForm1.SaveFileExecute(Sender: TObject);
begin
  SaveDialog1.Filter := 'Text files (*.txt)|*.TXT';
  if SaveDialog1.Execute then
  begin
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  end;
end;
```

파일을 읽고 쓰기 위해서 OpenDialog1, SaveDialog1 대화상자를 이용하며, Filter 프로퍼티를 기본적으로 텍스트 파일로 설정하였다.
Exit 액션에 대한 이벤트 핸들러는 간단히 다음과 같이 작성하면 된다.

```
procedure TForm1.ExitExecute(Sender: TObject);
begin
  Close;
end;
```

클립보드 작업을 위한 Cut, Copy, Paste 액션과 선택된 문자열을 삭제하기 위한 Delete 액

션의 OnExecute 이벤트 핸들러는 다음과 같이 작성한다.

```
procedure TForm1.CutExecute(Sender: TObject);
begin
  Memo1.CutToClipboard;
end;

procedure TForm1.CopyExecute(Sender: TObject);
begin
  Memo1.CopyToClipboard;
end;

procedure TForm1.PasteExecute(Sender: TObject);
begin
  Memo1.PasteFromClipboard;
end;

procedure TForm1.DeleteExecute(Sender: TObject);
begin
  Memo1.ClearSelection;
end;
```

다들 간단한 코드이므로 자세한 설명은 생략하겠다. 이제 모든 이벤트 핸들러의 작성이 끝난 것이다. 컴파일을 하고, 프로그램을 실행하면 간단한 에디터로 사용할 수 있을 것이다. 이렇게 액션 리스트를 사용하면, 과거에 스피드 버튼과 메뉴를 통해 중복해서 작성하던 여러 코드를 간단하게 중앙 집중적으로 관리할 수 있다. 그리고, 어플리케이션의 실행 블록을 체계적으로 나눌 수 있기 때문에 메뉴 아이템과 버튼이 많은 어플리케이션의 경우 코드의 직관성과 재사용성도 월등히 향상시킬 수 있다.
아무리 좋은 연장도 제대로 쓰지 않으면 아무 소용이 없는 법이므로, 어플리케이션의 크기가 조금 복잡하다 싶으면 꼭 액션 리스트를 쓸 것을 권하는 바이다. 아마도 훨씬 관리하기도 편해지고, 디버깅도 쉬워진 것을 느낄 수 있을 것이다.

정 리 (Summary)

이번 장에서는 델파이의 VCL 계층 구조와 핵심적인 클래스에 대해서 소개하고, 델파이가 제공하는 컴포넌트 들에 대해 간략하게나마 어떤 것들이 있는지 알아보았다.

그리고, 비교적 많이 쓰일 수 있는 공통 주제로 드래그-드롭, 드래그-도크, 액션 리스트의 활용 방법을 자세한 설명과 예제를 활용해서 설명하였다.
여기서 설명한 내용들은 매우 자주 쓰이면서도 중요한 내용이므로, 여러 차례 연습을 해서 반드시 몸에 체득하기 바란다.
다음 장에서는 어플리케이션에 파일 입출력과 출력 기능을 추가하는 방법에 대해서 알아보도록 한다.