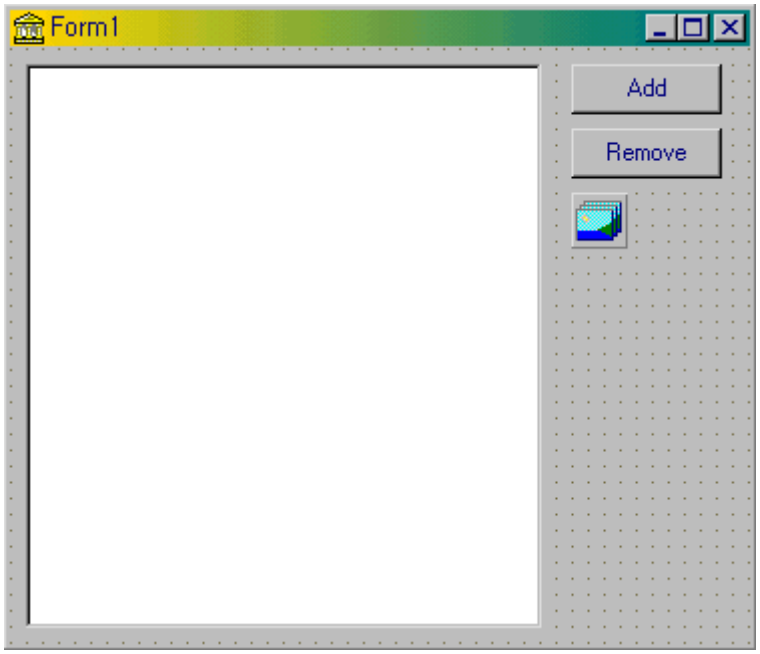
Win32 공통 컨트롤 정복

(Mastering Win32 Common Controls)

Win32 의 공통 컨트롤(common control) 컴포넌트 들은 유용하게 생각되면서도 사용하기가 비교적 까다로운 컴포넌트가 많다. 또한, 현재까지 나온 델파이 서적들에서도 기본적인 컴포넌트의 사용 방법에 대해서는 비교적 자세한 사용법을 기술하고 있지만 Win32 공통 컨트롤에 대한 설명은 비교적 부족한 것이 사실이다.
Win32 공통 컨트롤에는 트랙바(Track Bar)와 업다운(UpDown) 컨트롤, 핫키(HotKey) 컨트롤이나 진행바(Progress Bar)와 같은 단순한 컨트롤과 다소 복잡한 상태바(Statud Bar), 헤더 컨트롤, 이미지 리스트, 탭 컨트롤(Tab Control), 페이지 컨트롤(Page Control), 리스트뷰와 트리뷰 등이 있다. 그 밖에도 델파이 3 에서는 툴바(Tool bar)와 쿨바(Cool bar), 애니메이션 (Animate), 달력 컴포넌트(DateTimePicker) 등이 추가되었고 델파이 4 에서는 페이지 스크롤러(Page Scroller)와 컨트롤 바(Control Bar)가 추가 되었다.
지면 관계상 이들 모두를 다룰 수는 없고, 이번 장에서는 비교적 새로운 형태의 객체 접근 방법에 대한 사용자 인터페이스를 제공하면서, 사용법이 비교적 까다로운 트리뷰(TreeView)와 컴포넌트의 활용 방법을 소개하고, 리스트뷰(ListView) 컴포넌트와 이미지 리스트, 상태바 컨트롤을 같이 이용하여 탐색기와 유사한 어플리케이션을 작성해 볼 것이다. 그리고 명령을 수행하도록 하는 사용자 인터페이스를 제공하는 툴바(Toolbar)와 쿨바(Coolbar)의 사용방법에 대해서 간단하게 알아볼 것이다.

트리뷰 컴포넌트 시작하기

트리뷰 컴포넌트를 다루는 기본적인 방법에 대해서 알아보도록 하자. 새로운 어플리케이션을 시작하고 폼에 TTreeView 컴포넌트 하나와 버튼을 두개 올려 놓자. 버튼의 캡션은 각각 ‘Add’, ‘Remove’로 설정한다. 그리고 TImageList 컴포넌트를 다음과 같이 추가한다.



트리뷰 컴포넌트는 실제로 그 구조에 있어서 노드라는 객체를 사용하게 된다. 노드는 트리뷰 컴포넌트를 이루는 가장 기본적인 단위가 되며, 눈에 보이는 각각의 아이템 들이다. 이러한 노드는 TTreeNode 클래스로 선언되어 있다. 그럼 트리뷰 컴포넌트에 새로운 노드를 추가하는 방법을 알아보자.

노드를 추가할 때에는 어느 노드에 노드를 추가할 지를 지정해 주어야 한다. 이러한 노드의 부모는 루트 노드일 수도 있고, 다른 자식 노드일 수도 있다. 루트 노드의 경우에는 부모가 없다. 그러므로, 루트 노드의 경우 TTreeNode.Parent 의 값은 nil 이다. 루트 노드를 제외한 노드는 루트 노드를 포함한 다른 노드를 부모로 가지게 된다.

노드를 추가할 때에는 보통 현재 선택된 노드가 있으면 그 노드의 형제 노드로 추가되거나, 자식 노드로 추가 하는 등의 결정을 하게 된다. 이처럼 현재 선택된 노드를 알아볼 때 사용하는 것인 Selected 이다. 보통 이 프로퍼티를 사용하면 현재 선택된 노드를 TTreeNode 의 형태로 알아볼 수 있다. 만약 이 프로퍼티가 nil 이라면 현재 트리뷰가 비어 있거나, 아무 노드도 선택되지 않을 것이다. 트리뷰가 비어 있는지는 아이템의 컬렉션인 Items.Count 프로퍼티로 알아볼 수 있다. 이 값이 0 이면 트리뷰에 노드가 하나도 없는 경우이다.

노드를 추가하는 메소드에 대해서 알아보자. 트리뷰에서 사용하는 메소드로는 다음과 같은 것들이 있다.

```
function Add(Node: TTreeNode; const S: string): TTreeNode;
function AddFirst(Node: TTreeNode; const S: string): TTreeNode;
function Insert(Node: TTreeNode; const S: string): TTreeNode;
function AddChild(Node: TTreeNode; const S: string): TTreeNode;
```

```
function AddChildFirst(Node: TTreeNode; const S: string): TTreeNode;
function AddObject(Node: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
function InsertObject(Node: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
function AddObjectFirst(Node: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
function AddChildObject(Node: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
function AddChildObjectFirst(Node: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

상당히 많아 보이지만, 크게 2 가지 카테고리로 나누어 볼 수 있다.
우선 파라미터를 보면 메소드 이름에 Object 가 들어가 있는 것들에는 Ptr 이라는 포인터 파라미터가 하나씩 더 있는 것을 알 수 있다. Object 가 들어가 있지 않은 메소드 들은 단순히 기준이 되는 노드(Node 파라미터)와 트리뷰에 추가할 문자열(S 파라미터)만 지정해 주면 트리뷰에 노드가 추가된다. 그에 비해 Object 가 들어가 있는 메소드 들은 트리뷰에 노드를 추가하고, 각 노드에 객체를 지정할 수 있다. 이때 각 객체에 대한 포인터를 노드에 지정하는 파라미터가 S 파라미터이다.
그리고, First 가 들어가 있는 메소드 들은 노드의 위치를 지정하는 것으로 추가되는 노드가 동격의 노드들 중에서는 첫번째 위치로 추가된다는 의미이다. Add, Insert, AddChild 의 차이는 Add 는 형제 노드 중에서 제일 마지막 위치에, Insert 는 형제 노드 중 현재 위치를 대신 차지하게 하는 것이며, AddChild 는 선택된 노드의 자식으로 제일 마지막 위치에 추가되게 하는 메소드이다.
그러면 이제 연습을 해 보자. Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

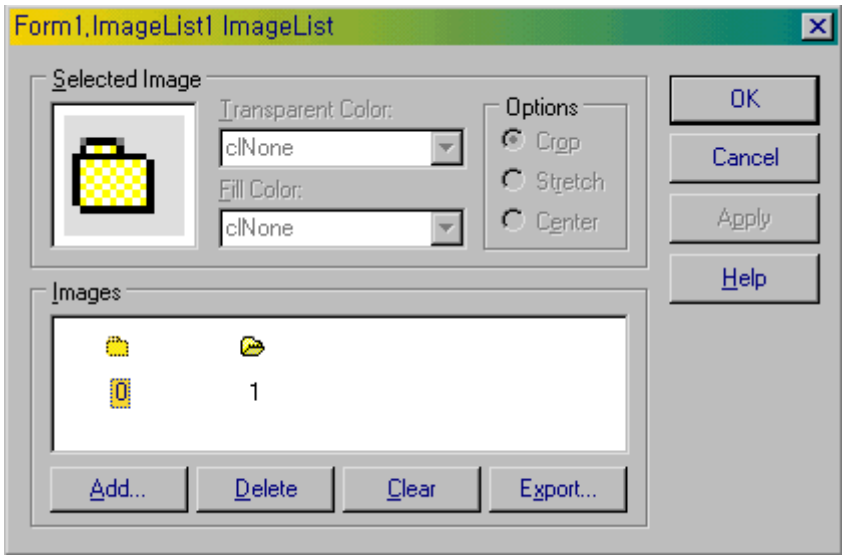
```
procedure TForm1.Button1Click(Sender: TObject);
var
  Text: string;
begin
  if TreeView1.Selected = nil then
  begin
    if TreeView1.Items.Count = 0 then
    begin
      with TreeView1.Items.AddFirst(nil, 'Root') do
      begin
        Selected := True;
      end;
    end
    else
    begin
```

```
      ShowMessage('부모 노드를 선택하세요 !');
      Exit;
    end;
  end
  else
  begin
    InputQuery('새 노드', '이름 ?', Text);
    TreeView1.Items.AddChild(TreeView1.Selected, Text);
  end;
end;
```

먼저 선택된 노드가 없는지 검사한다. 이때 Selected 프로퍼티의 값이 nil 이라면 트리뷰가 현재 비어 있는 경우가 있을 것이고, 노드가 선택되지 않은 경우가 있을 것이다. 현재 비어 있다면 AddFirst 메소드에 첫번째 파라미터로 nil 을 대입해서 루트 노드를 생성한다. 그리고, TTreeNode 의 Selected 프로퍼티를 True 로 설정해서 이 노드가 선택되도록 한다. 만약 노드가 선택되지 않은 경우라면 노드를 선택하라는 메시지를 화면에 띄우도록 한다. Selected 프로퍼티의 값이 nil 이 아니면 추가할 노드의 이름을 InputQuery 함수를 이용해서 입력받고, Selected 노드의 자식 노드로 새로운 노드를 추가한다.
사용법이 그다지 어렵지 않다는 것을 알 수 있을 것이다. 그러면 노드를 제거하는 부분을 만들어 보자. Button2 의 OnClick 이벤트 핸들러를 다음과 같이 작성하자.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if TreeView1.Selected = nil then
  begin
    ShowMessage('선택된 것이 없습니다.');
    Exit;
  end;
  if TreeView1.Selected.Level = 0 then
  begin
    ShowMessage('Root 는 제거할 수 없습니다');
    Exit;
  end;
  TreeView1.Selected.Delete;
end;
```

선택된 것이 없거나 선택된 노드가 루트일 경우에는 제거하지 않고, 선택된 노드가 있으면 Delete 메소드를 사용해서 노드를 제거한다.
Root 인지 알아보는 프로퍼티로 Level 프로퍼티를 사용했는데, 이 값이 0 이면 루트이며, 자식 노드가 하나 늘어날 때 마다 Level 프로퍼티가 하나씩 증가한다.
이제 처음에 추가한 TImageList 컴포넌트를 이용해서 노드를 상징하는 이미지를 나타내도록 해보자. 흔히 사용하는 폴더 이미지를 이용해서, 하부 노드가 보이는 경우에는 열린 폴더의 그림이 하부 노드가 안 보이도록 된 경우에는 닫힌 폴더가 나타내도록 하자. 이때 다음과 같이 닫힌 폴더 그림의 인덱스를 0, 열린 폴더를 1 로 설정한다. .



그리고, 트리뷰 컴포넌트가 ImageList1 을 사용하게 하기 위해 오브젝트 인스펙터에서 TreeView1 의 Images 프로퍼티를 ImageList1 으로 설정한다. 이런 작업이 끝났으면 TImageList 컴포넌트를 이용해서 노드가 선택되었을 때의 이미지를 설정하기 위해서 아래와 같이 Button1 의 OnClick 이벤트 핸들러를 수정한다.

```
procedure TForm1.Button1Click(Sender: TObject);

...(중략)

    if TreeView1.Items.Count = 0 then
    begin
      with TreeView1.Items.AddFirst(nil, 'Root') do
      begin
        Selected := True;
        ImageIndex := 0;
```

```
        SelectedIndex := 1;
      end;

... (중략)

    InputQuery('새 노드', '이름 ?', Text);
    with TreeView1.Items.AddChild(TreeView1.Selected, Text) do
    begin
      ImageIndex := 0;
      SelectedIndex := 1;
    end;
```

즉, 노드를 추가할 때 마다 그 노드의 기본적인 ImageIndex 와 SelectedIndex 프로퍼티에 대한 정보를 주는 것이다. ImageIndex 프로퍼티는 노드가 선택되지 않았을 때의 TImageList 컴포넌트에서의 이미지의 인덱스를 나타낸다. 여기서는 닫힌 폴더 그림을 나타내므로 0 을 대입한다. SelectedIndex 프로퍼티는 노드가 선택되었을 때 보여줄 이미지의 인덱스이다. 이 값이 같으면 노드가 선택되었을 때와 선택되지 않았을 때 보여지는 이미지가 동일하다.

트리노드(TreeNode) 정복

트리뷰 컴포넌트는 실제로 눈에 보이는 부분이다. 그렇지만, 그 내부의 동작은 기본적으로 각각의 트리 노드에 대해서 잘 알아야 트리뷰를 정복했다고 말할 수가 있는 것이다. 이번 섹션에서는 트리 노드를 다루는 법에 대해서 알아보도록 하자.

- 이름의 충돌문제 해결

트리의 경우에 흔히 같은 부모 노드를 가진 노드 들의 경우에 노드의 이름이 동일하면 안되는 경우가 있다. 예를 들어, 파일 시스템을 예로 들면 하나의 디렉토리에 들어있는 파일의 이름이 동일한 것이 둘 이상 존재해서는 안된다. 그렇지만, 트리뷰 자체에는 이를 검사하는 방법이 없기 때문에 이런 역할을 하는 루틴을 만들어 주어야 한다. 다음에 소개하는 함수가 이러한 이름 충돌 문제를 검사해 주는 함수이다.
소스 코드를 살펴 보자.

```
function IsDuplicate(ANode: TTreeNode; NewName: string; Inclusive: Boolean): Boolean;
var
```

```
  TempNode: TTreeNode;
begin
  if ANode = nil then
  begin
    Result := False;
    Exit;
  end;
  if Inclusive then
    if CompareText(ANode.Text, NewName) = 0 then
    begin
      Result := True;
      Exit;
    end;
  TempNode := ANode;
  repeat
    TempNode := TempNode.GetPrevSibling;
    if TempNode <> nil then
      if CompareText(TempNode.Text, NewName) = 0 then
      begin
        Result := True;
        Exit;
      end;
  until TempNode = nil;
  TempNode := ANode;
  repeat
    TempNode := TempNode.GetNextSibling;
    if TempNode <> nil then
      if CompareText(TempNode.Text, NewName) = 0 then
      begin
        Result := True;
        Exit;
      end;
  until TempNode = nil;
  Result := False;
end;
```

이 함수의 개요를 살펴 보면 파라미터로 트리 노드(ANode)와 비교할 이름(NewName)과 넘겨진 트리 노드도 검사할 것인지 여부(Inclusive)를 설정해주면, 만약 Inclusive 가 True 라면 현재의 노드의 이름과 새로운 이름이 같은지도 검사하며, 넘겨진 ANode 의 앞쪽에 있는 모든 형제 노드와 뒷쪽에 있는 모든 형제 노드의 이름을 비교해서 NewName 과 같은 것이 있으면 결과를 True 로 반환하는 함수이다.
즉, 트리 노드를 추가할 때 추가될 노드의 이름을 입력받은 후 추가될 위치에 있는 모든 형제 노드들의 이름에 동일한 것이 있는지 먼저 검사할 때 유용하게 쓰일 수 있는 함수이다.
함수의 개요를 읽어 보면 대부분의 소스 코드를 이해할 수 있을 것으로 믿는다.
그런데, 여기서 한가지 소개할 것은 근처의 노드를 얻어오는 메소드 들이다. 앞의 함수에서는 GetPrevSibling, GetNextSibling 이 사용되었다. 이들은 모두 TTreeNode 의 메소드로 각각 앞쪽의 형제 노드와 뒷쪽의 형제 노드를 TTreeNode 형으로 반환해 주는 메소드이다. 이들과 비슷한 역할을 하는 메소드로 다음과 같은 것들이 있다.

```
function GetFirstChild: TTreeNode;
function GetLastChild: TTreeNode;
function GetNext: TTreeNode;
function GetNextSibling: TTreeNode;
function GetNextChild(Value: TTreeNode): TTreeNode;
function GetNextVisible: TTreeNode;
function GetPrev: TTreeNode;
function GetPrevSibling: TTreeNode;
function GetPrevChild(Value: TTreeNode): TTreeNode;
function GetPrevVisible: TTreeNode;
```

Next 와 Prev 가 들어간 것의 차이는 쉽게 이해 할 수 있을 것으로 생각되며, Child 가 들어간 것과 들어가지 않은 것도 쉽게 알 수 있을 것이다. GetNext 와 GetPrev 는 Visible 프로퍼티와 자손 여부에 관계 없이 다음이나 이전 노드를 얻어오는 메소드이다. 이들 각각에 대한 자세한 설명은 도움말을 참고하기 바란다.
그러면, 실제로 IsDuplicate 함수를 사용해서 노드를 추가할 때 겹치는 이름이 발생하지 않도록 Button1 의 OnClick 이벤트 핸들러를 수정하도록 하자.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Text: string;
begin
```

... (중략)

```
InputQuery('새 노드', '이름 ?', Text);
if IsDuplicate(TreeView1.Selected.GetFirstChild, Text, True) then
begin
  ShowMessage('같은 이름의 노드가 이미 존재합니다 !');
  Exit;
end;
```

... (후략)

이렇게 수정하면 InputQuery 를 이용해서 새로운 노드의 이름을 얻어온 후에 이 이름과 현재 선택된 노드의 첫번째 자식 노드를 IsDuplicate 함수의 파라미터로 넘겨주면 현재 선택된 노드의 모든 자식 노드에 대해 중복되는 이름이 있는지 검사하게 된다. 이 값이 True 라면 같은 이름의 노드가 존재하는 것이므로 새로운 노드를 추가하지 않고 끝내게 된다.
이것으로 첫번째 예제가 완성되었다. 실행하고, Add 와 Remove 버튼을 눌러서 아이템으 추가하고 삭제해보기 바란다.

- 트리 노드와 각종 데이터의 연결

각 트리 노드에는 Data 라는 프로퍼티가 있다. 이 프로퍼티를 이용하면 트리 노드와 여러 가지 데이터를 연결해서 사용할 수가 있다. TTreeNode 의 Data 프로퍼티는 기본적으로 untyped 포인터 데이터 형이기 때문에 어떤 객체도 지정할 수 있다. Data 프로퍼티를 사용하는 방법을 알아보자.
클래스를 사용해서 객체를 연결하는 방법은 델파이가 클래스 인스턴스를 실제로 포인터로 간주하는 객체 참조 모델을 사용하기 때문에 간단하면서도 매우 편리하게 사용할 수 있다. 객체 참조 모델에 대해서는 오브젝트 파스칼과 C++, 자바에 대해서 비교하고 있는 6 장의 내용을 참고하기 바란다. 다음의 코드를 살펴보자

```
var
  SampleClass: TSampleClass
begin
  SampleClass: TSampleClass.Create;
  SampleClass.Free;
end;
```

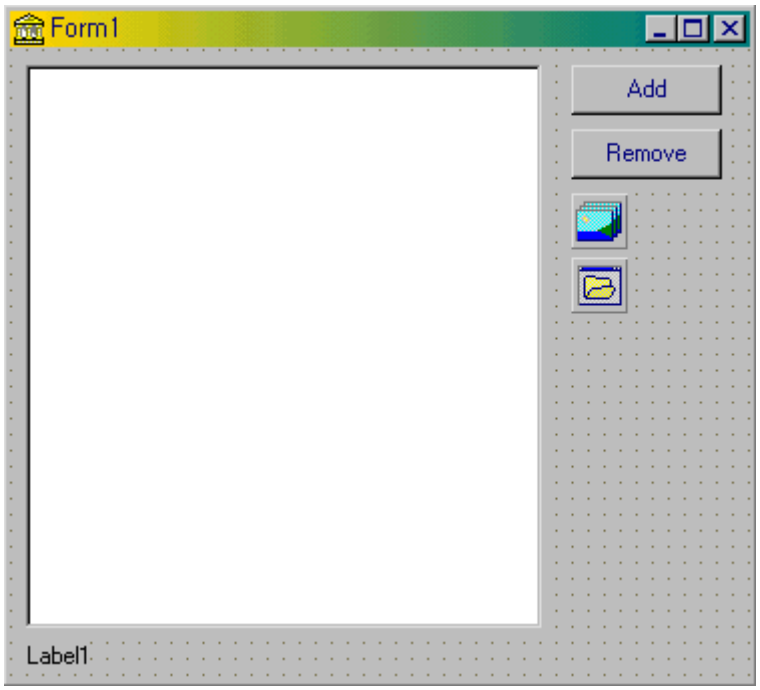
앞에서 SampleClass 변수는 실제로는 포인터이다. 실제로 Data 프로퍼티에 사용하는 방법은 다음과 같다.

```
Node.Data := TSampleClass.Create;
TSampleClass(Node.Data).Free;
```

첫째 줄에서 TSampleClass 인스턴스를 생성하고 이를 TTreeNode.Data 프로퍼티에 대입한다. 두번째 줄에서 포인터를 형변환(type cast)하고 이를 메모리에서 해제한다. 이렇게 형변환이 필요한 이유는 Data 프로퍼티가 untyped 포인터이기 때문이다.
그러면 이를 이용한 예제를 하나 만들어 보자. 앞에서 만든 첫번째 예제를 Open 하고 이를 확장하겠다. Add 버튼을 누르면, 파일 이름을 선택할 수 있는 대화상자가 나타나고 여기에서 선택한 파일의 이름을 Data 프로퍼티에 저장하고 이를 보여주는 예제이다. 먼저 다음 그림과 같이 TLabel, TOpenDialog 컴포넌트를 올려 놓자.
그리고, type 선언문에 사용할 데이터 클래스를 다음과 같이 선언해서 추가한다.

```
type

  TNodeData = class
    Text: string;
  end;
```



이제 Button1 의 OnClick 이벤트 핸들러를 다음과 같이 수정한다.

```
procedure TForm1.Button1Click(Sender: TObject);

... (중략)

        with TreeView1.Items.AddFirst(nil, 'Root') do
        begin
          Selected := True;
          Data := TNodeData.Create;
          TNodeData(Data).Text := '루트 디렉토리입니다 !';

... (중략)

      InputQuery('새 노드', '이름 ?', Text);
      OpenDialog1.Execute;
      if IsDuplicate(TreeView1.Selected.GetFirstChild, Text, True) then
      begin
        ShowMessage('같은 이름의 노드가 이미 존재합니다 !');
        Exit;
      end;
      with TreeView1.Items.AddChild(TreeView1.Selected, Text) do
      begin
        Data := TNodeData.Create;
        TNodeData(Data).Text := OpenDialog1.FileName;

... (후략)
```

처음 루트 노드를 추가할 때에는 Data 프로퍼티에 TNodeData 클래스의 인스턴스를 생성해서 대입하고, Text 멤버를 '루트 디렉토리 입니다 !'로 설정한다. 그리고, 다른 노드가 추가될 때에는 OpenDialog1 대화 상자를 띄워서 여기서 선택되는 파일의 이름을 Text 멤버로 설정해준다.
주의해서 보아야 할 것은 Data 프로퍼티에 TNodeData.Create 라는 클래스 constructor 를 이용해서 초기화를 한 후에, 실제 값을 TNodeData(Data)로 casting 해서 접근하는 테크닉이다.
이렇게 노드를 추가하면, 노드가 추가될 때마다 각 노드의 Data 프로퍼티가 TNodeData 클

래스의 인스턴스에 대한 참조값을 가지고 있게 되므로, 노드를 제거할 때에는 Data 프로퍼티에 저장된 값을 해제해 주어야 한다. 그러므로, Button2 의 OnClick 이벤트 핸들러도 다음과 같이 수정해 주어야 한다.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if TreeView1.Selected = nil then
  begin
    ShowMessage('선택된 것이 없습니다.');
    Exit;
  end;
  if TreeView1.Selected.Level = 0 then
  begin
    ShowMessage('Root 는 제거할 수 없습니다');
    Exit;
  end;
  if TreeView1.Selected.Data <> nil then
    TNodeData(TreeView1.Selected.Data).Free;
  TreeView1.Selected.Delete;
end;
```

이렇게 클래스를 이용하는 방법 외에 레코드를 사용하는 방법도 있다. 앞에서 보여준 TNodeData 클래스와 같은 내용을 레코드로 정의하고, 여기에 대한 포인터를 정의해서 사용하는 방법도 널리 쓰이고 있다. 예를 들어, 앞의 TNodeData 클래스와 동일한 일을 할 수 있도록 레코드를 다음과 같이 설정하고 사용할 수 있다.

```
pNodeData = ^TNodeData;
TNodeData = record
  Text: string;
end;
```

이렇게 하면 클래스 생성자를 사용하지 않고, 직접 레코드를 참조할 수 있는 장점이 있다. 다음과 같이 Data 프로퍼티에 적용해서 사용하면 된다.

```
Data := New(pNodeData);                    //레코드 포인터에 대한 메모리를 할당한다.
pNodeData(Data)^.Text := '루트 디렉토리 입니다 !';
```

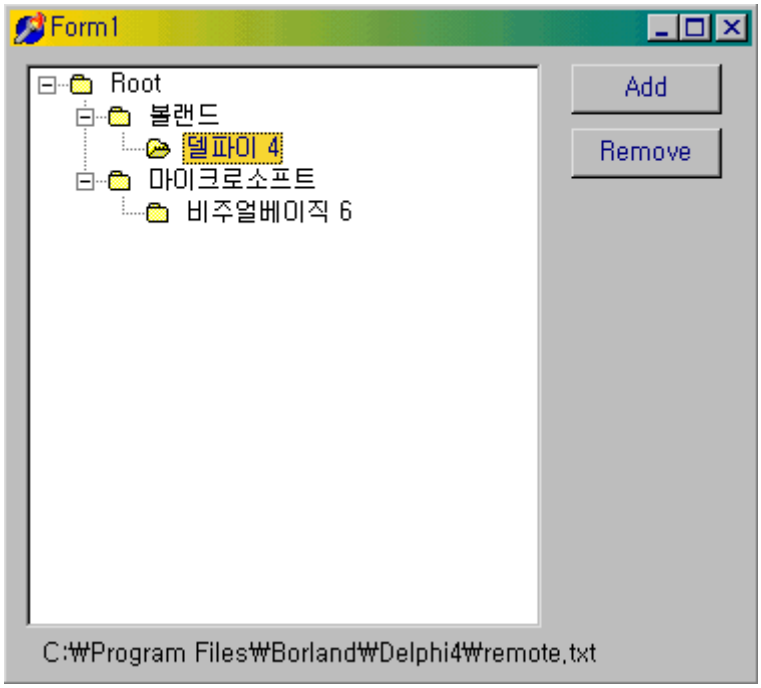
이렇게 노드에 데이터를 연결한 경우에는 메모리에서 해제할 때에도 다음과 같은 방법을 사용한다.

```
if TreeView1.Selected.Data <> nil then
  Dispose(pNodeData(TreeView1.Selected.Data));
```

이제 거의 완성되었는데, 노드를 추가할 때마다 파일을 선택해서 이 값들을 Data 프로퍼티에 저장했으면, 트리뷰의 특정 노드를 선택했을 때에는 그 노드에 저장된 값을 볼 수 있도록 하자. 이를 위해서 TLabel 컴포넌트를 하나 폼에 추가했었다.
이렇게 트리뷰의 노드를 선택했을 때 발생하는 이벤트가 OnChange 이벤트이다. 그러면, TreeView1 의 OnChange 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.TreeView1Change(Sender: TObject; Node: TTreeNode);
begin
  if TreeView1.Selected <> nil then
  begin
    if TreeView1.Selected.Data <> nil then
      Label1.Caption := TNodeData(TreeView1.Selected.Data).Text;
  end;
end;
```

소스가 그리 어려운 내용이 아니므로, 자세한 설명은 생략하기로 한다.
다음 그림은 이 프로그램의 실행화면이다.



리스트뷰 컴포넌트 시작하기

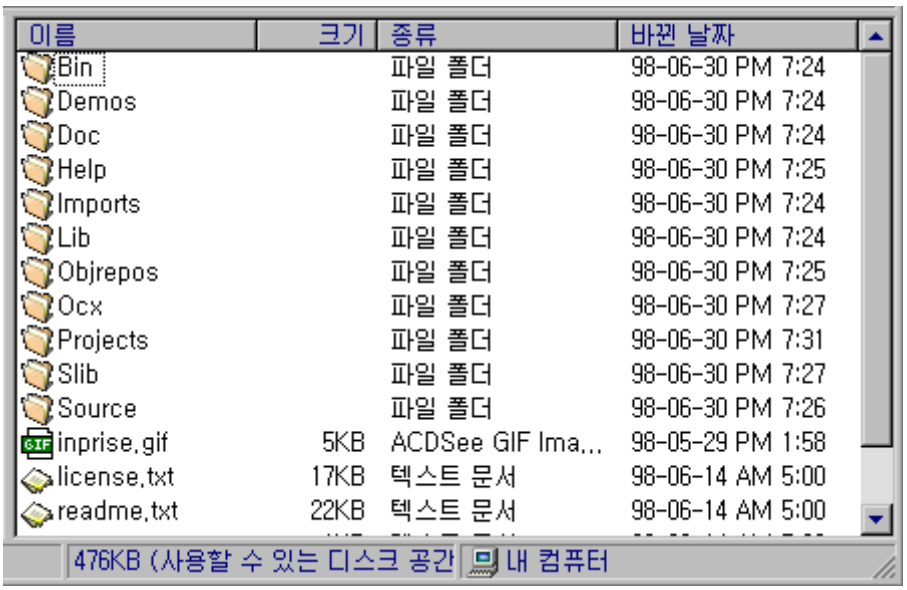
윈도우 95 에서 리스트뷰는 정말로 많이 사용되고 있다. 아마도 가장 자주 보게 되는 컨트롤이 이것이라고 생각되는데, 보통 윈도우 95 에서 파일을 보기 위해서 '내 컴퓨터' 아이콘을 더블 클릭하면 나타나는 윈도우가 바로 리스트뷰로 구성되어 있다.
리스트뷰 컨트롤에서 중요한 프로퍼티에는 다음과 같은 것들이 있다.

프로퍼티	설 명
Items	TListItem 의 컬렉션인 TListItems 데이터 형 프로퍼티로, 리스트의 내용을 결정한다. 각각의 아이템은 Caption, ItemIndex, StateIndex 프로퍼티를 가진다. ItemIndex 와 StateIndex 프로퍼티는 ImageList 컴포넌트와 연결하여 사용된다.
SubItems	TListItem 의 프로퍼티로 TStrings 데이터 형이다. 리스트뷰를 Detail 모드로 보면, SubItems 는 Caption 프로퍼티의 우측 각 컬럼의 텍스트 내용을 결정한다.
ViewStyle	탐색기 보기 형식의 큰 아이콘(vsIcon), 작은 아이콘(vsSmallIcon), 간단히(vsList), 자세히(vsReport)에 해당되는 보기 형식을 결정하는 프로퍼티이다.
Columns	TListColumn 의 컬렉션인 TListColumns 데이터 형 프로퍼티로, 리스트뷰를 Detail 모드로 볼 때의 각 컬럼의 형식을 결정한다.

리스트뷰는 이들 프로퍼티를 활용하여 어플리케이션을 작성한다. 트리뷰 보다는 비교적 쉬운 구조라고 할 수 있다. 기본적인 사용방법은 유사하다. 예를 들어, Edit 박스의 텍스트 내용을 새로운 리스트 아이템으로 리스트뷰에 추가시키려면 다음과 같이 코딩하면 된다.

```
var
  ListItem: TListItem;
begin
  ListItem := ListView1.Items.Add;
  ListItem.Caption := Edit1.Text;
```

ViewStyle 프로퍼티가 vsReport 로 설정된 경우에는, 탐색기의 '자세히' 보기 옵션을 선택한 경우를 생각하면 된다. 즉, 다음 그림과 같은 형식을 가진다.



여기에다가 새로운 컬럼을 추가하고자 하면 다음과 같은 식으로 코딩하면 된다.

```
var
  ListColumn: TListColumn;
begin
  ListColumn := ListView1.Columns.Add;
  ListColumn.Caption := Edit1.Text;
  ListColumn.Width := Length(Edit1.Text) * Font.Size;
```

여기에서 ListView.Items.Add 까지만 호출을 하여 ListItem 에 대한 프로퍼티만 설정한 경우에는 앞 그림의 파일 이름에 해당되는 것만 추가된 것이다. 우측의 3 가지 정보를 추가할 때에는 다음과 같이 SubItems 프로퍼티의 Add 메소드를 사용해야 한다.

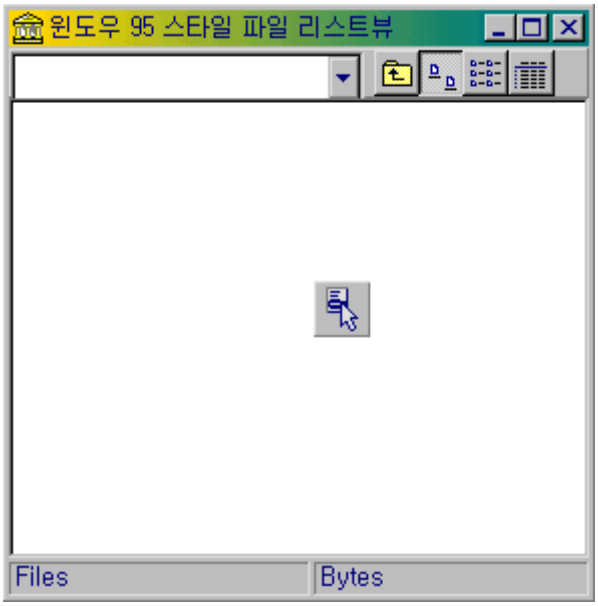
ListView1.Selected.SubItems.Add('서브 아이템 추가 !');

윈도우 95 스타일의 파일 리스트뷰 어플리케이션 제작

그러면, 리스트뷰를 기본 컴포넌트로 하여 이미지 리스트, 상태바 등을 활용한 윈도우 95 스타일의 파일 리스트뷰 어플리케이션을 제작해 보자.
지금까지 만들어본 예제 프로그램 중에서 600 라인이 넘는 비교적 큰 프로그램이기 때문에, 지면 관계상 이제까지의 방법처럼 차례차례 따라하도록 설명할 수는 없다. 소스는 Chap11 디렉토리의 Exam3.dpr 파일을 열면 모두 볼 수 있다. 여기서는 리스트뷰를 중심으로 필요한 부분만 설명하겠다. 소스에 주석을 달아놓았으므로 나머지 부분은 직접 소스를 참고하기 바란다.
여기에서 설명하지는 않지만, 소스에는 콤보 박스의 활용 방법과 콤보 박스에 이미지를 넣는 방법, 바이트 수를 KB 나 MB 단위로 바꾸는 유틸리티 함수 등의 많은 테크닉이 사용되었으므로 좋은 공부 재료가 될 것이다. 이 예제는 Markus Stephany(MirBir.St@T-Online.de)가 공개한 소스 코드에 기초한 것임을 미리 밝혀둔다. Markus Stephany 는 자신의 소스 코드를 공개하면서 Brad Stower 의 코드를 이용한 것이라고 밝혔으니, 결국에는 Brad Stower 에게 가장 감사해야 할 것 같다 (어쩌면 Brad Stower 역시 다른 사람의 소스를 참고했는지 모르지만).
파일의 정보를 얻기 위해서는 TSHFileInfo 구조체와 SHGetFileInfo 함수를 사용하는데 이들은 Win32 쉘의 기능을 이용하는 것이다. 이 어플리케이션을 실행하면 '내 컴퓨터'를 비롯한 여러가지 폴더를 더블 클릭하여 보는 모습과 거의 유사한 형태를 가진다. 그렇지만, 본질적으로는 조금 다른데 윈도우 탐색기를 비롯한 윈도우 95 의 쉘 인터페이스는 NameSpace 라고 불리는 영역을 사용하며, 모든 윈도우 객체를 가상 객체로 다루게 된다. 그렇기 때문에 전통적인 디렉토리-파일 개념의 접근과는 차이가 있다. 여기에 대해서는 이 책의 후반부에서 다룰 기회가 있을 것이다 (이 장을 집필하는 도중의 계획으로는).
쉽게 말해, 가상 폴더에는 프린터나 제어판 등의 실제 디렉토리가 아닌 것들에 대한 접근이 가능하지만, 이번 장의 파일 리스트뷰에서는 전통적인 디렉토리-파일 접근 방법을 사용하므로 이들을 볼 수는 없다.

- 폼디자인

이 예제의 폼의 형태는 다음과 같다.



패널 하나를 기본적으로 폼에 올려 놓고 Align 프로퍼티를 alClient 로 설정하여 전체를 담을 수 있도록 하고, Panel2 를 Panel1 위에 올려 놓고 Align 프로퍼티를 alTop 으로 설정한 후 콤보 박스 1 개와 스피드 버튼 4 개를 패널에 정렬한다. 그리고 상태바 컴포넌트를 Panel1 에 올려 놓고, Align 프로퍼티를 alBottom 으로 설정한다. 여기에 리스트뷰 컴포넌트를 추가하고 Panel1 위에서 Align 프로퍼티를 alClient 로 설정하면 전체적인 모양이 완성된다. 스피드 버튼 4 개의 힌트를 보여주기 위해서, ShowHint 프로퍼티를 True 로 설정하고 각각의 비트맵을 적당하게 대입하고, 이들의 Hint 프로퍼티를 각각 '위로', '큰 아이콘', '작은 아이콘', '자세히'로 설정한다.

마지막으로 TPopupMenu 컴포넌트를 폼에 올려 놓고, 이 컴포넌트를 더블 클릭한 후 MenuItem 을 추가한다. 이 메뉴 아이템의 Caption 을 '파일 정보'로 설정한다. 이제 사용할 컴포넌트는 모두 추가하였다. 이들의 Name 프로퍼티를 다음과 같이 설정한다.

컴포넌트	Name 프로퍼티	컴포넌트	Name 프로퍼티
리스트뷰	Files	상태바(Status Bar)	FState
팝업 메뉴	Menu	메뉴 아이템	MenuItem
콤보 박스	ComboBox1	위로 스피드 버튼	SpeedButton1
큰 아이콘 스피드 버튼	SpeedButton2	작은 아이콘 스피드 버튼	SpeedButton3
자세히 스피드 버튼	SpeedButton4		

FState 상태바를 선택하고, 오브젝트 인스펙터에서 Panels 프로퍼티를 더블 클릭하여 프로퍼티 에디터를 띄우고, 2 개의 패널을 추가한다. 각 패널의 Text 프로퍼티는 '파일', '바이트'로 설정한다. 이제 폼 디자이너에서 해야할 일은 모두 다 끝났다.

- 전역 변수의 선언과 유틸리티 함수

이제 코딩을 할 차례인데, 먼저 파일과 이미지 리스트에 대한 처리를 하기 위해서 uses 절에 FileCtrl.pas 와 ImgList.pas 유닛을 추가한다. 또한, 쉘의 여러가지 처리를 위해서는 ShellAPI.pas 유닛도 추가해 주어야 한다.
그리고, 전역 변수를 다음과 같이 선언한다.

```
Smalls, Larges: TImageList;
Path: TFileName;                    //현재 디스플레이되는 패스
Drives: set of 0..25;               //드라이브에 대한 플래그
```

Smalls 와 Larges 전역 변수는 파일 리스트뷰에서 사용할 이미지 리스트의 내용을 담게될 전역 변수이다. Path 는 현재 디스플레이되는 패스의 내용을 반영한다. Drives 변수는 알파벳 A~Z 까지를 드라이브로 사용할 수 있다고 볼 때, 이를 정수형의 세트로 선언하여 사용한다.
그리고, 이 예제에서는 몇 가지 유틸리티 함수를 예제 내에 포함시켜 사용하고 있는데 이들을 정리하면 다음과 같은 것들이 있다. 구체적인 구현 방법은 소스 코드를 참고하기 바란다.

1. function GetKb(c: Integer): string;

이 함수는 파라미터로 넘겨주는 정수형을 'KB', 'MB' 단위로 계산하여 이를 문자열의 형태로 반환하는 함수이다.

2. function GetSize(c, typ: Integer): string;

파일을 ' KB' 크기 단위로 변환하여 문자열로 반환하는 함수이다. 파라미터 c 는 크기를 담게 되고, typ 파라미터에는 디렉토리나 파일의 종류를 넘겨주게 된다. 여기서 faDirectory 와 같이 디렉토리가 넘어가면 빈 문자열을 반환한다.

3. function GetMod(a: TFileTime): string;

TFileTime 데이터 형의 내용을 윈도우 탐색기에서 볼 수 있는 형태의 문자열로 변환해주는 함수이다. 최근에 변경된 내용에 대한 정보는 SHGetFileInfo 함수를 이용해서 얻을 수 있다.

4. function GetCount(a: Char; b: string): Integer;

문자열에 특정 문자가 몇 개 있는지 파악해서 그 수를 반환하는 함수이다. 이 함수가 사용되는 것은 패스를 문자열로 넘겼을 때 서브 디렉토리가 몇 개 있는지를 파악하기 위해서 사용된다. 예를 들어 'c:₩Program Files₩Borland₩Delphi 4₩Demos₩...' 이라고 하면 '₩' 문자가 5 개 존재하므로 GetCount 함수에 '₩'자와 이 문자열을 파라미터로 넘기면 5를 반환하게 된다.

5. function NumPos(a: Char; b: String; c: Integer): Integer;

이 함수는 앞의 GetCount 와 함께 연관되어 사용되는데, 첫번째 파라미터에 찾을 문자와 두번째 파라미터에 문자열을 넘겨주고, 마지막 파라미터로 몇 번째 것의 위치를 지정할 것인지를 결정하면 문자열에서의 특정 문자의 위치를 반환한다. 이 함수는 패스를 문자열로 사용하되, 이들을 이용해서 서브 디렉토리와 상위 디렉토리를 오갈 수 있도록 처리하기 위해서 사용하게 된다. 예를 들어 패스가 'c:₩Program Files₩Borland₩Delphi 4₩' 디렉토리가 현재의 패스라고 할 때, '위로' 버튼을 누르면 'c:₩Program Files₩Borland₩'가 패스로 변경되어야 한다. 이때 NumPos('₩', 'c:₩Program Files₩Borland₩Delphi 4₩', 3)을 호출하면 'c:₩Program Files₩Borland₩' 문자까지의 위치가 반환되므로, 여기까지만 복사해서 지정하면 된다.

6. procedure GetInfo(FileName: TFileName; var i: TSHFileInfo);

이 함수는 파일 이름이나 디렉토리 이름과, TSHFileInfo 형으로 선언된 변수를 파라미터로 넘기면 이 파일이나 디렉토리의 정보를 TSHFileInfo 형 변수에 담아서 반환하게 된다.

- 리스트뷰 컬럼의 설정

우선 '내 컴퓨터'와 같이 최상위 위치로 올라갔을 때와 디렉토리에 담긴 파일 들을 보여줄 때 리스트뷰의 컬럼의 설정이 달라지게 된다. '내 컴퓨터'의 위치라면 각 디스크 드라이브의 이름 이외에, 드라이브의 종류와 디스크 크기, 남은 공간을 나타내면 될 것이다. 이런 프로시저를 SetColumnForDrives 라고 선언하고, 이를 다음과 같이 구현한다.

```
procedure TForm1.SetColumnForDrives;
        //리스트뷰의 컬럼 헤더를 드라이브 특성에 맞게 설정한다.
begin
```

```
  with Files.Columns do
  begin
    Clear;
    with Add do
    begin
      Caption := '이 름';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
    with Add do
    begin
      Caption := '종 류';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
    with Add do
    begin
      Caption := '디스크 크기';
      Width := ColumnHeaderWidth;
      Alignment := taRightJustify;
    end;
    with Add do
    begin
      Caption := '남은 공간';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
  end;
end;
```

마찬가지로, 디렉토리를 선택했을 때에는 각각의 파일에 대한 컬럼을 설정해야 할 것이다. 여기서는 이름과 크기, 종류, 바뀐 날짜를 설정해야 할 것이다. 이 프로시저는 SetColumnForFiles 라고 선언하고, 이를 다음과 같이 구현한다.

```
procedure TForm1.SetColumnForFiles;
          //리스트뷰의 컬럼 헤더를 설정한다.
```

```
begin
  with Files.Columns do
  begin
    Clear;
    with Add do
    begin
      Caption := '이 름';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
    with Add do
    begin
      Caption := '크 기';
      Width := ColumnHeaderWidth;
      Alignment := taRightJustify;
    end;
    with Add do
    begin
      Caption := '종 류';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
    with Add do
    begin
      Caption := '바뀐 날짜';
      Width := ColumnHeaderWidth;
      Alignment := taLeftJustify;
    end;
  end;
end;
```

- 리스트뷰에 아이템 추가하기

헤더 컬럼을 설정할 때와 마찬가지로, 내 컴퓨터와 같이 드라이브를 나열할 때와 디렉토리에서 파일을 나열할 때의 구현 방법이 다르다. 먼저 드라이브의 내용을 나열하는 SetDrives 프로시저를 선언하고, 이를 다음과 같이 구현한다.

```
procedure TForm1.SetDrives;      //리스트뷰에 시스템의 드라이브를 보여 준다.
var
  ct: Byte;
  new: TListItem;
  info: TSHFileInfo;
  drv: string;
```

new 변수에는 추가한 리스트 아이템을 대입하여, 여러가지 프로퍼티를 설정하도록 할 것이며, TSHFileInfo 구조체 형인 info 변수는 GetInfo 함수를 호출하여 드라이브에 대한 정보를 담게 된다. drv 변수에는 드라이브의 이름을 저장한다.

```
const
  drt: array[0..6] of string = ('알 수 없는 장치', '루트 디렉토리 없음', '플로피 디스크',
    '로컬 드라이브', '네트워크 드라이브', 'CD-ROM 드라이브', 'RAM 디스크');
```

drt 상수는 GetDriveType API 함수를 호출했을 때 반환하는 값이 0~6 까지의 정수이기 때문에 문자열로 쉽게 보여줄 수 있도록 선언하였다.

```
begin
  SpeedButton1.Enabled := False;
  Files.Items.BeginUpdate;
  Files.Items.Clear;
  SetColumnForDrives;
  Screen.Cursor := crHourGlass;
```

여기까지의 코드는 그다지 어려울 것이 없다. SpeedButton1 은 상위 레벨로 올라가는 버튼이므로, 현재 더이상 올라갈 것이 없는 상황에서는 Enabled 프로퍼티를 False 로 설정해야 할 것이다. 그리고, BeginUpdates 메소드는 아이템의 업데이트를 시작할 때 성능을 향상시켜주는 메소드로, 리스트뷰 외에도 트리뷰를 비롯한 여러가지 컨트롤에 있는 메소드이다. 항상 컨트롤을 업데이트할 때에는 이 메소드를 호출하는 것을 습관화하는 것이 좋다.

```
  for ct := 0 to 25 do                          //모든 드라이브를 읽어 온다.
    if ct in Drives then
    begin
      new := Files.Items.Add;
```

```
    drv := Char(ct + Ord('A')) + ':₩';          //드라이브의 루트 패스
    GetInfo(drv, info);                         //드라이브의 쉘 정보를 읽어온다
    new.Caption := info.szDisplayName;
    new.ImageIndex := info.iIcon;
    new.SubItems.Add(drt[GetDriveType(PChar(drv))]);
    new.SubItems.Add(GetKB(DiskSize(ct + 1)));
    new.SubItems.Add(GetKB(DiskFree(ct + 1)));
    new.SubItems.Add('드라이브');               //아이템의 종류 결정
    new.SubItems.Add(drv);                      //OnDoubleClick 이벤트 핸들러를 위해
  end;
```

컴퓨터에 있는 모든 드라이브를 for 루프를 통해 읽어와서, 이 드라이브가 Drives 에 있는지 검사한다. Drives 변수는 폼의 OnCreate 이벤트 핸들러에서 컴퓨터에 사용가능한 드라이브를 모두 저장하게 된다. new 변수는 새롭게 추가된 리스트 아이템을 저장해서 처리하며, drv 변수는 드라이브의 루트 패스 문자열로 지정한다. 일단 드라이브 루트 패스 문자열을 저장했으면, GetInfo 함수를 호출하여 TSHFileInfo 데이터 형인 info 변수에 드라이브의 정보를 담아 온다. 이 구조체의 szDisplayName 멤버는 드라이브의 이름이 저장되어 있으며, iIcon 멤버에는 시스템 이미지 리스트의 이미지 인덱스 값이 지정되어 있다. 폼의 OnCreate 이벤트 핸들러에서 시스템 이미지 리스트를 Smalls, Larges 이미지 리스트 변수에 저장하게 되므로, 이 인덱스 값을 그대로 이용할 수 있다.
SubItems 프로퍼티를 5 개 Add 하였는데, 순서대로 드라이브의 종류와 디스크의 크기, 남은 디스크 용량, 아이템의 종류(드라이브), 드라이브 루트 패스 문자열이 저장된다. 이들은 각각 SubItems[0] ~ SubItems[4]로 접근이 가능하다. 예를 들어 SubItems[3]에는 아이템의 종류인 '드라이브'가 들어 있고, SubItems[4]에는 드라이브의 루트 패스 문자열이 저장된다.

```
  Files.Items.EndUpdate;
  Screen.Cursor := crDefault;
  FState.Panels[0].Text := IntToStr(Files.Items.Count) + ' 객체';
  FState.Panels[1].Text := '';
  ComboBox1Update;
end;
```

리스트뷰의 변경이 끝났으면, 내용을 EndUpdate 메소드를 호출하여 업데이트 한다. 커서의 모양을 다시 원래의 형태로 바꿔 주고, 드라이브의 수를 상태바의 첫번째 패널에 표시한다. 마지막으로 콤보 박스의 내용을 업데이트하도록 콤보박스의 OnUpdate 이벤트 핸들러

를 호출한다.
비슷한 방식으로 먼저 디렉토리의 내용을 나열하는 SetFiles 프로시저를 선언하고, 이를 다음과 같이 구현한다. 이 프로시저는 SetDrives 프로시저의 호출을 포함하기 때문에, 다른 이벤트 핸들러에서는 SetFiles 프로시저만 호출해주면 된다.

```
procedure TForm1.SetFiles;
var
  ct: Integer;
  res: Word;
  rec: TSearchRec;
  new: TListItem;
  info: TSHFileInfo;
  oldstyle: TViewStyle;
```

로컬 변수의 선언부에서 SetDrives 프로시저와 다른 몇 개의 변수가 있다. res 변수는 파일을 나열할 때 사용하는 FindFirst, FindNext 함수의 결과값을 저장하는 변수로, 값이 0 이면 해당되는 파일이 있다는 의미가 된다. rec 변수 역시 FindFirst, FindNext 함수와 함께 사용되는 변수로, 9 장에서 이미 이들에 대해 설명한 바 있으므로 자세한 내용은 생략하겠다. oldStyle 변수는 현재의 리스트뷰 보기 스타일을 저장했다가 나중에 다시 복구하기 위해서 선언한 변수이다. ct 변수는 SetDrives 와는 달리 모든 파일 크기의 총합을 내어 저장하는 변수로 사용된다.

```
begin
  Caption := '윈도우 95 형 파일 리스트뷰 [' + Path + ']';
  if Path = 'Drives' then
  begin                              //드라이브를 보여주어야 한다면 ...
    SetDrives;
  end
  else
  begin
    SpeedButton1.Enabled := True;  //드라이브를 보여주는 것이 아니면, 더 상위로 올라갈 수 있다.
```

여기까지의 코드는 그다지 어렵지 않을 것이다. 우선 폼의 캡션에 패스를 같이 표시하고, 드라이브를 보여주어야 하는 경우라면 SetDrives 프로시저를 호출한다.

```
    Files.Items.BeginUpdate;
```

```
oldstyle := Files.ViewStyle;
Files.ViewStyle := vsList;        //속도가 빠르다고 한다.
Files.Items.Clear;
Screen.Cursor := crHourGlass;
if Path[Length(Path)] <> '₩' then Path := path + '₩';
```

리스트뷰를 업데이트 하기 위해서 우선 BeginUpdate 메소드를 호출하고, ViewStyle 프로퍼티를 vsList 로 설정한다. 이 프로퍼티를 변경하는 것은 업데이트 속도가 가장 빠르기 때문이다. 이를 위해 oldStyle 변수에 현재의 ViewStyle 을 저장했다가, 나중에 복구한다. 그리고, 패스의 마지막 문자가 '₩'가 아니면 이를 추가한다.
다음의 코드는 디렉토리에 있는 볼륨 ID 와 '.'과 '..' 디렉토리를 제외한 모든 파일과 디렉토리를 검색해서 이들의 정보를 리스트뷰의 아이템으로 추가하는 코드이다. 9 장에서 설명한 FindFirst, FindNext 함수의 사용하여 구현한다.

```
FillChar(rec, SizeOf(TSearchRec), 0);
ct := 0;
res := FindFirst(Path + '*.*', faAnyFile - faVolumeId, rec);
while res = 0 do
begin          //결과가 0 이 아니면 더 이상 파일이 없는 것임
               //'.'과 '..'은 제외한다.
  if not (((rec.Attr and faDirectory) > 0) and
    ((rec.Name = '.') or (rec.Name = '..'))) then
  begin
    new := Files.Items.Add;
    GetInfo(Path + rec.Name, info);  //파일이나 폴더의 쉘 정보를 얻어온다.
    new.Caption := info.szDisplayName;
    new.ImageIndex := info.iIcon;
    new.SubItems.Add(GetSize(rec.Size, rec.Attr));
    new.SubItems.Add(info.szTypeName);
    new.SubItems.Add(GetMod(rec.FindData.ftLastWriteTime));
    if (rec.Attr and faDirectory) > 0 then new.SubItems.Add('디렉토리')
    else new.SubItems.Add('파일');
    new.SubItems.Add(Path + rec.Name);
    Inc(ct, rec.Size);                    //파일 크기를 더한다.
  end;
  res := FindNext(rec);
```

```
    end;
    FindClose(rec);
```

새로운 아이템을 추가할 때, 유틸리티 함수인 GetSize, GetMod 함수를 이용하여 파일이나 디렉토리의 크기와 변경된 시간을 문자열로 얻어오게 되며, 마지막으로 4 번째로 Add 하는 SubItems[3]에 해당되는 내용은 디렉토리이면 '디렉토리', 파일이면 '파일'로 설정한다. 그리고, 각 파일이 크기를 계속 더해서 디렉토리의 파일 크기의 합을 표시하는데 사용한다.

```
    Files.CustomSort(@SortProc, 4);
    Files.ViewStyle := oldstyle;
    Files.Items.EndUpdate;
    FState.Panels[0].Text := IntToStr(Files.Items.Count) + ' 객체';
    FState.Panels[1].Text := GetKB(ct);
    Screen.Cursor := crDefault;
    ComboBox1Update;
  end;
end;
```

디렉토리를 파일에 우선하여 배열하도록 하기 위해 사용자 정의 정렬 루틴을 먼저 호출한다. 정렬하는 방법에 대해서는 다시 설명할 것이다. 그리고, ViewStyle 프로퍼티의 값과 커서의 값을 복구하고, 상태바의 내용을 업데이트 한다. 마지막으로 콤보 박스의 내용을 업데이트 한다.

- 사용자 정의 정렬 루틴의 활용

리스트뷰 이외에도 동적으로 값을 포함하게 되는 다른 여러가지 컨트롤(리스트 박스, 콤보 박스, 트리 뷰 등)에서 정렬 루틴은 자주 사용되는 중요한 부분이다. 보통 다른 컨트롤 처럼 아이템의 텍스트 만을 정렬하면 되는 경우에는 컨트롤의 AlphaSort 메소드를 이용하여 아스키 배열의 순서대로 간단하게 정렬을 할 수 있다. 그렇지만, 객체를 포함하고 있는 컨트롤이거나 다른 정보를 가지고 있어서 이들의 내용에 따라서 다르게 정렬을 해야할 경우에는 개발자가 직접 정렬에 대한 코드를 제공해야 한다.
이런 정렬 루틴을 사용자 정의 정렬(CustomSort)라고 한다.
사용자 정의 정렬 루틴은 기본적으로 사용 방법이 어느 컨트롤이나 동일하다. 일단 정렬 루틴을 작성해야 하는데, 이때 이 루틴은 콜백 함수로 사용된다. 콜백 함수에 대해서는 44 장에서 자세하게 설명하므로 이를 참고하기 바란다.
간단하게 설명하면 파라미터의 수와 데이터 형이 동일한 함수를 작성하고, 이 함수의 주소

를 CustomSort 메소드에 넘겨주면 된다. CustomSort 메소드는 2 개의 파라미터를 가지는데, 첫번째 파라미터에는 정렬 루틴으로 사용될 콜백 함수의 주소를 지정하고, 두번째 파라미터에는 옵션에 해당되는 내용을 설정하게 된다. 간단하게 사용 방법의 예를 들면 다음과 같다.

```
function CustomSortProc(Item1, Item2: TListItem; ParamSort: integer): integer; stdcall;
begin
  Result := -lstrcmp(PChar(TListItem(Item1).Caption), PChar(TListItem(Item2).Caption));
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ListView1.CustomSort(@CustomSortProc, 0);
end;
```

각각의 컨트롤마다 CustomSort 의 파라미터의 내용은 다를 수 있기 때문에, 도움말을 참고하여 적절한 루틴을 작성해야 할 것이다.
그러면, 이번 예제에서 작성한 정렬 루틴을 살펴 보자.
윈도우 95 스타일의 리스트뷰이므로, 정렬 루틴에서 고려해야 할 것은 '자세히' 보기 스타일인 vsReport 로 ViewStyle 프로퍼티가 설정된 경우, 파일의 이름 외에도 크기와 종류, 변경된 날짜에 대한 컬럼 헤더를 클릭하면 이들의 속성에 따라 정렬하도록 해야 한다는 것이다. 이때 Path 변수의 내용이 'Drives'라면 디렉토리에 대한 내용이 아니므로 정렬 루틴을 호출하지 않아야 한다. 그리고, 윈도우 탐색기처럼 컬럼 헤더를 반복해서 클릭하면 오름차순과 내림차순으로 번갈아가면서 정렬하도록 하기 위해서는 이들의 정보를 가지고 있을 전역 변수를 하나 선언해서 사용해야 할 것이다.
전역 변수 ReverseOrder 를 전역 변수 선언부(Form1 등의 변수가 선언된 부분)에 다음과 같이 추가한다.

```
ReverseOrder: array[0..4] of Boolean;
```

이 변수는 각각의 정렬이 오름차순인지, 내림차순인지를 결정한다. 배열로 선언한 이유는 각각의 정렬 기준에 따라서 오름차순 여부를 다르게 설정할 수 있도록 하기 위해서이다. 정렬할 요소는 0~4 까지 5 가지이다. 0 번에 해당되는 것은 파일의 이름이다. 이것은 아이템의 캡션에 해당된다. 1~3 에 해당되는 3 가지는 컬럼 헤더의 3 가지가 대응된다. 즉, 1 번은 SubItems[0]인 파일의 크기, 2 번은 SubItems[1]인 파일의 종류, 3 번은 SubItems[2]인 바뀐 날짜가 해당된다. 그렇다면, 마지막 4 번은 무엇일까 ? 컬럼 헤더에 나타나 보이

지는 않지만 디렉토리인지 파일인지 여부에 따라서 결정하는 것이다. 그러므로, SetFiles 프로시저에서 CustomSort(@SortProc, 4)의 의미는 디렉토리를 먼저 보여주고, 파일을 보여주라는 의미인 것이다.
처음에 파일을 보여줄 때 ReverseOrder 의 값을 초기화 할 필요가 있으므로, SetFiles 프로시저에 다음과 같은 코드를 추가한다.

```
procedure TForm1.SetFiles;        //리스트뷰에 파일과 폴더를 보여준다.
var
  ... (중략)

begin
  for i := Low(ReverseOrder) to High(ReverseOrder) do
    ReverseOrder[i] := False;

 ... (후략)
```

그리고, 컬럼 헤더를 클릭할 때마다 이를 이용해서 정렬하도록 호출해야 할 것이므로 Files 리스트뷰 컴포넌트의 OnColumnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.FilesColumnClick(Sender: TObject; Column: TListColumn);
begin
  if not (Path = 'Drives') then
  begin
    Files.CustomSort(@SortProc, Column.Index);
    ReverseOrder[Column.Index] := not ReverseOrder[Column.Index];
  end;
end;
```

즉, 패스가 디렉토리인 경우에 클릭한 컬럼의 인덱스를 넘겨서 컬럼 내용에 따라 정렬하고, 정렬 순서의 내용을 변경한다.
그러면, 실제 정렬을 담당할 콜백 함수를 작성하도록 하자. 기본적으로 파라미터의 수와 데이터 형만 일치하면 되므로, 함수의 이름이나 파라미터의 이름은 아무런 상관이 없다. 다음과 같이 콜백 함수를 작성한다.

```
function SortProc(I1, I2: TListItem; ColumnNo: Integer): Integer; stdcall;
var
  i: Integer;
```

```
  s1, s2: string;
  ds1, ds2, ts1, ts2: string;
```

내부에서 사용할 지역 변수를 먼저 선언한다. i 는 결과 값을 임시로 저장하기 위한 것으로, 콜백 함수 마지막 부분에서 ReverseOrder 변수의 값에 따라 이 값의 부호를 바꿔서 결과 값을 반환한다. 결과 값은 보통 1, 0, -1 중 하나를 반환하는데, 2 개의 아이템의 값이 같으면 0 을 반환하고, 첫번째 아이템이 더 크면 1(사실 양수면 된다), 작으면 -1(음수면 된다)을 반환한다.
s1, s2 는 코드의 중복을 막기 위해 임시로 문자열을 저장하기 위해 사용되며, ds1~ts2 는 날짜와 시간을 비교하기 위해서 사용된다. ColumnNo 파라미터로 클릭한 헤더의 인덱스가 넘어오므로, 이를 바탕으로 정렬 루틴을 다르게 작성해야 할 것이다.

```
begin
  i := 0;
  case ColumnNo of
    0: i := CompareStr(I1.Caption, I2.Caption);
```

ColumnNo 가 0 이면 '이 름' 헤더를 클릭한 경우이므로 아이템의 캡션을 서로 비교하면 된다. 이와 같이 CompareStr 함수를 사용하면 두 문자열을 비교하여 1, 0, -1 을 반환하므로, 코딩이 편하다.

```
    1: begin
         s1 := I1.SubItems[0];
         s2 := I2.SubItems[0];
         if s1 = '' then s1 := '0 KB';
         if s2 = '' then s2 := '0 KB';
         s1 := Copy(s1, 0, Length(s1) - 3);
         s2 := Copy(s2, 0, Length(s2) - 3);
         if StrToInt(s1) > StrToInt(s2) then i := 1 else i := -1;
         if StrToInt(s1) = StrToInt(s2) then
           i := CompareStr(I1.Caption, I2.Caption);
       end;
```

'크 기' 헤더를 클릭한 경우이다. 먼저 s1, s2 변수에 아이템의 문자열을 대입한다. 그리고, 디렉토리인 경우에는 문자열의 내용이 비어있게 되므로 값을 '0 KB'로 설정한다.
Copy 함수를 사용하여 뒤의 3 자를 잘라낸다. 즉, ' KB'를 문자열에서 삭제한다. 이제 숫

자만 남게 되었으므로 이들을 StrToInt 함수를 이용하여 정수로 변환하여 비교한다. 만약 크기가 같다면 파일 이름 순으로 정렬한다.

```
2: begin
     i := not CompareStr(I1.SubItems[1], I2.SubItems[1]);
     if (I1.SubItems[1] = I2.SubItems[1]) then
       i := CompareStr(I1.Caption, I2.Caption);
   end;
```

'종 류' 헤더를 클릭한 경우로, 단순히 SubItems[1]의 문자열을 직접 비교하면 된다. 종류가 같은 경우에는 아이템의 캡션을 비교한다.

```
3: begin
     s1 := I1.SubItems[2];
     s2 := I2.SubItems[2];
     ds1 := Copy(s1, 0, Pos(' ', s1));
     ds2 := Copy(s2, 0, Pos(' ', s2));
     ts1 := Copy(s1, Pos(' ', s1) + 2, Length(s1) - Pos(' ', s1) - 1);
     ts2 := Copy(s2, Pos(' ', s2) + 2, Length(s2) - Pos(' ', s2) - 1);
     if Length(ts1) = 7 then Insert('0', ts1, 4);
     if Length(ts2) = 7 then Insert('0', ts1, 4);
     if Copy(ts1, 4, 2) = '12' then
     begin
       ts1[4] := '0';
       ts1[5] := '0';
     end;
     if Copy(ts2, 4, 2) = '12' then
     begin
       ts2[4] := '0';
       ts2[5] := '0';
     end;
     i := CompareStr(ds1, ds2);
     if ds1 = ds2 then i := CompareStr(ts1, ts2);
   end;
```

'바뀐 날짜' 헤더를 클릭한 경우이다. SubItems[2] 문자열을 s1, s2 로 대입한 후, 이 문자

열을 다시 날짜와 시간 부분으로 쪼갠 후 비교한다. 이때 시간 부분의 길이를 맞추기 위해 시간이 한 개의 문자로 되어 있는 부분에는 앞에 '0'을 추가한다(예: AM 1:23 은 AM 01:23 으로). 그리고, 12 시인 경우에는 00 시로 변경해야 올바른 정렬이 된다.
이 루틴은 과거에 필자가 사용하던 처리 루틴을 그대로 가져와서 썼기 때문에 다소 복잡하다. 그렇지만, 아마도 잘 찾아보면 시간과 날짜, 문자열을 잘 변환해서 비교할 수 있는 루틴이 아마도 존재할 것이다. 필자가 이 부분을 수정하지 않은 이유는 기본적으로 잘 동작하기 때문이다 (물론 필자가 게으른 탓도 있다). 시간이 되는 독자는 이 부분을 조금 우아하게 수정해보기 바란다.

```
 4: begin
      i := CompareStr(I1.SubItems[3], I2.SubItems[3]);
      if (I1.SubItems[3] = I2.SubItems[3]) then
        i := CompareStr(I1.Caption, I2.Caption);
    end;
```

이 부분은 SetFiles 프로시저에 의해 파라미터가 4 가 넘어온 경우에만 실행된다. 디렉토리를 파일 보다 먼저 정렬하는 역할을 하는데, 역시 문자열을 비교하여 정렬한다.

```
end;
  if ReverseOrder[ColumnNo] then i := -i;
  Result := i;
```

마지막으로 ReverseOrder 변수 배열의 해당 필드의 내용을 검사하여, True 이면 값의 부호를 변경하고 이를 반환한다.

- 시스템 이미지 리스트 얻는 방법과 패스 변경의 구현

마지막으로 패스의 변경과 시스템 이미지 리스트를 얻는 방법을 설명하겠다. 나머지 부분도 공부할 만한 내용이 많지만, 지면 관계상 소스를 직접 참고하기를 바란다. 주석을 어느 정도 달아놓았기 때문에, 델파이의 도움말의 도움을 받아가면서 부족하면 쉽게 이해할 수 있을 것으로 믿는다.
폼의 OnCreate, OnDestory 이벤트 핸들러의 내용을 살펴 보면 다음과 같다.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Integer(Drives) := GetLogicalDrives;   //PC 가 가지고 있는 드라이브를 지정한다.
```

```
  CreateImages;                          //리스트뷰의 이미지 리스트를 설정한다.
  Path := ExtractFilePath(Application.ExeName);
  SetFiles;                              //리스트뷰를 업데이트 한다.
  SetColumnForFiles;                     //컬럼 헤더를 설정한다.
end;
```

GetLogicalDrives 함수는 PC 가 가지고 있는 논리적 드라이브의 세트를 반환하는 값으로, 이 함수를 이용하여 Drives 변수에 값을 저장한다. CreateImages 프로시저는 시스템 이미지 리스트를 리스트뷰의 이미지 리스트로 설정하는 역할을 하는데, 여기에 대해서는 다시 설명할 것이다. 일단 패스는 현재 실행 파일이 있는 패스로 설정하고, SetFiles 와 SetColumnForFiles 프로시저를 호출한다.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Smalls.Free;
  Larges.Free;
end;
```

시스템 이미지 리스트를 저장했던 전역 변수의 내용을 해제한다. 이런 부분을 소홀히 하면 리소스가 새게 되므로 주의하기 바란다.
그러면, CreateImages 프로시저는 어떻게 구현했는지 알아보자.

```
procedure CreateImages;          //시스템 이미지 리스트를 설정한다.
var
  SysIL: DWORD;
  SFI: TSHFileInfo;
begin
  Larges := TImageList.Create(Form1);
  Smalls := TImageList.Create(Form1);
```

SysIL 변수는 SHGetFileInfo 함수의 결과값을 저장되며, SFI 변수에 실질적인 내용이 저장된다. 그리고, TImageList 형의 전역 변수 Larges, Smalls 를 생성한다.

```
  SysIL := SHGetFileInfo('', 0, SFI, SizeOf(SFI), SHGFI_SYSICONINDEX or
    SHGFI_LARGEICON);
  if SysIL <> 0 then
```

```
  begin
    Larges.Handle := SysIL;
    Larges.ShareImages := TRUE;
  end;
  SysIL := SHGetFileInfo('', 0, SFI, SizeOf(SFI), SHGFI_SYSICONINDEX or
    SHGFI_SMALLICON);
  if SysIL <> 0 then
  begin
    SmallS.Handle := SysIL;
    SmallS.ShareImages := TRUE;
  end;
  Form1.Files.SmallImages := SmallS;
  Form1.Files.LargeImages := Larges;
end;
```

구현 방법은 단순하다. SHGetFileInfo 함수를 호출한 후, 이 함수의 결과를 이미지 리스트의 핸들로 사용하는 것이다. 이때 SHGetFileInfo 함수의 파라미터로 5 번째 파라미터의 내용에 SHGFI_LARGEICON, SHGFI_SMALLICON 이 설정되는 것에 따라 큰 아이콘과 작은 아이콘의 이미지 리스트를 가져오게 된다.
이번에는 패스를 변경하는 방법을 구현하도록 한다. 이 예제에서 패스의 변경이 요구되는 경우는 콤보 박스를 클릭하여 변경하는 것과 '위로' 스피드 버튼을 클릭한 경우, 그리고 폴더나 드라이브를 더블 클릭한 경우이다. 이 중에서 콤보 박스의 내용을 변경한 경우의 구현 방법은 소스 코드를 참고하기 바라며, 나머지 2 가지 경우에 대해서 살펴 보도록 하자.
폴더나 드라이브를 더블 클릭한 경우에는 리스트 뷰의 OnDoubleClick 이벤트 핸들러를 호출하게 된다.

```
procedure TForm1.FilesDblClick(Sender: TObject);
            //리스트 아이템을 더블 클릭하면 드라이브, 폴더를 연다.
begin
  if Files.Selected = nil then Exit;        //선택된 아이템이 없다.
  if Files.Selected.SubItems[3] = '디렉토리' then
  begin
    Path := Files.Selected.SubItems[4];
    SetFiles;
  end
```

먼저 알아야 하는 것은, 아이템의 SubItems[4]의 내용에는 현재 파일이나 디렉토리의 완전한 경로가 들어 있다는 것이다. 디렉토리를 더블 클릭한 경우, 선택된 아이템의 SubItems[3]의 내용은 '디렉토리'이다. 이런 경우에는 SubItems[4]의 경로를 Path 전역 변수로 지정한 후 SetFiles 프로시저를 호출하면 간단하게, 그 디렉토리로 들어가는 것을 구현할 수 있다.

```
  else
  if Files.Selected.SubItems[3] = '드라이브' then
  begin
    if not DirectoryExists(Files.Selected.SubItems[4]) then
      ShowMessage(Files.Selected.Caption + '에 접근할 수 없습니다.')
    else
    begin
      Path := Files.Selected.SubItems[4];
      SetColumnForFiles;
      SetFiles;
    end;
  end
```

선택된 아이템이 드라이브이면 (SubItems[3]의 값이 '드라이브'), 컬럼 헤더의 내용을 파일에 맞도록 다시 설정하고 SetFiles 프로시저를 호출한다. 이때 존재하지 않는 디렉토리를 더블 클릭한 경우에는 접근할 수 없는 메시지를 보여주도록 한다.

```
  else
    ShowProperties(Files.Selected.SubItems[4], Files.Selected.SubItems[3]);
end;
```

파일이 선택된 경우의 처리 방법으로, ShowProperties 프로시저를 호출하여 파일의 정보(경로와 크기)를 보여주도록 처리한다. 이 프로시저는 팝업 메뉴에서 파일 정보를 선택한 경우에도 호출된다. 자세한 것은 소스 코드를 참고하기 바란다.
여기에 비해, 상위 레벨로 옮겨가는 것은 다소 구현이 복잡하다. SpeedButton1 ('위로' 버튼)의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
```

```
  if Path[Length(Path) - 1] = ':' then       //루트 디렉토리에 있다면 ...
  begin
    Path := 'Drives';
    SetFiles;
    Files.SetFocus;
    if Files.Items.Count > 0 then Files.Selected := Files.Items[0];
  end
```

먼저, 패스가 루트 디렉토리에 있는지를 먼저 알아야 한다. 루트 디렉토리에 있다면 Path 변수의 마지막에서 두번째 문자가 ':'일 것이다 ('c:₩'를 생각해보라 !). 이 때에는 Path 변수값을 'Drives'로 설정하고, SetFiles 를 호출한다. 물론 SetFiles 내부에서는 SetDrives 프로시저를 호출할 것이다. 그리고, 첫번째 아이템(주로 A 드라이브)을 선택한다.

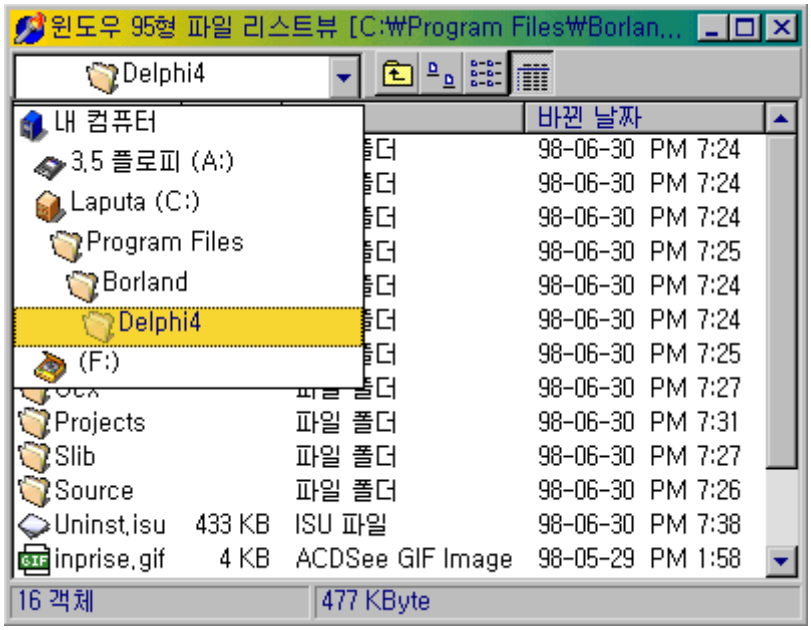
```
  else
  begin
    Path := Copy(Path, 1, Length(Path) - 1);       //일단 마지막 '₩'를 삭제한다.
    while Path[Length(Path)] <> '₩' do             //그리고, 제일 뒤의 디렉토리 이름을 삭제한다.
      Path := Copy(Path, 1, Length(Path)-1);
    SetFiles;
    Files.SetFocus;
    if Files.Items.Count > 0 then Files.Selected := Files.Items[0];
  end;
end;
```

이번에는 상위 디렉토리로 올라가도록 해야 한다. 일단 마지막의 '₩' 문자를 삭제한 뒤에 Copy 함수를 이용해서 '₩' 문자가 다시 나타날 때까지 한 글자씩 뒤에서부터 삭제해 나간다. 이렇게 하면 제일 뒤의 디렉토리 문자열이 삭제될 것이므로, 이 문자열을 Path 변수롤 하여 SetFiles 프로시저를 호출한다.

- 그 밖에

윈도우 95 스타일의 파일 리스트뷰를 구현하는 방법에 대해서 알아 보았다. 그 밖에도 이 예제에는 콤보 박스의 내용을 직접 그리는 방법과 탭을 이용하여 디렉토리의 깊이를 표현하는 방법, 크기와 시간, 날짜 등을 처리하는 여러가지 유틸리티 함수 등이 구현되어 있다. 이들은 소스 코드를 참고하여 익혀두기 바란다.
이제 이 예제를 한번 실행해보자. 스피드 버튼을 눌러 보이는 형태를 바꿔도 보고, 컬럼

헤더를 클릭하여 정렬도 한번 해도록 하라. 실행 화면은 다음과 같다.



툴바와 쿨바의 디자인

툴바는 보통 폼의 제일 윗부분에 위치하여, 버튼을 비롯한 여러가지 컨트롤을 담게 되는 일종의 패널이다. 쿨바 역시 일종의 툴바이지만, 밴드를 이용하여 크기를 변경하고, 이동을 할 수 있다.

버튼을 추가하는 것이 가장 흔한 툴바의 사용 방법이지만, 컬러 그리드나 스크롤 바, 라벨 등의 컨트롤도 추가할 수 있다. 보통 툴바를 구현하기 위해서는 패널을 폼에 추가하고, 스피드 버튼 등을 추가해 사용하거나, 툴바 컴포넌트를 이용하게 된다.

툴바 컴포넌트(TToolBar)를 사용하면, 버튼이나 다른 컨트롤을 자동으로 크기와 위치에 맞추어 정렬할 수 있으며, TToolButton 컨트롤을 버튼으로 사용할 경우에는 버튼들을 그룹으로 만들어 그 기능 별로 관리할 수 있으며, 그룹 별로 디스플레이 옵션을 다르게 줄 수도 있다. 쿨바 컴포넌트(TCoolBar)는 툴바 컴포넌트의 편리함에 밴드를 이용해 위치와 크기를 가변할 수 있는 성능을 제공한다.

필자의 개인적인 생각으로는 패널을 이용한 방법 보다는, 툴바나 쿨바 컴포넌트를 이용하라고 권하고 싶다. 이들은 윈도우의 컨트롤을 그대로 사용하는 것이기 때문에, 운영체제가 이들 컨트롤의 성능을 향상시키면, 어플리케이션의 성능도 같이 향상된다.

툴바의 사용 예제는 8 장에서 소개한 바 있으므로, 여기서는 툴바 컴포넌트의 기능에 대해서 간략하게 정리하고 넘어가도록 하겠다.

- 툴바 컴포넌트를 이용하여 툴바 생성하기

툴바 컴포넌트는 Win32 페이지에 위치하는데, 폼에 추가하면 자동적으로 폼의 가장 위쪽에 정렬된다. 사용을 위해서는 툴 버튼이나 다른 버튼을 바에 추가하여 사용한다.
툴 버튼은 툴바 컴포넌트와 작업하기 위해 디자인 된 버튼으로 스피드 버튼과 마찬가지로 푸시 버튼으로 사용할 수도 있으며, 토글과 라디오 버튼과 같은 용도로 사용하게 할 수도 있다.

1. 툴 버튼의 추가

툴 버튼을 툴바에 추가하려면, 툴바에서 오르쪽 버튼을 누르면 나오는 팝업 메뉴에서 New Button 메뉴를 선택하면 된다. 툴바에 있는 모든 툴 버튼 들은 같은 높이와 폭을 가지게 된다. 다른 컨트롤 들을 툴바에 추가하면, 이들 역시 동일한 높이를 가지게 된다.

2. 버튼에 이미지 대입하기

각 툴 버튼은 런타임에서 보여주게될 이미지를 결정하기 위해 ImageIndex 프로퍼티를 가지고 있다. 이 때 버튼에 추가할 이미지는 Images 프로퍼티에 대입된 ImageList 컴포넌트에 의해 결정되며, 하나의 이미지만 제공된 경우 비활성화된 형태를 보여주기 위해 이미지를 가공하게 된다.
또한, 툴 버튼에는 마우스 포인터가 위를 지나가거나 버튼이 비활성화된 이미지를 따로 지정하게 할 수 있는데, 이들을 각각 DisabledImages, HotImages 프로퍼티를 이용해서 설정한다.

3. 툴 버튼을 라디오 그룹처럼 사용하기

툴 버튼을 그룹지어 라디오 버튼처럼 사용하려면, 서로 연관시킬 버튼 들을 선택하고 이들의 Style 프로퍼티를 tbsCheck 로 설정한다. 그리고, Grouped 프로퍼티를 True 로 지정하면 된다. 이렇게 하면, 라디오 버튼을 사용하듯이 이들 중 하나만 선택하도록 하는 것이 가능하다. 서로 인접한 버튼 들의 Style 프로퍼티와 Grouped 프로퍼티가 각각 tbsCheck, True 이면 하나의 그룹으로 취급된다. 그룹을 해제하려면, 다음의 방법들 중에 하나를 이용하면 된다.

- Grouped 프로퍼티를 False 로 설정한다.
- Style 프로퍼티를 변경한다. 툴바에 공간을 확보하거나, 툴바의 그룹을 나누기 위해서 Style 프로퍼티의 값을 tbsSeparator 나 tbsDivider 로 설정한다.
- 툴 버튼 옆에 다른 컨트롤을 위치시킨다.

4. 토글되는 툴 버튼

버튼을 토글하여 사용하려면, Style 프로퍼티를 tbsCheck 로 선택하면 된다. 그룹을 짓고, 이들이 여러 개 눌려 있거나, 올라와 있도록 할 수 있게 하려면 AllowAllUp 프로퍼티를 이용하면 된다. 이 프로퍼티가 True 이면 여러 버튼 들이 한 번 클릭하면 누른 채로 있고, 다시 누르면 눌린 채로 있게 된다.

5. 툴 버튼에 메뉴 대입

툴 버튼을 선택하고 오브젝트 인스펙터에 보면 DropDownMenu 프로퍼티가 있다. 이 프로퍼티에 TPopupMenu 컴포넌트의 이름을 대입하고, AutoPopup 프로퍼티를 True 로 설정하면 버튼이 눌릴 때마다 팝업 메뉴가 뜨게 된다.

6. 툴바 숨기기와 보여주기

어플리케이션에 여러 개의 툴바를 사용할 경우에 이들 중 사용자가 보고자 하는 것만 보도록 할 수 있다. 이럴 때에는 툴바를 적당하게 보이고, 숨기기만 하면 된다. 이를 위해서는 Visible 프로퍼티를 이용한다.

- 쿨바 컴포넌트의 활용

쿨바 컴포넌트는 독립적으로 이동과 크기 변화가 가능한 밴드를 가지고 있는 컴포넌트이다. 사용자는 밴드를 위치시키기 위해서는 단순히 각 밴드의 좌측에 있는 그립을 클릭하여 드래그하면 된다.
쿨바 컴포넌트 역시 Win32 페이지에 있으며, 이를 폼에 추가하면 폼의 가장 위에 위치한다. 참고로, TWindow 컨트롤에서 상속받은 윈도우 컨트롤 들만 분리돈 밴드에 위치시킬 수 있다. 그러므로, 스피드 버튼이나 라벨과 같이 그래픽 컨트롤을 상속한 경우에는 분리된 밴드에 보이게 할 수 없다.
쿨바 컴포넌트에서 사용하는 중요한 프로퍼티를 소개하면 다음과 같은 것들이 있다.

프로퍼티	설 명
AutoSize	이 값을 True 로 설정하면 밴드 들에 맞게 크기를 자동으로 조절한다.
FixedSize	이 값을 True 로 설정하면 밴드의 높이를 일정하게 유지한다.
Vertical	이 값이 True 이면 수평보다 수직에 맞추어 밴드를 설정한다.
ShowText	이 값을 False 로 설정하면 각 밴드의 텍스트 프로퍼티를 런타임에 보여주지 않

	도록 한다.
BandBorderStyle	bsNone 을 선택하면 바 주위의 보더를 제거할 수 있다.
FixedOrder	이 값을 True 로 설정하면, 사용자가 밴드의 순서를 바꾸지 못하도록 한다.
Bitmap	배경 비트맵을 설정한다.
Images	각 밴드의 프로퍼티로 이미지를 저장할 TImageList 객체를 지정한다.
ImageIndex	각 밴드에 대입할 이미지의 이미지의 인덱스이다.

정 리 (Summary)

이번 장에서는 Win32 공통 컨트롤의 사용 방법에 대해서 알아 보았다. 물론 다루지 못한 많은 수의 컴포넌트가 있지만, 이들도 사용자 인터페이스를 향상시키는 데에 유용하므로 델파이의 도움말과 데모 프로그램 등을 통해 사용방법을 익혀둘 것을 권하고 싶다.
특히, 이번 장에서 다룬 트리뷰와 리스트뷰 컨트롤은 사용 하기에 따라서는 고급스러운 어플리케이션을 만들 수 있을 것이다.
이것으로 제 2 부의 내용을 모두 마친다. 비록 다루지 못한 컴포넌트 들이 많지만, 사용하는 방법은 대개 비슷하므로 여러 차례 연습해보면 사용하는 법을 쉽게 익힐 수 있을 것이다.
3 부에서는 데이터베이스 어플리케이션을 제작하는 기초적인 방법에서부터, 고급스러운 테크닉에 이르기까지 자세하게 알아보도록 할 것이다.