17장 클립보드를 통한 정보 공유

* 클립보드 활용법

* 독자적인 클립보드 데이터 형식 만들기

옛날 옛적, 인간들은 생존을 위해 투쟁했다. 그들은 어두운 동굴에 살았으며, 돌도끼나 창으로 사냥을 해서 먹고 살았다. 으르렁거리는 소리와 손짓 발짓을 섞어서 서로 의사를 교환하던 시절이었다. 그러나 그때에도 외계인들이 전해준 컴퓨터를 가지고 열심히 게임도 하고, 불법 복제도 하고 살았다는 이야기가 있다(?). 그때의 컴퓨터는 소프트웨어와 하드웨어의 한계 때문에 한 번에 단 하나의 애플리케이션만을 돌릴 수 있었으며, 애플리케이션들 사이에 정보를 교환하려면 디스크라고 불리는 납작한 돌판에 정보를 기록하는 방법밖에 없었다.

현대의 인간이 그때보다 얼마나 발전했는지는 모르겠지만, 어쨌든 하드웨어와 소프트웨어만큼은 발전했다고 할 수 있다. 윈도우즈 NT나 윈도우즈 9x 같은 멀티태스킹 운영체제에서는 동시에 여러 개의 애플리케이션들을 돌릴 수 있으므로 컴퓨터 사용자의 입장에서 보면 상당히 발전한 셈이다. 멀티태스킹 환경에서는 한 컴퓨터 안에서 서로 다른 애플리케이션들 사이에 정보를 교환할 수 있다. 윈도우즈의 경우에는 Win32 클립보드와 동적 데이터 교환(DDE, Dynamic Data Exchange)이라는 것을 통해서 정보를 공유한다. 한 애플리케이션이 클립보드를 지원하기만 한다면 역시 클립보드를 지원하는 다른 애플리케이션이 어떻게 구현되어 있는지를 알 필요 없이 클립보드를 통해서 데이터를 복사하거나 붙여넣을 수 있다.

이번 장에서는 델파이로 Win32 클립보드를 다루는 방법에 대해서 설명하겠다. 이 책의 이전 판들에서는 DDE도 다루었지만, 사실 COM이라는 좀더 진보된 기술이 나온 지금 시점에서 DDE 같은 사장된 기술에 귀중한 지면을 낭비할 필요는 없다고 생각한다. 23장, “COM과 ActiveX”에서 좀더 진보된 정보 공유 기술이라고 할 수 있는 COM에 대해 자세히 설명할 것이다. 그러나 간단히 구현할 수 있는 정보 공유 기법이라면 역시 클립보드이므로 이번 장에서는 클립보드에 대해서 설명하겠다.

클립보드 활용법

윈도우즈 프로그래밍 경험이 있는 사람이라면 Win32의 클립보드에 대해서 어느 정도 알고 있을 것이다. 윈도우즈 프로그래밍 초보자라도, 윈도우즈를 사용해 본 경험이 있는 이상 클립보드란 것이 어떤 것인지는 알고 있을 것이다.

문서를 다루는 대부분의 애플리케이션들에는 클립보드를 사용하는 편집 또는 Edit 메뉴가 있다. 그런데 클립보드란, 정확히 무엇을 말하는 것일까? 클립보드란 용어는 데이터를 담을 특정 메모리 영역과 애플리케이션이 그 메모리 영역에 데이터를 쓰거

나 거기에서 데이터를 읽는 데 필요한 Win32 API 함수들의 집합을 총체적으로 말하는 것이라고 할 수 있다. 델파이 코드 편집기에서 소스 코드를 복사해 메모장에 붙이는 것은 결국 Win32의 특정한 메모리 영역에 데이터를 쓰고 읽는 과정인 것이다.

그런데 단순히 데이터를 쓰고 읽는 것이 전부라면 클립보드 관련 API 함수는 단 두 개면 충분하겠지만, 클립보드의 작용은 좀더 복잡하다. 클립보드에 들어가는 데이터의 형식도 여러 가지이며, 또 클립보드의 내용이 변했는지를 알고 싶어하는 애플리케이션들도 있기 때문에 여러 가지 API 함수들이 존재한다. 단, 클립보드의 메모리 관리는 Win32가 알아서 하기 때문에 애플리케이션이 신경쓸 필요는 없다.

클립보드의 데이터 형식

클립보드에 복사하거나 클립보드로부터 붙여넣을 수 있는 데이터의 형식은 25가지가 미리 정의되어 있다. 그 중 가장 자주 쓰는 것들은 다음과 같다.

CF_BITMAP	비트맵 데이터
CF_DIB	색상 팔레트 정보가 포함된 비트맵 데이터
CF_PALETTE	색상 팔레트
CF_TEXT	각 줄이 CR/LF(캐리지 리턴/라인피드)로 끝나는 문자 배열. 가장 많이 쓰이는 형식이다.

그 밖의 형식들에 대해서는 Win32 API 도움말의 "SetClipboradData" 부분을 보기 바란다. 그리고 미리 정의된 형식 말고도 개발자가 직접 데이터 형식을 지정할 수 있는데, 그에 대해서는 이번 장에서 다시 이야기하겠다.

전통적인 윈도우즈 프로그래밍에서는 클립보드 관련 API 함수들을 이용해야 했으며, 클립보드에 담긴 데이터의 형식도 코드에서 직접 판단해야 했었다. 델파이는 Win32 클립보드를 캡슐화한 델파이 클래스의 인스턴스인 Clipboard를 전역 객체로 제공한다.

클립보드 활용 : 텍스트

16장, "MDI 애플리케이션"에서도 클립보드로 텍스트를 다루는 기법이 소개되었다. 16장의 예제 MDI 애플리케이션의 텍스트 편집 자식 폼 부분에 클립보드에 대한 내용이 나오는데, 여기에서 메뉴 항목들을 통해 텍스트 잘라내기/복사/붙여넣기 등을 구현하는 방법이 설명되었다.

그 MDI 애플리케이션에서 실제적인 텍스트 편집 기능을 가지고 있는 TMemo 컴포넌트는 자식 폼의 클라이언트 영역에 꽉차게 배치되었었다. TMemo 컴포넌트 자체에 전역 Clipboard 객체를 다루는 메소드가 포함되어 있다.
CutToClipBoard(), CopyToClipBoard(), 그리고 PasteFromClipBoard()가 바로 그것이다. ClearSelection()과 SelectAll()은 클립보드와는 별로 관계가 없는 메소드이지만, 클립보드에 텍스트를 복사하거나, 클립보드로부터 텍스트를 붙여넣을 때 꼭 필요한 메소드들이다. 목록 17.1은 16장의 예제 애플리케이션에 나온 Edit 메뉴 항목들의 이벤트 프로시저이다.

목록 17.1 클립보드로 텍스트를 다루는 예제 코드

```
procedure TMdiEditForm.mmiCutClick(Sender: TObject);
begin
  inherited;
  memMainMemo.CutToClipBoard;
end;

procedure TMdiEditForm.mmiCopyClick(Sender: TObject);
begin
  inherited;
  memMainMemo.CopyToClipBoard;
end;

procedure TMdiEditForm.mmiPasteClick(Sender: TObject);
begin
  inherited;
  memMainMemo.PasteFromClipBoard;
end;
```

목록 17.1을 보면 알겠지만, 단 한 번의 메소드 호출로 클립보드 작업을 끝낼 수 있다. 그리고 컴포넌트의 메소드를 사용하지 않고, Clipboard.AsText 속성을 통해서 직접 텍스트를 클립보드에 넣을 수도 있다. 16비트 환경에서 Clipboard.AsText 속성에 담을 수 있는 텍스트의 최대 크기는 255자였으며, 그보다 더 큰 문자열을 담으려면 Clipboard의 SetTextBuf()나 GetTextBuf() 메소드를 통해서 텍스트를 복사해야 했었다. 그러나 32비트 델파이에서는 AsText가 긴 문자열형이기 때문에 그러한 한계는 없어졌다. 물론, SetTextBuf()나 GetTextBuf()를 이용한 기법도 여전히 유효하다.

```
Clipboard.AsText := '델파이 5가 최고';
```



클립보드의 GetTextBuf()와 SetTextBuf() 메소드는 PChar형의 버퍼를 통해서 클립보드의 데이터를 쓰거나 읽는다. 이 두 메소드를 사용하기 위해서 PChar형의 변수를 선언할 필요는 없다. 그냥 긴 문자열형 변수를 PChar로 형 변환시키면 된다.

클립보드 활용 : 비트맵

클립보드는 이미지도 복사하거나 붙여넣을 수 있다. 16장의 예제 MDI 애플리케이션에도 그러한 기법이 나온다. 그 예제 애플리케이션에서 클립보드로 이미지를 다루는 이벤트 처리 메소드들이 목록 17.2에 나와 있다.



목록 17.2 클립보드로 이미지 다루는 루틴들

```
procedure TMdiBMPForm.mmiCopyClick(Sender: TObject);
begin
  inherited;
  ClipBoard.Assign(imgMain.Picture);
end;

procedure TMdiBMPForm.mmiPasteClick(Sender: TObject);
{ 이 메소드는 클립보드의 내용을 imgMain에 복사한다. }
begin
  inherited;
  // 클립보드의 내용을 imgMain에 복사
  imgMain.Picture.Assign(ClipBoard);
  ClientWidth := imgMain.Picture.Width;
  { 스크롤바의 범위를 비트맵의 크기에 맞게 조정한다. }
  VertScrollBar.Range := imgMain.Picture.Height;
  HorzScrollBar.Range := imgMain.Picture.Width;
end;
```



어떤 유닛에서 전역 변수 Clipboard를 사용하려면 그 유닛의 uses절에 ClipBrd를 추가해야 한다.

목록 17.2를 보면 mmiCopyClick() 이벤트 처리 메소드가 Clipboard.Assign() 메소드를 이용해서 이미지를 클립보드에 복사했음을 알 수 있다. 이렇게 하면 CF_BITMAP 형식을 지원하는 다른 Win32 애플리케이션(PBRUSH.EXE 등)이 클립보드의 내용을 사용할 수 있다.

mmiPasteClick()은 TImage의 메소드를 통해서 클립보드의 내용을 TImage에 복사하는데, TImage는 CF_BITMAP이나 CF_PICTURE이면 비트맵으로 표시한다. 그런 다음, mmiPasteClick()은 비트맵의 크기에 맞게 스크롤바들을 조정한다.



CF_PICTURE는 Win32의 표준 클립보드 데이터 형식이 아니다. CF_PICTURE는 비트맵뿐만 아니라, 메타 파일 등 TPicture에 사용할 수 있는 이미지 형식을 나타내기 위해 델파이가 정의한 것이다. 만일 개발자가 TPicture에 다른 그래픽 형식을 직접 등록하면 CF_PICTURE는 그 형식도 지원하게 된다. TPicture 호환 그래픽 형식에 대해서는 델파이 도움말의 TPicture 항목을 참고하기 바란다.

독자적인 클립보드 데이터 형식 만들기

주소 입력 프로그램을 사용한다고 생각해 보자. 만일 이전에 입력한 레코드와 아주 조금만 다른 레코드를 입력하는 경우, 이전 레코드를 클립보드에 복사해서 현재 레코드에 붙일 수 있다면 일일이 모든 필드들을 입력하지 않아도 되기 때문에 무척 편리할 것이다. 또한, 클립보드에 있는 내용을 편지 발송 폼에 직접 붙여넣을 수 있다

면 더욱 편리할 것이다. 그러나 클립보드는 주소 레코드의 필드 구성을 알지 못하므로 그냥 붙여넣을 수는 없다. 다행히 Win32 클립보드는 개발자가 독자적인 데이터 형식을 직접 등록할 수 있도록 한다. 이번에는 독자적인 클립보드 데이터 형식을 정의하고 사용하는 예제를 살펴보자. 이 예제를 통해서 독자적인 데이터 형식을 사용하는 방법뿐만 아니라, 그 데이터를 다른 애플리케이션들이 사용할 수 있도록 CF_TEXT 형식으로 클립보드에 저장하는 기법도 배울 수 있을 것이다.

클립보드 인식 객체 만들기

델파이의 객체 지향적 성격을 중요시하는 독자라면 새로운 클립보드 데이터 형식을 만든다고 했을 때 TClipboard 클래스를 상속하는 새로운 클립보드 클래스를 만들 생각을 하게 될 것이다. 나쁘지 않은 생각이다. TClipboard 클래스를 상속하는 클립보드 클래스를 만들어서 개발자가 정의한 형식의 데이터를 처리하도록 메소드를 재정의하거나 새로운 메소드를 만들면 된다. 그러나 문제는 새로운 형식을 추가할 때마다 클래스를 손봐야 하기 때문에 유지보수에 무리가 생기며, 서로 다른 클립보드 클래스들을 통해서 두 가지 이상의 클립보드 데이터 형식을 사용하려고 하면 클립보드 클래스끼리 충돌하는 현상도 생긴다.

좀더 나은 접근 방식이라면, 클립보드를 중심으로 클래스를 만들 것이 아니라 데이터를 중심으로 클래스를 만들고, 그 클래스가 클립보드, 즉 TClipboard를 인식(aware)하게 만드는 것이다. 델파이 컴포넌트들이 클립보드를 사용할 때에도 그러한 접근 방식을 사용한다. TMemo 컴포넌트는 자신의 데이터를 클립보드에 넣는 방법을 알고 있으며, TImage 역시 자신의 데이터를 어떻게 클립보드에 넣는지 알고 있다. 모든 컴포넌트들은 동일한 TClipboard 클래스를 사용하므로 상속 클래스간의 충돌 같은 일은 일어나지 않는다. 이번에 제시할 예제에서도 그러한 접근 방식을 사용한다. 이번 예제의 데이터 형식은 사람 이름, 나이, 생일 정보를 담는 레코드에 기반한 것이다. 목록 17.3은 데이터 형식과 그 데이터를 클립보드에 복사하거나 클립보드로부터 붙여넣는 메소드를 정의한 유닛의 소스 코드이다.

목록 17.3 Ch17\Clipbrd\cbdata.pas

```
unit cbdata;
interface
uses
  SysUtils, Windows, clipbrd;
```

```
const

  DDGData = 'CF_DDG'; // 클립보드 데이터 형식으로 등록할 상수
type

  // 클립보드에 저장할 데이터 형식을 뜻하는 레코드

  TDataRec = packed record
    LName: string[10];
    FName: string[10];
    MI: string[2];
    Age: Integer;
    BirthDate: TDateTime;
  end;

 { TDataRec을 감싸는 클래스를 정의한다. 이 클래스에는 TDataRec을 클립보드
   에 복사하거나, 클립보드로부터 가져오는 메소드가 포함된다. }
  TData = class
  public
    Rec: TDataRec;
    procedure CopyToClipBoard;
    procedure GetFromClipBoard;
  end;

var
  CF_DDGDATA: word; //RegisterClipboardFormat()의 리턴값을 저장할 변수

implementation

procedure TData.CopyToClipBoard;
{ 이 메소드는 TDataRec의 필드 Rec을 두 가지 형식, 즉 이진 데이터와 텍스트로
  클립보드에 복사한다. 이렇게 하면 두 형식 모두 클립보드에서 사용할 수 있게
  된다.}
const
  CRLF = #13#10;
var
  Data: THandle;
  DataPtr: Pointer;
  TempStr: String[50];
```

```
begin
  // 힙에서 SizeOf(TDataRec) 바이트를 할당한다.
  Data := GlobalAlloc(GMEM_MOVEABLE, SizeOf(TDataRec));
  try
    // 할당된 메모리 블록을 가리키는 포인터를 얻는다.
    DataPtr := GlobalLock(Data);
    try
      // Rec의 데이터를 메모리 블록에 넣는다.
      Move(Rec, DataPtr^, SizeOf(TDataRec));
          { 한번에 여러 개의 클립보드 데이터 형식을 클립보드에 넣을 때에는 우선
            Clipboard.Open을 호출해야 한다. 하나의 형식만 넣을 때에는 호출하지
            않아도 된다. }
      ClipBoard.Open;
      // 우선 독자적인 형식, 즉 이진 데이터 형식으로 복사한다.
      ClipBoard.SetAsHandle(CF_DDGDATA, Data);
      // 텍스트 형식으로 복사한다.
      with Rec do
        TempStr := FName+CRLF+LName+CRLF+MI+CRLF+IntToStr(Age)+CRLF+
                DateTimeToStr(BirthDate);
      ClipBoard.AsText := TempStr;
          { Clipboard.Open을 호출했으므로 Clipboard.Close도 호출해야 한다. }
      Clipboard.Close
    finally
      // 잠겨진 힙 메모리를 푼다.
      GlobalUnlock(Data);
    end;
  except
      { GlobalFree()는 예외가 발생했을 때에만 호출하면 된다. 예외가 없는 한
        클립보드가 알아서 메모리를 해제한다. }
    GlobalFree(Data);
    raise;
  end;
end;

procedure TData.GetFromClipBoard;
{ 이 메소드는 클립보드에 있는 데이터가 CF_DDGDATA이면 이 클래스의 TDataRec
  필드에 클립보드의 데이터를 넣는다. }
var
  Data: THandle;
```

```
  DataPtr: Pointer;
  Size: Integer;
begin
  // 클립보드의 핸들을 얻는다.
  Data := ClipBoard.GetAsHandle(CF_DDGDATA);
  if Data = 0 then Exit;
  // Data 핸들이 가리키는 메모리 블록을 가리키는 포인터를 얻는다.
  DataPtr := GlobalLock(Data);
  try
    // 가져올 데이터의 크기를 얻는다.
    if SizeOf(TDataRec) > GlobalSize(Data) then
      Size := GlobalSize(Data)
    else
      Size := SizeOf(TDataRec);
    // 데이터를 TDataRec에 복사한다.
    Move(DataPtr^, Rec, Size)
  finally
    // 메모리 블록에 대한 포인터를 해제한다.
    GlobalUnlock(Data);
  end;
end;

initialization
  // 클립보드 형식을 등록한다.
  CF_DDGDATA := RegisterClipBoardFormat(DDGData);
end.
```

이 유닛은 우선 Win32 API 함수 RegisterClipBoardFormat()을 호출해서 새 데이터 형식을 Win32 클립보드에 등록한다. 이 함수의 리턴값은 등록된 형식의 ID이다. 다른 애플리케이션도 같은 문자열을 이용해서 이 형식을 등록하면 같은 ID를 돌려받게 된다. 새 형식은 클립보드가 인식할 수 있는 형식들의 목록이 담겨 있는 Clipboard.Formats 속성에 추가된다.

이 유닛은 클립보드에 넣을 데이터를 담은 레코드와 그 레코드를 캡슐화하는 객체를 정의한다. TDataRec은 사람의 이름을 담을 문자열 필드와 나이를 뜻하는 Integer 필드, 그리고 생일을 뜻하는 TDatTime 필드로 구성된다.

TDataRec을 감싸는 TData 클래스에는 CopyToClipboard()와 GetFromClipboard()라는 메소드가 있다.

TData.CopytoClipboard()는 TData.Rec의 내용을 CF_DDGDATA와 CF_TEXT 두 가지 형식으로 클립보드에 복사한다. CF_TEXT는 텍스트를 뜻하는 미리 정의된 형식이다. TData.Rec의 내용을 텍스트로 넣을 때에는 각 필드들을 CR/LF, 즉 줄 바꿈으로 분리된 하나의 문자열로 만든 후 만들어진 문자열을 클립보드에 복사한다. 문자열이 아닌 필드들은 물론 적절히 문자열로 변환한다. 레코드, 즉 이진 데이터를 클립보드에 복사할 때에는 Clipboard.SetAsHandle()을 사용하는데, 이 메소드의 첫 번째 인자는 데이터의 형식을 뜻하는 THandle값으로, 이번 예제에서는 CF_DDGDATA이다.

THandle형인 CF_DDGDATA는 클립보드에 보낼 데이터가 있는 메모리 블록을 뜻하는 핸들이다. 다음의 "THandle 다루기" 박스 기사를 참고하기 바란다. 다음의 코드는 전역 힙에서 Sizeof(TDataRec) 바이트의 메모리를 할당하고, 그 메모리에 대한 핸들을 변수 Data에 넣는다.

```
Data := GlobalAlloc(GMEM_MOVEABLE, SizeOf(TDataRec));
```

그 메모리 블록에 대한 포인터는 다음의 문장을 통해서 얻는다.

```
DataPrt := GlobalLock(Data);
```

그런 다음, Move() 함수를 이용해서 레코드의 데이터를 메모리 블록으로 옮긴다. 그런 다음에는 ClipBoard.Open() 메소드로 클립보드를 열어서 다른 애플리케이션들이 끼여들지 못하게 한 다음, 두 형식의 데이터를 클립보드에 복사하고, 클립보드를 다시 닫는다.

```
ClipBoard.Open;
ClipBoard.SetAsHandle(CF_DDGDATA, Data);
with Rec do
  TempStr := FName+CRLF+LName+CRLF+MI+CRLF+IntToStr(Age)+CRLF+
                       DateTimeToStr(BirthDate);
ClipBoard.AsText := TempStr;
Clipboard.Close
```

일반적으로, 한 가지 형식의 데이터만을 클립보드에 복사할 때에는 위에서처럼 Open() 메소드를 호출할 필요가 없다. 왜냐하면, ClipBoard.SetTextBuf()라든지, ClipBoard.AsText 등을 사용해서 데이터를 복사할 때에는 그 메소드나 속성의 내부에서 Open()과 Close()를 호출하기 때문이다. ClipBoard.Open()을 먼저 호출하면 내부적으로 Open()이나 Close()가 호출되지 않기 때문에 한꺼번에 여러 형식의 데이터를 클립보드에 넣을 수 있다. 위에서 Open() 메소드를 호출하지 않는다면 이 메소드를 호출한 후에 CF_TEXT 형식의 데이터만 클립보드에 남아 있게 된다. 그리고 Open() 메소드를 호출했으면 반드시 Close()로 닫아야 한다.

여기까지 했으면 GlobalAlloc() 함수로 클립보드에 할당한 메모리는 Win32가 관리한다. 단, 복사 과정에서 예외가 발생할 수 있으므로 예외 처리에서 GlobalFree()를 호출해야 한다. 예외가 발생하지 않았는데도 GlobalFree()를 호출하면 Win32가 메모리를 제대로 해제할 수 없으므로 문제가 생긴다.

이 메소드를 호출하고 나면 클립보드에 CF_DDGDATA와 CF_TEXT 형식의 데이터가 들어 있는 상태가 된다. 클립보드에 있는 두 형식의 데이터를 사용하는 것에 대해서는 잠시 후에 예제 프로그램을 통해서 이야기하겠다.

TData.GetFromClipboard()는 TData.CopyToClipBoard()의 반대 역할을 한다.
즉, CF_DDGDATA 형식의 데이터를 클립보드에서 가져와서 TData.Rec 필드에 넣는 것이다. 코드에 대한 설명은 주석만으로도 충분할 것이다. 이 유닛은 잠시 후에 이야기할 예제 애플리케이션이 사용한다. 이 유닛을 통해서 개발자가 정의한 데이터 형식을 클립보드에 넣거나 가져오는 방법을 충분히 이해할 수 있을 것이다.



GetAsHandle()로 얻은 핸들을 해제하면 안된다. 그 핸들은 애플리케이션에 속한 것이 아니라, 클립보드에 속한 것이다. 따라서, 핸들이 참조하는 데이터는 반드시 다른 변수에 복사해서 사용해야 한다.

THandle 다루기

THandle은 Win32가 관리하는 메모리 블록들에 대한 정보가 담겨 있는 테이블의 색인을 뜻하는 32비트값이다. THandle에는 여러 가지 형들이 있으며, 델파이는 TIcon, TBitmap, TCanvas 등 여러 컴포넌트들의 속성에 THandle을 포함시키고 있다.
여러 클립보드 함수 등 Win32의 함수들 중에는 힙 메모리를 이용하는 것들이 있다. 힙 메모리에 접근하려면 다음과 같은 메모리 할당/해제 함수들을 사용해야 한다.

GlobalAlloc()	인자로 지정된 바이트만큼의 메모리를 힙에서 할당하고, 그 메모리 블록을 뜻하는 THandle을 돌려준다.
GlobalFree()	GlobalAlloc()으로 할당한 메모리를 해제한다.
GlobalLock()	GlobalAlloc()으로 할당한 메모리 블록을 가리키는 포인터를 돌려준다.
GlobalUnlock()	GlobalLock()으로 잠근 메모리를 푼다.

독자적인 클립보드 데이터 형식을 사용하는 예제 프로그램

독자적인 클립보드 데이터 형식을 사용하는 예제 프로그램의 메인 폼은 그림 17.1과 같다.

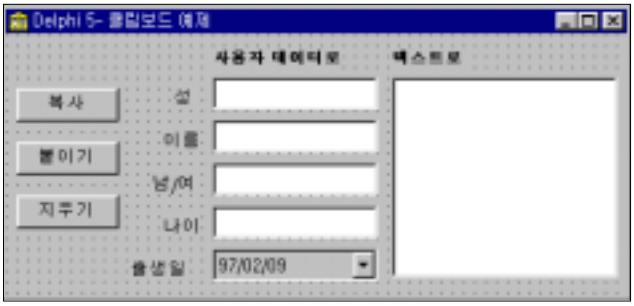


그림 17.1
독자적인 클립보드 데이터 형식을 사용하는 예제 프로그램의 메인 폼

폼에는 TDataRec의 필드들에 넣을 값을 입력하는 컨트롤들이 있다. 목록 17.4는 이번 예제의 소스 코드인데, 이번 예제는 부록 CD의 DdgCp.dpr에 해당한다.

목록 17.4 Ch17\Clipbrc\Main.pas

```
unit MainFrm;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, clipbrd, Mask, ComCtrls;
type

  TMainForm = class(TForm)
    edtFirstName: TEdit;
    edtLastName: TEdit;
    edtMI: TEdit;
    btnCopy: TButton;
    btnPaste: TButton;
    meAge: TMaskEdit;
    btnClear: TButton;
    lblFirstName: TLabel;
    lblLastName: TLabel;
    lblMI: TLabel;
    lblAge: TLabel;
    lblBirthDate: TLabel;
    memAsText: TMemo;
    lblCustom: TLabel;
    lblText: TLabel;
    dtpBirthDate: TDateTimePicker;
    procedure btnCopyClick(Sender: TObject);
    procedure btnPasteClick(Sender: TObject);
    procedure btnClearClick(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation
uses cbdata;
```

```
{$R *.DFM}

procedure TMainForm.btnCopyClick(Sender: TObject);
// 이 메소드는 폼의 컨트롤들에 담긴 데이터를 클립보드에 복사한다.
var
  DataObj: TData;
begin
  DataObj := TData.Create;
  try
    with DataObj.Rec do
    begin
      FName     := edtFirstName.Text;
      LName     := edtLastName.Text;
      MI        := edtMI.Text;
      Age       := StrToInt(meAge.Text);
      BirthDate := dtpBirthDate.Date;
      DataObj.CopyToClipBoard;
    end;
  finally
    DataObj.Free;
  end;
end;

procedure TMainForm.btnPasteClick(Sender: TObject);
{ 이 메소드는 클립보드에서 CF_DDGDATA 형식의 데이터를 가져와서
 폼의 컨트롤들에 넣는다. CF_TEXT 형식의 데이터는 폼의 TMemo
 컴포넌트에 넣는다. }
var
  DataObj: TData;
begin
  btnClearClick(nil);
  DataObj := TData.Create;
  try
    // CF_DDGDATA 형식의 데이터가 있는지 확인
    if ClipBoard.HasFormat(CF_DDGDATA) then
      // CF_DDGDATA 형식의 데이터를 폼의 컨트롤들에 넣는다.
      with DataObj.Rec do
      begin
        DataObj.GetFromClipBoard;
```

```
        edtFirstName.Text := FName;
        edtLastName.Text  := LName;
        edtMI.Text        := MI;
        meAge.Text        := IntToStr(Age);
        dtpBirthDate.Date := BirthDate;
      end;
  finally
    DataObj.Free;
  end;
  // 텍스트 형태의 데이터는 폼의 TMemo 컴포넌트에 넣는다.
  if ClipBoard.HasFormat(CF_TEXT) then
    memAsText.PasteFromClipBoard;
end;

procedure TMainForm.btnClearClick(Sender: TObject);
var
  i: integer;
begin
  // 폼의 모든 컨트롤들의 내용을 비운다.
  for i := 0 to ComponentCount - 1 do
    if Components[i] is TCustomEdit then
      TCustomEdit(Components[i]).Text := '';
end;

end.
```

사용자가 Copy 버튼을 클릭하면 프로그램은 메인 폼의 TEdit와 TMaskEdit 컨트롤들의 내용을 TData 객체의 TDataRec 필드에 넣은 다음, TData.CopyToClipboard 메소드를 호출해서 데이터를 클립보드에 복사한다.

사용자가 Paste 버튼을 클릭하면 그 반대의 작업이 수행된다. 우선, 클립보드의 데이터가 CF_DDGDATA인지 확인한 후 클립보드에서 데이터를 가져와서 폼의 에디트 컨트롤들에 채워넣는다. 그리고 텍스트 형식으로 된 데이터는 폼의 TMemo 컴포넌트에 넣는다. 그림 17.2는 붙여넣기를 수행한 후의 모습이다. 텍스트 형식의 데이터는 메모장 같은 다른 윈도우즈 애플리케이션들로 사용할 수 있다.



그림 17.2
클립보드에서 메인 폼으로 데이터를 붙여넣은 모습

Clear 버튼을 클릭하면 메인 폼의 모든 컨트롤들의 내용이 비워진다.

요약

애플리케이션들 사이에서 데이터를 공유할 수 있다는 것은 매우 편리한 기능이다. 애플리케이션이 클립보드를 통해서 다른 애플리케이션들과 데이터를 공유할 수 있다면 사용자는 좀더 효율적으로 애플리케이션을 사용할 수 있을 것이다. 이번 장에서는 델파이 컴포넌트에 구현되어 있는 클립보드 관련 기능을 이용하는 방법과 함께 독자적인 클립보드 데이터 형식을 정의하고 사용하는 방법을 배웠다. 애플리케이션들 사이에서 데이터를 공유하는 좀더 강력한 방법은 COM이다. COM에 대해서는 23장에서 이야기하겠다.

쉬어가는 페이지