16장 MDI 애플리케이션

* MDI 애플리케이션 만들기
* 메뉴 다루기
* 기타 MDI 기법들

MDI, 즉 다중 문서 인터페이스(Multiple Document Interface)는 윈도우즈 2.0 시절 마이크로소프트 엑셀이 도입한 것이다. 엑셀은 MDI 덕분에 여러 개의 스프레드시트를 동시에 다룰 수 있었다. 윈도우즈 3.1의 프로그램 관리자와 파일 관리자 역시 MDI를 채용했으며, 볼랜드의 윈도우즈용 파스칼(Pascal for Windows) 역시 MDI를 채용했다.

윈도우즈 95가 개발되는 동안 마이크로소프트가 MDI 기능을 폐기하는 것이 아닌가 하는 생각을 가진 개발자들이 많았다. 물론, 윈도우즈 95에는 MDI 기능이 포함되었으며, 마이크로소프트가 공식적으로 MDI의 폐기에 대해 이야기한 적은 없다.



마이크로소프트는 윈도우즈의 MDI 구현에 문제가 있다는 것을 발견했으며, 개발자들에게 MDI 모델로 애플리케이션을 만드는 것에 대해 부정적인 견해를 밝혔다. 마이크로소프트는 그 이후로도 자신들의 제품을 MDI 스타일로 만들긴 했지만, 윈도우즈의 MDI 구현을 사용한 것은 아니었다. 윈도우즈 MDI 구현에 문제가 있으며, 마이크로소프트는 그러한 문제를 고칠 계획이 없다는 점을 명심하기 바란다. 이번 장에 나오는 내용들은 MDI 모델 중 안전하게 구현된 부분에 대한 것들이다.

여러 개의 폼들 사이에서 발생하는 이벤트들을 병렬적으로 처리한다는 것은 쉬운 일이 아니다. 전통적인 윈도우즈 프로그래밍이라면 MDICLIENT라는 윈도우즈 클래스와 MDI 구조체들, 그리고 MDI에 관련된 여러 함수들과 메시지들에 대해 알고 있어야 한다. 그러나 델파이 5로서는 MDI 애플리케이션의 작성이 매우 간단해졌다. 이번 장에는 델파이 5의 기능을 최대한 발휘해서 기본적인 MDI 애플리케이션을 간단히 만들 수 있는 기법뿐만 아니라, 좀더 고난도의 기법들도 제시할 것이다.

MDI 애플리케이션 만들기

MDI 애플리케이션을 만들려면 우선 fsMDIForm과 fsMDIChild라는 폼 스타일과 몇 가지 MDI 프로그래밍 기법들에 친숙해져야 한다. 그럼, MDI에 관련된 기본적인 개념들과 함께 특정 MDI 자식 폼에 대해 MDI가 작동하는 방식을 설명하겠다.

MDI의 기본적인 개념들

MDI 애플리케이션을 이해하려면 우선 MDI 애플리케이션이 어떻게 구성되는가부터 알아야 한다. 그림 16.1은 이번 장에서 만들 예제 MDI 애플리케이션의 모습이다.

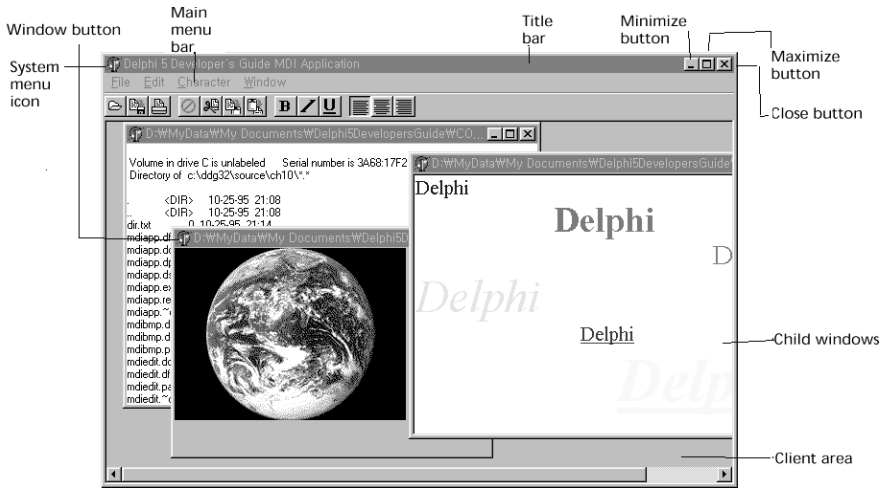


그림 16.1
MDI 응용 프로그램의 구조

MDI 애플리케이션의 구성 요소는 다음과 같다.맥치

- 프레임 윈도우 : 애플리케이션의 메인 윈도우이다. 이 윈도우에는 제목 표시줄과 메뉴줄, 시스템 메뉴가 포함되며, 좌측 상단에는 최소화, 최대화, 닫기 버튼이 나타난다. 프레임 윈도우 안의 빈 공간, 즉 클라이언트 윈도우 또는 클라이언트 영역이라고 한다.
- 클라이언트 윈도우 : MDI 애플리케이션의 관리자이다. 클라이언트 윈도우는 모든 MDI 관련 명령들을 처리하며, 자신의 표면 위에서 자식 윈도우들을 관리한다(MDI 윈도우를 그리는 작업까지). 프레임 윈도우를 만들면 VCL이 자동으로 클라이언트 윈도우를 생성한다.
- 자식 윈도우 : MDI 자식 윈도우는 텍스트 파일이나 스프레드시트, 비트맵 등등 애플리케이션이 다루는 실제 문서를 뜻하는 것이다. 자식 윈도우 역시 프레임 윈도우와 마찬가지로, 제목 표시줄과 시스템 메뉴, 최소화/최대화/닫기 버튼을 가지며, 메뉴를 가질 수도 있다. 또한, 자식 윈도우에 도움말 버튼이 있을 수도 있다. 자식 윈도우의 메뉴는 프레임 윈도우의 메뉴와 결합되기도 한다. 자식 윈도우는 클라이언트 영역 밖으로 이동되지 않는다.

델파이 5로 MDI 애플리케이션을 만들 때에는 MDI 윈도우에 관련된 메시지를 알 필요가 없다. 클라이언트 윈도우는 MDI 기능을 관리하는 역할을 한다(계단식 배열, 바둑판식 배열 등). 자식 윈도우들을 계단식으로 배열할 때(cascade) 전통적인 방식에서는 다음과 같이 SendMessage() API 함수로 WM_MDICASCADE 메시지를 클라이언트에게 보냈었다.

```
procedure TFrameForm.Cascade1Click(Sender: TObject);
begin
  SendMessage(ClientHandle, WM_MDICASCADE, 0, 0);
end;
```

델파이 5에서는 그냥 Cascade() 메소드를 호출하면 된다.

```
procedure TFrameForm.Cascade1Click(Sender: TObject);
begin
  cascade;
end;
```

이제부터는 텍스트 편집, 비트맵 파일 표시, RTF 편집기의 기능을 갖는 자식 MDI 윈도우들이 포함된, 완전한 형태의 MDI 애플리케이션 예제를 만드는 과정을 설명하겠다. 이 예제 애플리케이션은 각각 서로 다른 종류의 정보를 표시하는 자식 윈도우들로 구성된 MDI 애플리케이션을 작성하는 방법을 보여주기 위한 것이다. 예를 들어서, 텍스트 편집 자식 윈도우로는 텍스트 기반의 파일을 편집할 수 있으며, RTF(rich text format) 편집 자식 윈도우로는 RTF 문서를 편집할 수 있고, 비트맵 보기 자식 윈도우로는 윈도우즈 비트맵 파일을 표시할 수 있다.

또한, Win32 API를 이용한 몇 가지 고급 MDI 기법들도 제시할 것이다. 이번 애플리케이션은 MDI 자식 윈도우들을 좀더 효율적으로 다루기 위해서 Win32 API를 사용한다. 그럼, 자식 폼(윈도우)들의 작성부터 살펴보자. 자식 폼들을 작성한 다음에 메인 폼을 작성하겠다.

자식 폼

앞에서도 이야기했듯이 이번 MDI 애플리케이션은 세 종류의 자식 폼으로 구성된다. 각각 TMdiEditForm, TMdiRTFForm, TMdiBMPForm이다. 이 세 폼 클래스는 기반 클래스인 TMDIChildForm을 상속한다.

다음은 TMDIChildForm에 대한 설명이다. TMDIChildForm에 대해 설명한 다음 세 종류의 폼에 대해 설명하겠다.

기반 클래스 TMDIChildForm

MDI 애플리케이션의 자식 폼들은 몇 가지 공통적인 기능을 가진다. 자식 폼들은 모두 동일한 File 메뉴를 가지며, FormStyle 속성은 fsMDIChild이다. 또한, 세 폼 모두 TToolBar를 사용한다. 동일한 기능들을 기반 클래스로 만들어두면 같은 코드를 반복해서 작성하거나, 동일한 컴포넌트를 매번 설정할 필요가 없기 때문에 기반 폼 클래스인 TMDIChildForm을 만들어서 상속하도록 하는 것이다. 목록 16.1은 TMDIChildForm이 정의된 MdiChildFrm.pas의 소스 코드이다.


목록 16.1 Ch16\MdiApp\MdiChildFrm.pas

```
unit MdiChildFrm;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs,  Menus, ComCtrls, ToolWin, ImgList;

type

  TMDIChildForm = class(TForm)

     { 폼을 구성하는 컴포넌트들은 생략했음.
         부록 CD의 소스를 참고하기 바람. }

    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure mmiExitClick(Sender: TObject);
    procedure mmiCloseClick(Sender: TObject);
    procedure mmiOpenClick(Sender: TObject);
```

```
    procedure mmiNewClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormDeactivate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  MDIChildForm: TMDIChildForm;

implementation
uses MainFrm,  Printers;

{$R *.DFM}

procedure TMDIChildForm.FormClose(Sender: TObject; var Action:
 TCloseAction);
begin
  Action := caFree;
end;

procedure TMDIChildForm.mmiExitClick(Sender: TObject);
begin
  MainForm.Close;
end;

procedure TMDIChildForm.mmiCloseClick(Sender: TObject);
begin
  Close;
end;

procedure TMDIChildForm.mmiOpenClick(Sender: TObject);
begin
  MainForm.mmiOpenClick(nil);
end;

procedure TMDIChildForm.mmiNewClick(Sender: TObject);
begin
  MainForm.mmiNewClick(nil);
end;
```

```
procedure TMDIChildForm.FormActivate(Sender: TObject);
begin
  MainForm.SetToolBar(tlbMain);
end;

procedure TMDIChildForm.FormDeactivate(Sender: TObject);
begin
  tlbMain.Parent := nil;
end;

procedure TMDIChildForm.FormDestroy(Sender: TObject);
begin
  MainForm.SetToolBar(nil);
end;

end.
```



지면 관계상 목록 16.1에서 TMdiChildForm을 구성하는 컴포넌트들을 생략했다. 부록 CD의 소스를 참고하기 바란다.

TMDIChildForm에는 메인 메뉴의 메뉴 항목들에 대한 이벤트 처리 메소드들이 포함되어 있으며, 공통적인 툴바 버튼들에 대한 이벤트 처리 메소드들도 포함되어 있다. 툴바 버튼들의 이벤트 처리 메소드는 자신에게 해당하는 메뉴 항목의 이벤트 처리 메소드를 호출하는 역할만 한다. 그리고 몇몇 프로시저들 역시 메인 폼의 메소드를 호출하는 역할을 한다. 예를 들어서, mmiNewClick() 프로시저는 MainForm.mmiNewClick() 이벤트 처리 메소드를 호출한다. TMainForm.mmiNewClick()은 새 MDI 자식 폼을 생성하는 프로시저이다. mmiOpenClick()과 mmiExitClick() 역시 메인 폼의 해당 이벤트 처리 메소드를 호출한다. TMainForm의 기능에 대해서는 "메인 폼" 부분에서 설명하겠다.

각각의 MDI 자식 폼들은 공통적인 기능을 가지고 있으므로 기반 클래스에서 공통적인 기능을 정의한 후 개별적인 MDI 자식 폼들이 기반 클래스를 상속하도록 만드는 것이 합리적이다. 그렇게 하면 매번 동일한 컴포넌트나 동일한 메소드를 설정하거나 정의할 필요가 없다. 각각의 자식 폼들은 이 기반 폼 클래스의 메인 메뉴와 툴바들을 상속받는다.

TMDIChildForm.FormClose() 이벤트 처리 메소드의 코드를 보면 Action 인자를 caFree로 설정하고 있음을 알 수 있다. 이렇게 하면 TMDIChildForm의 인스턴스를 닫을 때 인스턴스가 실제로 해제된다. Close() 메소드는 Action 인자의 값에 따라 서로 다른 처리를 한다. 즉, OnClose 이벤트가 발생했다고 해도 반드시 폼이 종료되거나 파괴되는 것은 아니며, Action 인자에 원하는 값을 넣어서 종료 처리 방식을 직접 지정할 수 있다. Action 인자는 TCloseAction형으로, 사용할 수 있는 값은 다음과 같다.

caNone	아무 일도 하지 않는다.
caHide	폼을 숨기되, 해제하지는 않는다.
caFree	폼을 해제한다.
caMinimize	폼을 최소화한다(기본값).

TCloseAction은 열거형이다.

폼이 활성화되면 OnActive 이벤트 처리 메소드가 호출된다. 여기서 자식 폼의 활성화 작업을 하면 된다.

TMdiChildForm.FormActivate() 이벤트 처리 메소드를 보면 MainForm.SetToolBar() 메소드를 호출하고 있다. MainForm.SetToolBar() 메소드는 자식 폼의 툴바의 부모를 메인 폼으로 설정해서 자식 폼의 툴바가 메인 폼 위에 나타나도록 하는 역할을 한다. 이렇게 하면 서로 다른 종류의 MDI 자식 폼이 활성화될 때마다 메인 폼의 툴바가 그에 맞는 툴바로 변하게 된다.

텍스트 편집 자식 폼

텍스트 편집 자식 폼은 텍스트 파일을 적재하고 편집할 수 있는 기능을 가지고 있다. 텍스트 편집 자식 폼의 클래스는 TMdiEditForm으로, TMDIChildForm을 상속한 클래스이다. TMdiEditForm에는 자신의 클라이언트 영역에 꽉차는 TMemo 컴포넌트가 있다.

그리고 TMdiEditForm은 TSaveDialog와 TFontDialog 컴포넌트도 포함하고 있다.

TMdiEditForm은 애플리케이션이 실행될 때 자동으로 생성되면 안되기 때문에 Project Options 대화상자의 Auto-create forms 목록에서 제거해야 한다.



이번 예제 애플리케이션에서는 메인 폼 TMainForm을 제외한 다른 모든 폼들이 자동 생성 폼 목록에서 제거되어 있다. 그 폼들은 모두 프로젝트의 소스 코드를 통해서 동적으로 생성/파괴된다.

TMdiEditForm의 기능

TMdiEditForm의 소스 코드는 목록 16.2와 같다.



목록 16.2 Ch16\MdiApp\MdiEditFrm.pas의 소스 코드

```
unit MdiEditFrm;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Menus, ExtCtrls, Buttons, ComCtrls,
  ToolWin, MdiChildFrm, ImgList;

type

  TMdiEditForm = class(TMDIChildForm)

     { 폼을 구성하는 컴포넌트들은 생략했음.
         부록 CD의 소스를 참고하기 바람. }

    { File 메뉴의 이벤트 처리 메소드들 }
    procedure mmiSaveClick(Sender: TObject);
```

```
    procedure mmiSaveAsClick(Sender: TObject);

    { Edit 메뉴의 이벤트 처리 메소드들 }
    procedure mmiCutClick(Sender: TObject);
    procedure mmiCopyClick(Sender: TObject);
    procedure mmiPasteClick(Sender: TObject);
    procedure mmiDeleteClick(Sender: TObject);
    procedure mmiUndoClick(Sender: TObject);
    procedure mmiSelectAllClick(Sender: TObject);

    { Character 메뉴의 이벤트 처리 메소드들 }
    procedure CharAlignClick(Sender: TObject);
    procedure mmiBoldClick(Sender: TObject);
    procedure mmiItalicClick(Sender: TObject);
    procedure mmiUnderlineClick(Sender: TObject);
    procedure mmiWordWrapClick(Sender: TObject);
    procedure mmiFontClick(Sender: TObject);

    { 폼의 이벤트 처리 메소드들 }
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure mmiPrintClick(Sender: TObject);
  public
    { 사용자 정의 메소드들 }
    procedure OpenFile(FileName: String);
    procedure SetButtons;
  end;

var
  MdiEditForm: TMdiEditForm;

implementation

uses Printers;

{$R *.DFM}
{ File 메뉴의 이벤트 처리 메소드들 }

procedure TMdiEditForm.mmiSaveClick(Sender: TObject);
begin
```

```
  inherited;
  { 폼의 Caption 속성이 비어 있다는 것은 지정된 파일 이름이 없다는 뜻이다.
     그러므로 mmiSaveAsClick을 호출해서 파일 이름을 선택받는다. }
  if Caption = '' then
    mmiSaveAsClick(nil)
  else begin
    { 폼의 Caption 속성에 지정된 파일 이름으로 텍스트를 저장한다. }
    memMainMemo.Lines.SaveToFile(Caption);
    memMainMemo.Modified := false; // 텍스트가 저장되었으므로 False로 설정
  end;
end;

procedure TMdiEditForm.mmiSaveAsClick(Sender: TObject);
begin
  inherited;
  SaveDialog.FileName := Caption;
  if SaveDialog.Execute then
  begin
    { SaveDialog로 지정된 파일 이름을 폼의 Caption 속성에 넣는다. }
    Caption := SaveDialog.FileName;
    mmiSaveClick(nil); // 텍스트를 저장
  end;
end;

{ Edit 메뉴의 이벤트 처리 메소드들 }

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;

procedure TMdiEditForm.mmiDeleteClick(Sender: TObject);
begin
  inherited;
  memMainMemo.ClearSelection;
end;

procedure TMdiEditForm.mmiUndoClick(Sender: TObject);
begin
  inherited;
  memMainMemo.Perform(EM_UNDO, 0, 0);
end;

procedure TMdiEditForm.mmiSelectAllClick(Sender: TObject);
begin
  inherited;
  memMainMemo.SelectAll;
end;

{ Character 메뉴의 이벤트 처리 메소드들 }
procedure TMdiEditForm.CharAlignClick(Sender: TObject);
begin
  inherited;
  mmiLeft.Checked := false;
  mmiRight.Checked := false;
  mmiCenter.Checked := false;

{ VCL에서 TAlignment는 다음과 같이 정의되어 있다.

  TAlignment = (taLeftJustify, taRightJustify, taCenter);

 각 메뉴 항목의 Tag 속성에는 자신에 해당하는 TAlignment의 값(0, 1, 2)이
 들어 있으므로 Tag 속성의 값을 직접 TMemo의 Alignment 속성에 넣으면 된다. }

 { 사용자가 메뉴 항목들 중 하나를 선택하면 그 항목의 Checked 속성을 True로 하고,
```

```
    Memo 컨트롤의 정렬 방식을 그 항목에 맞게 설정한다. }
  if Sender is TMenuItem then
  begin
    TMenuItem(Sender).Checked := true;
    memMainMemo.Alignment := TAlignment(TMenuItem(Sender).Tag);
  end
 { 사용자가 메인 폼의 TToolButton들 중 하나를 클릭해서 이 메소드가 실행되는
    것이라면 TMemo의 정렬 방식을 버튼에 맞게 설정하고, 버튼에 해당하는 메뉴
    항목의 Checeked를 True로 한다. }
  else if Sender is TToolButton then
  begin
    memMainMemo.Alignment := TAlignment(TToolButton(Sender).Tag);
    case memMainMemo.Alignment of
      taLeftJustify:   mmiLeft.Checked := True;
      taRightJustify:  mmiRight.Checked := True;
      taCenter:        mmiCenter.Checked := True;
    end;
  end;
  SetButtons;
end;

procedure TMdiEditForm.mmiBoldClick(Sender: TObject);
begin
  inherited;
  if not mmiBold.Checked then
    memMainMemo.Font.Style := memMainMemo.Font.Style + [fsBold]
  else
    memMainMemo.Font.Style := memMainMemo.Font.Style - [fsBold];
  SetButtons;
end;

procedure TMdiEditForm.mmiItalicClick(Sender: TObject);
begin
  inherited;
  if not mmiItalic.Checked then
    memMainMemo.Font.Style := memMainMemo.Font.Style + [fsItalic]
  else
    memMainMemo.Font.Style := memMainMemo.Font.Style - [fsItalic];
  SetButtons;
```

```
end;

procedure TMdiEditForm.mmiUnderlineClick(Sender: TObject);
begin
  inherited;
  if not mmiUnderline.Checked then
    memMainMemo.Font.Style := memMainMemo.Font.Style + [fsUnderline]
  else
    memMainMemo.Font.Style := memMainMemo.Font.Style - [fsUnderline];
  SetButtons;
end;

procedure TMdiEditForm.mmiWordWrapClick(Sender: TObject);
begin
  inherited;
  with memMainMemo do
  begin
    WordWrap := not WordWrap;
     { TMemo가 자동 줄 바꿈(wordwrap) 상태이면 스크롤바가 필요 없으므로
         가로 스크롤바를 없앤다. 아니면, 가로/세로 스크롤바를 모두 표시한다. }
    if WordWrap then
      ScrollBars := ssVertical
   else
      ScrollBars := ssBoth;
    mmiWordWrap.Checked := WordWrap;
  end;
end;

procedure TMdiEditForm.mmiFontClick(Sender: TObject);
begin
  inherited;
  FontDialog.Font := memMainMemo.Font;
  if FontDialog.Execute then
    memMainMemo.Font := FontDialog.Font;
end;

{ 폼의 이벤트 처리 메소드들 }
procedure TMdiEditForm.FormCloseQuery(Sender: TObject;
     var CanClose: Boolean);
```

```
{ 사용자가 메모 컴포넌트의 내용을 고친 후 저장하지 않고 폼을 종료하려고 하면
 사용자에게 변경한 내용을 저장하겠느냐고 묻는다. }
const
  CloseMsg = '''%s'' has been modified, Save?';
var
  MsgVal: integer;
  FileName: string;
begin
  inherited;
  FileName := Caption;
  if memMainMemo.Modified then
  begin
    MsgVal := MessageDlg(Format(CloseMsg, [FileName]), mtConfirmation,
        mbYesNoCancel, 0);
    case MsgVal of
      mrYes:    mmiSaveClick(Self);
      mrCancel: CanClose := false;
    end;
  end;
end;

procedure TMdiEditForm.OpenFile(FileName: string);
begin
  memMainMemo.Lines.LoadFromFile(FileName);
  Caption := FileName;
end;

procedure TMdiEditForm.SetButtons;
{ 메모 컴포넌트의 현재 상태에 따라 메인 메뉴 항목들이나
 툴바 버튼들의 속성을 설정한다. }
begin
  mmiBold.Checked := fsBold in memMainMemo.Font.Style;
  mmiItalic.Checked := fsItalic in memMainMemo.Font.Style;
  mmiUnderLine.Checked := fsUnderline in memMainMemo.Font.Style;

  tbBold.Down      := mmiBold.Checked;
  tbItalic.Down    := mmiItalic.Checked;
  tbUnderline.Down := mmiUnderLine.Checked;
  tbLAlign.Down    := mmiLeft.Checked;
```

```
    tbRAlign.Down    := mmiRight.Checked;
    tbCAlign.Down    := mmiCenter.Checked;
  end;

  end.
```

TMdiEditForm의 대부분의 메소드들은 TMdiChildForm으로부터 상속받은 메뉴들의 클릭 이벤트 처리 메소드들이다. 물론, TMdiEditForm만의 메뉴들에 대한 이벤트 처리 메소드들도 있다.

그런데 이 폼에는 File | New와 File | Open, File | Close, 그리고 File | Exit 메뉴에 대한 이벤트 처리 메소드가 없다. 이 메뉴 항목들의 처리는 TMdiChildForm에 이미 구현되어 있다.

Edit 메뉴 항목들의 이벤트 처리 메소드는 모두 TMemo의 메소드를 호출하는 한 줄의 코드로 이루어져 있다. 예를 들어서, mmiCutClick()과 mmiCopyClick, mmiPasteClick()은 윈도우즈 클립보드를 통해서 내용을 잘라내거나, 복사하거나, 붙이는 작업을 수행한다. 다른 Edit 메뉴 항목들의 이벤트 처리 메소드들은 메모 컴포넌트의 내용을 삭제하거나, 비우거나, 일정 부분을 선택하는 작업을 한다.

Character 메뉴는 메모 컴포넌트의 폰트나 정렬에 관련된 것이다.

TToolButton 컴포넌트들의 Tag 속성에는 텍스트 정렬(왼쪽, 가운데, 오른쪽 등)에 관련된 값을 담고 있다. Tag 속성들을 TAlignment 열거형의 값들에 일치하게 설정했기 때문에 메뉴나 버튼을 클릭했을 때 별다른 조건문 없이 Tag 속성을 그냥 메모 컴포넌트의 Alignment 속성에 넣기만 하면 클릭한 버튼이나 메뉴에 해당하는 설정대로 메모 컴포넌트의 텍스트들이 정렬된다.

텍스트 정렬에 관련된 모든 메뉴 항목과 툴바 버튼들의 클릭 이벤트는 모두 CharAlignClick() 프로시저 하나로 연결된다. 따라서, CharAlignClick()에는 자신을 메뉴 항목이 호출했는지, 아니면 툴바 버튼이 호출했는지 판단하는 코드가 들어 있다.

CharAlignClick()은 메모 컴포넌트의 속성들에 따라 여러 가지 메뉴 항목들과 툴바 버튼들을 적절히 설정하는 SetButtons() 메소드를 호출한다.

mmiWordWrapClick() 이벤트 처리 메소드는 mmiWordWrap 메뉴 항목의 Checked 속성을 전환하고, 그 속성에 따라 메모 컴포넌트의 자동 줄 바꿈(Word warp)을 활성화/비활성화시킨다. 그리고 자동 줄 바꿈이 활성화되는 경우에는 가로 스크롤바를 제거한다.

mmiFontClick() 이벤트 처리 메소드는 폰트 대화상자, 즉 TFontDialog를 띄워서 폰트를 선택받은 다음, 그에 따라 메모 컴포넌트의 폰트를 변경한다. 폰트 대화상자를 띄우기 전에 메모 컴포넌트의 현재 폰트를 Font 속성에 넣어서 폰트 대화상자에 메모 컴포넌트의 현재 폰트가 나타나도록 하는 부분도 주목하기 바란다.

mmiSaveAsClick() 이벤트 처리 메소드는 저장 대화상자, 즉 TSaveDialog를 띄워서 파일 이름을 선택받은 다음, 그 이름의 파일에 메모 컴포넌트의 내용을 저장한다. 파일을 저장한 다음에는 폼의 제목 표시줄에 파일 이름이 나타나도록 TMdiEditForm.Caption 속성을 갱신한다.

mmiSaveClick() 이벤트 처리 메소드는 TMidEditForm.Caption 속성에 파일 이름이 존재하지 않으면, 즉 처음으로 파일을 저장하는 것이면 mmiSaveAsClick()을 호출한다. 파일 이름이 존재하면 그 파일 이름을 사용해서 메모 컴포넌트의 내용을 저장한다. 저장을 한 후에는 메모 컴포넌트 memMainMemo의 Modified 속성을 False로 한다. 사용자가 TMemo 컴포넌트의 내용을 변경하면 Modified 속성은 자동으로 True가 된다. 그러나 내용을 저장한다고 해서 자동으로 False가 되는 것은 아니기 때문에 직접 False로 설정하는 것이다.

FormCloseQuery() 메소드는 OnCloseQuery 이벤트가 발생하면 실행된다. 이 메소드는 memMainMemo.Modified 속성을 검사해서 그 속성이 True이면 사용자가 아직 메모 컴포넌트의 내용을 저장하지 않은 것이므로 사용자에게 파일을 저장할 기회를 준다.

공용 메소드인 TMdiEditForm.OpenFile()은 FileName 인자로 지정된 파일을 memMainMemo.Lines에 로드하고, 폼의 Caption을 FileName으로 설정한다.

TMdiEditForm의 기능은 이 정도이다. 다른 폼들도 비슷한 기능을 가지고 있다.

RTF 편집 폼

RTF 편집 폼은 RTF 파일을 로드하고 편집할 수 있는 폼이다. 이 폼의 클래스는 TMdiRtfForm이며, 역시 TMDIChildForm을 상속한다. 이 폼에는 클라이언트 영역에 꽉차는 TRichEdit 컴포넌트가 있다.

TMdiRtfForm은 TRichEdit 컴포넌트가 있다는 점을 제외하고는 TMdiEditForm과 거의 같다. TMdiRtfForm은 TRichEdit 컴포넌트가 있기 때문에 텍스트의 일정 부분만 편집 특성을 다르게 적용시킬 수 있다. TMemo의 경우에는 폰트나 텍스트 정렬 같은 편집 특성이 전체 텍스트에 적용된다.

목록 16.3은 TMdiRtfForm을 정의하는 소스 코드이다.

목록 16.3 Ch16\MdiApp\MdiRtfFrm.pas

```
unit MdiRtfFrm;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs,  MdiChildFrm, StdCtrls, ComCtrls,
  ExtCtrls, Buttons, Menus, ToolWin, ImgList;

type

  TMdiRtfForm = class(TMDIChildForm)

     { 폼을 구성하는 컴포넌트들은 생략했음.
         부록 CD의 소스를 참고하기 바람. }

    { File 메뉴의 이벤트 처리 메소드들 }
    procedure mmiSaveClick(Sender: TObject);
    procedure mmiSaveAsClick(Sender: TObject);

    { Edit 메뉴의 이벤트 처리 메소드들 }
    procedure mmiCutClick(Sender: TObject);
    procedure mmiCopyClick(Sender: TObject);
    procedure mmiPasteClick(Sender: TObject);
    procedure mmiDeleteClick(Sender: TObject);
    procedure mmiUndoClick(Sender: TObject);
    procedure mmiSelectAllClick(Sender: TObject);

    { Character 메뉴의 이벤트 처리 메소드들 }
    procedure CharAlignClick(Sender: TObject);
    procedure mmiBoldClick(Sender: TObject);
    procedure mmiItalicClick(Sender: TObject);
    procedure mmiUnderlineClick(Sender: TObject);
    procedure mmiWordWrapClick(Sender: TObject);
```

```
    procedure mmiFontClick(Sender: TObject);

    { 폼의 이벤트 처리 메소드들 }
    procedure FormShow(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure reMainSelectionChange(Sender: TObject);
  public
    { 사용자 정의 메소드들 }
    procedure OpenFile(FileName: String);
    function GetCurrentText: TTextAttributes;
    procedure SetButtons;
  end;

var
  MdiRtfForm: TMdiRtfForm;

implementation
{$R *.DFM}
{ File 메뉴의 이벤트 처리 메소드들 }

procedure TMdiRtfForm.mmiSaveClick(Sender: TObject);
begin
  inherited;
  reMain.Lines.SaveToFile(Caption);
end;

procedure TMdiRtfForm.mmiSaveAsClick(Sender: TObject);
begin
  inherited;
  SaveDialog.FileName := Caption;
  if SaveDialog.Execute then
  begin
    Caption := SaveDialog.FileName;
    mmiSaveClick(Sender);
  end;
end;

{ Edit 메뉴의 이벤트 처리 메소드들 }
```

```
procedure TMdiRtfForm.mmiCutClick(Sender: TObject);
begin
  inherited;
  reMain.CutToClipBoard;
end;

procedure TMdiRtfForm.mmiCopyClick(Sender: TObject);
begin
  inherited;
  reMain.CopyToClipBoard;
end;

procedure TMdiRtfForm.mmiPasteClick(Sender: TObject);
begin
  inherited;
  reMain.PasteFromClipBoard;
end;

procedure TMdiRtfForm.mmiDeleteClick(Sender: TObject);
begin
  inherited;
  reMain.ClearSelection;
end;

procedure TMdiRtfForm.mmiUndoClick(Sender: TObject);
begin
  inherited;
  reMain.Perform(EM_UNDO, 0, 0);
end;

procedure TMdiRtfForm.mmiSelectAllClick(Sender: TObject);
begin
  inherited;
  reMain.SelectAll;
end;

{ Character 메뉴의 이벤트 처리 메소드들 }

procedure TMdiRtfForm.CharAlignClick(Sender: TObject);
```

```
begin
  inherited;
  mmiLeft.Checked := false;
  mmiRight.Checked := false;
  mmiCenter.Checked := false;

 { 사용자가 메뉴 항목을 클릭해서 이 프로시저가 실행된 것이면 그 메뉴 항목의
     Checked 속성을 True로 하고, TRichEdit의 현재 문단의 정렬 속성을
     적절히 설정한다. }
  if Sender is TMenuItem then
  begin
    TMenuItem(Sender).Checked := true;
    with reMain.Paragraph do
      if mmiLeft.Checked then
        Alignment := taLeftJustify
      else if mmiRight.Checked then
        Alignment := taRightJustify
      else if mmiCenter.Checked then
        Alignment := taCenter;
  end
 { 메인 폼의 툴바 버튼을 클릭해서 이 프로시저가 실행된 것이면 TRichEdit의
     현재 문단의 정렬 속성을 적절히 설정하고, 툴바 버튼에 해당하는 메뉴
     항목의 Checked 속성을 True로 한다. }
  else if Sender is TSpeedButton then
  begin
    reMain.Paragraph.Alignment :=
       TAlignment(TSpeedButton(Sender).Tag);
    case reMain.Paragraph.Alignment of
      taLeftJustify: mmiLeft.Checked := True;
      taRightJustify: mmiRight.Checked := True;
      taCenter: mmiCenter.Checked := True;
    end;
  end;
  SetButtons;
end;

procedure TMdiRtfForm.mmiBoldClick(Sender: TObject);
begin
  inherited;
```

```
  if not mmiBold.Checked then
    GetCurrentText.Style := GetCurrentText.Style + [fsBold]
  else
    GetCurrentText.Style := GetCurrentText.Style - [fsBold];
end;

procedure TMdiRtfForm.mmiItalicClick(Sender: TObject);
begin
  inherited;
  if not mmiItalic.Checked then
    GetCurrentText.Style := GetCurrentText.Style + [fsItalic]
  else
    GetCurrentText.Style := GetCurrentText.Style - [fsItalic];
end;

procedure TMdiRtfForm.mmiUnderlineClick(Sender: TObject);
begin
  inherited;
  if not mmiUnderline.Checked then
    GetCurrentText.Style := GetCurrentText.Style + [fsUnderline]
  else
    GetCurrentText.Style := GetCurrentText.Style - [fsUnderline];
end;

procedure TMdiRtfForm.mmiWordWrapClick(Sender: TObject);
begin
  inherited;
  with reMain do
  begin
     { TRichEdit가 자동 줄 바꿈(wordwrap) 상태이면 스크롤바가 필요 없으므로
         가로 스크롤바를 없앤다. 아니면, 가로/세로 스크롤바를 모두 표시한다. }
    if WordWrap then
      ScrollBars := ssVertical
    else
      ScrollBars := ssNone;
    mmiWordWrap.Checked := WordWrap;
  end;
end;
```

```
procedure TMdiRtfForm.mmiFontClick(Sender: TObject);
begin
  inherited;
  FontDialog.Font.Assign(reMain.SelAttributes);
  if FontDialog.Execute then
    GetCurrentText.Assign(FontDialog.Font);
  reMain.SetFocus;
end;

{ 폼의 이벤트 처리 메소드들 }

procedure TMdiRtfForm.FormShow(Sender: TObject);
begin
  inherited;
  reMainSelectionChange(nil);
end;


procedure TMdiRtfForm.FormCloseQuery(Sender: TObject;
   var CanClose: Boolean);
{ 사용자가 TRichEdit의 내용을 고친 후 저장하지 않고 폼을 종료하려고 하면
 사용자에게 변경한 내용을 저장하겠느냐고 묻는다. }
const
  CloseMsg = '''%s'' has been modified, Save?';
var
  MsgVal: integer;
  FileName: string;
begin
  inherited;
  FileName := Caption;
  if reMain.Modified then
  begin
    MsgVal := MessageDlg(Format(CloseMsg, [FileName]), mtConfirmation,
mbYesNoCancel, 0);
    case MsgVal of
      mrYes: mmiSaveClick(Self);
      mrCancel: CanClose := false;
    end;
  end;
```

```
end;

procedure TMdiRtfForm.reMainSelectionChange(Sender: TObject);
begin
  inherited;
  SetButtons;
end;

procedure TMdiRtfForm.OpenFile(FileName: String);
begin
  reMain.Lines.LoadFromFile(FileName);
  Caption := FileName;
end;

function TMdiRtfForm.GetCurrentText: TTextAttributes;
{ 이 함수는 TRichEdit의 현재 문단 또는 선택된 텍스트의 텍스트 특성들을
 돌려준다. }
begin
  if reMain.SelLength > 0 then
    Result := reMain.SelAttributes
  else
    Result := reMain.DefAttributes;
end;

procedure TMdiRtfForm.SetButtons;
{ TRichEdit 컴포넌트의 현재 상태에 따라 메인 폼의 메뉴 항목들과
 버튼들의 속성을 설정한다. }
begin

  with reMain.Paragraph do
  begin
    mmiLeft.Checked := Alignment = taLeftJustify;
    mmiRight.Checked := Alignment = taRightJustify;
    mmiCenter.Checked := Alignment = taCenter;
  end;

  with reMain.SelAttributes do
  begin
    mmiBold.Checked := fsBold in Style;
```

```
    mmiItalic.Checked := fsItalic in Style;
    mmiUnderline.Checked := fsUnderline in Style;
  end;
  mmiWordWrap.Checked := reMain.WordWrap;

  tbBold.Down      := mmiBold.Checked;
  tbItalic.Down    := mmiItalic.Checked;
  tbUnderline.Down := mmiUnderline.Checked;
  tbLAlign.Down    := mmiLeft.Checked;
  tbRAlign.Down    := mmiRight.Checked;
  tbCAlign.Down    := mmiCenter.Checked;
end;

procedure TMdiRtfForm.mmiPrintClick(Sender: TObject);
begin
  inherited;
  reMain.Print(Caption);
end;

end.
```

TMdiEditForm과 마찬가지로, TMdiRtfForm의 대부분의 메소드들은 여러 가지 메뉴 항목들이나 툴바 버튼들의 이벤트 처리 메소드이다. 대부분의 메소드들이 하는 일도 TMdiEditForm의 것들과 비슷하다.

TMdiRtfForm의 File 메뉴 항목들은 기반 클래스 TMdiChilForm에 구현된 메소드들이 처리한다. TMdiRtfForm은 TMdiChilForm을 상속한 것임을 기억하기 바란다.

그리고 mmiSaveClick()과 mmiSaveAsClick()은 reMain.Lines.SaveToFile() 메소드를 이용해서 reMain(TRichEdit 컴포넌트)의 내용을 파일에 저장한다.

TMdiRtfForm의 Edit 메뉴 항목들의 이벤트 처리 메소드들은 TMdiEditForm의 것들과 거의 같다. TMemo가 아니라, TRichEdit의 메소드들을 호출한다는 점만 다르다. 메소드들의 이름도 모두 같지만, memMain이 아니라 reMain의 메소드들이라는 점만 다른 것이다.

TMdiRtfForm의 Character 메뉴 항목들은 TRichEidt 컴포넌트의 현재 문단이나 선택된 텍스트의 특성을 변경하는 역할을 한다(TMemo 컴포넌트에서는 텍스트의 특성이 전체 텍스트에 적용된다는 점을 기억하기 바란다). 변경할 특성을 문단에 적용하는 것인지, 아니면 선택된 텍스트에 적용하는 것인지는 GetCurrentText() 함수의 리턴값으로 판단한다. GetCurrentText()는 TRichEdit.SelLength값을 이용해서 선택된 텍스트가 있는지 판단한다. SelLength가 0이면 선택된 텍스트가 없는 것이다. TRichEdit.SelAttributes 속성은 TRichEdit 컴포넌트에서 선택된 텍스트의 텍스트 특성을 담고 있으며, DefAttributes 속성은 TRichEdit 컴포넌트의 현재 문단의 텍스트 특성을 담고 있다.

mmiFontClick() 이벤트 처리 메소드는 사용자에게 폰트를 선택받은 다음, 현재 문단 또는 선택된 텍스트의 폰트를 변경한다.

자동 줄 바꿈은 TMdiEditForm에서와 같은 방식으로 처리한다.

TRichEdit에서 선택된 부분이 변경되면 OnSelectionChange 이벤트가 발생한다. 사용자가 TRichEdit 컴포넌트 안에서 캐럿(텍스트 커서)을 이동시키면 그에 따라 TRichEdit의 SelStart 속성값이 변하며, 그러면 OnSelectionChange 이벤트가 발생한다. TRichEdit의 OnSelectionChange 이벤트 프로시저에서는 현재 선택된 부분 또는 캐럿이 가리키고 있는 부분의 텍스트 특성에 맞게 메뉴 항목들이나 버튼들의 상태를 갱신한다. TRichEdit에서는 텍스트의 각 부분마다 서로 다른 폰트나 특성을 부여할 수 있으므로 이렇게 캐럿의 위치에 따라 동적으로 메인 메뉴나 버튼들의 상태를 갱신해 줘야 한다. TMemo에서는 전체 텍스트가 동일한 특성을 가지므로 그럴 필요가 없었다.

기능상의 면에서는 RTF 편집 폼이나, 텍스트 편집 폼이나 큰 차이가 없다. 주된 차이는, RTF에서는 텍스트의 개별 문단이나 선택된 텍스트의 특성을 자유롭게 변경할 수 있다는 점이다. 메모 컴포넌트를 사용하는 텍스트 편집 폼에서는 그렇게 할 수 없다.

세 번째 MDI 자식 폼인 비트맵 표시 폼

비트맵 표시 폼의 기능은 윈도우즈 비트맵 파일을 로드하고 표시하는 것이다. 다른 두 MDI 자식 폼과 마찬가지로, 비트맵 표시 폼인 TMdiBmpForm 역시 기반 클래스 TMDIChildForm을 상속한 것이다. 이 폼에는 클라이언트 영역에 꽉차는 TImage 컴포넌트가 있다.

TMdiBmpForm은 편집 기능이 없으므로 메인 메뉴는 TMDIChildForm으로부터 상속받은 것들뿐이다. 목록 16.4는 TMdiBmpForm을 정의하는 소스 코드이다.

목록 16.4 TMdiBmpForm을 정의하는 MdiBmpFrm.pas의 소스 코드

```
unit MdiBmpFrm;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms,
 Dialogs,  MdiChildFrm, ExtCtrls, Menus, Buttons, ComCtrls, ToolWin,
 ImgList;

type
  TMdiBMPForm = class(TMDIChildForm)
    mmiEdit: TMenuItem;
    mmiCopy: TMenuItem;
    mmiPaste: TMenuItem;
    imgMain: TImage;
    procedure mmiCopyClick(Sender: TObject);
    procedure mmiPasteClick(Sender: TObject);
    procedure mmiPrintClick(Sender: TObject);
  public
    procedure OpenFile(FileName: string);
  end;

var
  MdiBMPForm: TMdiBMPForm;

implementation

uses ClipBrd, Printers;

{$R *.DFM}

procedure TMdiBMPForm.OpenFile(FileName: String);
begin
  imgMain.Picture.LoadFromFile(FileName);
  Caption := FileName;
end;
```

```
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;

procedure TMdiBMPForm.mmiPrintClick(Sender: TObject);
begin
  inherited;

  with ImgMain.Picture.Bitmap do
  begin
      Printer.BeginDoc;
      Printer.Canvas.StretchDraw(Canvas.ClipRect, imgMain.Picture.Bitmap);
      Printer.EndDoc;
  end; { with }
end;

 end.
```

TMdiBmpForm은 다른 폼에 비해 코드가 별로 없다. 이 폼의 File 메뉴 항목들 역시 TMdichildForm의 이벤트 처리 메소드들을 상속받는다. Edit 메뉴 항목들은 윈도우즈 클립보드에 비트맵을 복사하거나, 클립보드로부터 비트맵을 폼에 붙여넣는 역할을 한다. TImage.Picture.Assign() 메소드로 클립보드의 데이터를 TImage 컴포넌트에 복사할 때 TImage 컴포넌트는 CF_BITMAP과 CF_PICTURE 포맷을 비트맵으로 인식한다.

윈도우즈 클립보드

두 애플리케이션이 정보를 교환할 때 가장 간단한 수단이 바로 윈도우즈의 클립보드이다. 클립보드란, 사실 윈도우즈의 어떤 애플리케이션이라도 접근할 수 있는 전역 메모리 블록이다.
클립보드는 텍스트, OEM 텍스트, 비트맵, 메타 파일 같은 표준 형식의 데이터를 지원하며, 기타 특별한 용도의 형식들도 지원한다. 그리고 애플리케이션이 사용하는 특정 형식을 지원하도록 클립보드의 기능을 확장할 수도 있다.
델파이 4는 윈도우즈 클립보드의 기능을 TClipBoard형의 Clibboard라는 전역 변수로 캡슐화하기 때문에 API를 사용하지 않고도 쉽게 클립보드를 사용할 수 있다. TClipBorad 클래스에 대해서는 17장, "클립보드를 통한 정보 공유"에서 이야기하겠다.

메인 폼

메인 폼은 여러 MDI 폼을 띄우거나 관리하는 폼이다. 메인 폼 클래스는 TMainForm이며, 인스턴스 이름은 당연히 MainForm이다. MainForm은 텍스트 편집 폼이나 RTF 편집 폼, 비트맵 보기 폼 같은 MDI 자식 폼들의 부모 역할을 한다.

TMainForm은 다른 자식 폼들과는 달리, TMDIChildForm을 상속하지 않는다.

TMainForm의 FormStyle은 fsMDIForm이다(다른 자식 폼들은 TMDIChildForm의 fsMDIChiLd를 상속받는다). TMainForm은 TMainMenu와 TOpenDialog 컴포넌트를 포함하며, 단 하나의 버튼으로 구성된 툴바도 포함하고 있다. 목록 16.5는 TMainForm을 정의하는 소스 코드이다.

목록 16.5 TMainForm을 정의하는 MainFrm.pas의 소스 코드

```
unit MainFrm;

interface

uses
  WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Messages, Dialogs, SysUtils, ComCtrls,
  ToolWin,  ExtCtrls, Buttons, ImgList;
```

```
type
  TMainForm = class(TForm)

     { 폼을 구성하는 컴포넌트들은 생략했음.
         부록 CD의 소스를 참고하기 바람. }

    { File 메뉴의 이벤트 처리 메소드들 }
    procedure mmiNewClick(Sender: TObject);
    procedure mmiOpenClick(Sender: TObject);
    procedure mmiExitClick(Sender: TObject);

    { Window 메뉴의 이벤트 처리 메소드들 }
    procedure mmiTileClick(Sender: TObject);
    procedure mmiArrangeIconsClick(Sender: TObject);
    procedure mmiCascadeClick(Sender: TObject);
    procedure mmiCloseAllClick(Sender: TObject);
  public
    { 사용자 정의 메소드들 }
    procedure OpenTextFile(EditForm: TForm; Filename: string);
    procedure OpenBMPFile(FileName: String);
    procedure OpenRTFFile(RTFForm: TForm; FileName: string);
    procedure SetToolBar(AToolBar: TToolBar);
  end;

var
  MainForm: TMainForm;

implementation
uses MDIBmpFrm, MdiEditFrm, MdiRtfFrm, FTypForm;

const
  { 파일 확장자를 뜻하는 상수들 }
  BMPExt      = '.BMP';  // 비트맵 파일
  TextExt     = '.TXT';  // 텍스트 파일
  RTFExt      = '.RTF';  // Rich Text Format 파일

{$R *.DFM}

procedure TMainForm.mmiNewClick(Sender: TObject);
```

```
begin
 { GetFileType 함수를 이용해서 새로 열고자 하는 파일의 종류를 얻고,
     그에 따라 적절한 파일 열기 메소드를 호출한다. }
  case GetFileType of
    mrTXT: OpenTextFile(nil, ''); // 텍스트 파일을 연다.
    mrRTF: OpenRTFFile(nil, '');  // RTF 파일을 연다.
    mrBMP:
     begin
              { OpenDialog의 기본 필터를 BMP로 설정한다. }
       OpenDialog.FilterIndex := 2;
       mmiOpenClick(nil);
     end;
  end;
end;

procedure TMainForm.mmiOpenClick(Sender: TObject);
var
  Ext: string[4];
begin
 { OpenDialog에서 선택된 파일의 종류에 따라 적절한 메소드를 호출한다. }
  if OpenDialog.Execute then
  begin
     { 파일의 확장자를 얻어서 파일 종류를 판단하고, 적절한 파일 열기
         메소드에 파일 이름을 넘겨서 파일을 연다. }
    Ext := ExtractFileExt(OpenDialog.FileName);
    if CompareStr(UpperCase(Ext), TextExt) = 0 then
      OpenTextFile(ActiveMDIChild, OpenDialog.FileName)
    else if CompareStr(UpperCase(Ext), BMPExt) = 0 then
      OpenBMPFile(OpenDialog.FileName)
    else if CompareStr(UpperCase(Ext), RTFExt) = 0 then
      OpenRTFFile(ActiveMDIChild, OpenDialog.FileName);
  end;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  Close;
end;
```

```
{ Window 메뉴의 이벤트 처리 메소드들  }

procedure TMainForm.mmiTileClick(Sender: TObject);
begin
  Tile;
end;

procedure TMainForm.mmiArrangeIconsClick(Sender: TObject);
begin
   ArrangeIcons;
end;

procedure TMainForm.mmiCascadeClick(Sender: TObject);
begin
  Cascade;
end;


procedure TMainForm.mmiCloseAllClick(Sender: TObject);
var
  i: integer;
begin
 { MDIChildren 속성을 통해서 모든 자식 폼들을 역순으로 종료시킨다. }
  for i := MdiChildCount - 1 downto 0  do
    MDIChildren[i].Close;
end;

{ 사용자 정의 메소드들 }
procedure TMainForm.OpenTextFile(EditForm: TForm; FileName: string);
begin
  { EditForm이 TMdiEditForm형이면 사용자에게 파일의 내용을 현재 폼에 로드할
     것인지 묻는다. 로드하지 않는다고 하면 TEditForm의 새 인스턴스를 생성하고,
     파일을 그 인스턴스에 로드한다. }
  if (EditForm <> nil) and (EditForm is TMdiEditForm) then
    if MessageDlg('Load file into current form?', mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
    begin
      TMdiEditForm(EditForm).OpenFile(FileName);
       Exit;
```

```
    end;
 { TMdiEDitForm의 새 인스턴스를 생성하고, 그 인스턴스의 OpenFile() 메소드를
     호출한다. }
  with TMdiEditForm.Create(self) do
    if FileName <> '' then
      OpenFile(FileName)
end;

procedure TMainForm.OpenRTFFile(RTFForm: TForm; FileName: string);
begin
  { RTFForm이 TMdiRTFForm형이면 사용자에게 파일의 내용을 현재 폼에 로드할
     것인지 묻는다. 로드하지 않는다고 하면 TEditForm의 새 인스턴스를 생성하고,
     파일을 그 인스턴스에 로드한다. }
  if (RTFForm <> nil) and (RTFForm is TMdiRTFForm) then
    if MessageDlg('Load file into current form?', mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then begin
        (RTFForm as TMdiRTFForm).OpenFile(FileName);
      Exit;
    end;
 { TMdiRTFForm의 새 인스턴스를 생성하고, 그 인스턴스의 OpenFile() 메소드를
     호출한다. }
    with TMdiRTFForm.Create(self) do
      if FileName <> '' then
        OpenFile(FileName);
end;

procedure TMainForm.OpenBMPFile(FileName: String);
begin
  { TMdiBmpForm의 새 인스턴스를 생성하고, BMP 파일을 그 인스턴스에 로드한다. }
  with TMdiBmpForm.Create(self) do
    OpenFile(FileName);
end;

procedure TMainForm.SetToolBar(AToolBar: TToolBar);
begin
  if (MDIChildCount = 1) and (AToolBar = nil) then
    tlbMain.Parent := self
  else begin
    tlbMain.Parent := nil;
```

```
    if not (AToolBar = nil) then
      AToolBar.Parent := self;
  end;
end;

end.
```

TMainForm은 TFileTypeForm형의 FileTypeForm이라는 폼을 사용하는데, 이 폼의 소스 코드는 목록 16.6과 같다.

목록 16.6 TFileTypeForm을 정의하는 FTypForm.pas의 소스 코드

```
unit FTypForm;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Buttons;

const
  mrTXT = mrAll+1;
  mrBMP = mrAll+2;
  mrRTF = mrAll+3;

type

  TFileTypeForm = class(TForm)
    rgFormType: TRadioGroup;
    btnOK: TButton;
    procedure btnOkClick(Sender: TObject);
  end;

var
  FileTypeForm: TFileTypeForm;
```

```
function GetFileType: Integer;

implementation

function GetFileType: Integer;
{ 이 함수는 사용자가 선택한 파일의 종류를 돌려준다. 리턴값은 위에서
 정의한 세 상수들 중 하나이다. }
begin
  FileTypeForm := TFileTypeForm.Create(Application);
  try
    Result := FileTypeForm.ShowModal;
  finally
    FileTypeForm.Free;
  end;
end;

{$R *.DFM}

procedure TFileTypeForm.btnOkClick(Sender: TObject);
begin
 { 선택된 파일의 종류에 따라 적절한 모달 결과값을 돌려준다. }
  case rgFormType.ItemIndex of
   0: ModalResult := mrTXT;
   1: ModalResult := mrRTF;
   2: ModalResult := mrBMP;
  end;
end;

 end.
```

TFileTypeForm은 사용자가 파일 종류를 선택할 때 사용하는 폼이다. 이 폼은 모달로 표시되며, 이 폼의 ModalResult 속성은 사용자가 선택한 TRadioButton의 상태에 따라 정해진다. TFileTypeForm 인스턴스의 생성, 표시, 해제를 담당하는 것은 GetFileType() 함수이다. 이 함수는 TFileTypeForm.ModalResult 속성의 값을 리턴값으로 돌려준다. 이 폼은 자동으로 생성하지 않기 때문에 Project Options 대화상자의 Auto-create forms 목록에서 제거했다.

TMainForm의 툴바에는 단 하나의 버튼만 있다. 그 버튼의 역할은 새 자식 폼을 여는 것이다. 한 자식 폼이 활성화되면 메인 폼의 툴바는 그 자식 폼의 툴바로 대체된다. 그러한 작업을 하는 것은 SetToolBar() 메소드인데, TMdiChildForm의 OnFormActivate와 OnFormDestroy 이벤트 처리 메소드들이 SetToolBar() 메소드를 호출한다.

File | Open 메뉴 항목을 클릭하면 실행되는 TMainForm.mmiNewClick()에서는 파일의 종류에 따라 TMainForm의 공용 메소드인 OpenTextFile()과 OpenRTFFile(), 그리고 OpenBMPFile() 중 하나를 호출한다.

OpenTextFile()은 TForm의 인스턴스와 파일 이름을 인자로 받는다. TForm 인스턴스는 애플리케이션 안에서 현재 활성화된 폼을 뜻한다. OpenTextFile()은 인자로 넘겨받은 TForm의 인스턴스를 검사해서 현재 활성화된 폼이 TMdiEditForm의 인스턴스이면 사용자에게 파일의 내용을 현재 폼에 로드할 것인지 아니면, 새로운 자식 폼에 로드할 것인지 묻고, 사용자의 선택에 따라 적절한 처리를 한다. 이렇게 새 폼을 만들 것인지, 현재 폼에 로드할 것인지를 사용자에게 묻는 것은 사용자에 대한 배려 차원의 문제이다. 파일 열기를 이런 식으로 처리하면 사용자는 파일 열기 메뉴 하나로 새 폼 생성과 파일 열기를 모두 처리할 수 있다.

OpenRTFFile()은 OpenTextFile()과 동일한 논리로 작동된다. 단, TForm 인자가 TMdiRTFForm인지 검사한다는 점이 다르다.

OpenBMPFile()은 항상 새 폼에 파일을 로드하는 것으로 간주한다. TMdiBmpForm은 비트맵의 표시 기능만 있지, 편집 기능은 없기 때문이다. 만일 TMdiBmpForm에 편집 기능까지 추가한다면 OpenBMPFile() 역시 OpenTextFile()이나 OpenRTFFile()처럼 구현해야 할 것이다.

mmiNewClick() 이벤트 처리 메소드는 GetFileType() 함수를 호출해서 사용자에게 파일 종류를 선택하도록 한다. 그런 다음, 선택한 파일 종류에 해당하는 OpenXXXFile() 함수를 호출한다. 만일 파일이 .bmp 파일이면 이 애플리케이션에 비트맵 파일을 새로 만드는 기능은 없으므로 당연히 기존의 비트맵 파일을 여는 것이다. 따라서, OpenDialog를 띄우기 전에 OpenDialog.Filter의 속성을 BMP 필터로 설정해서 사용자가 비트맵 파일을 선택하기 편하게 만들어준다.

mmiOpenClick() 이벤트 처리 메소드는 OpenDialog를 띄우고, 선택한 파일의 종류에 따라 적절한 OpenXXXFile() 메소드를 호출한다. 이때 OpenTextFile()과 OpenRTFFile()에 대해서는 첫 번째 인자에 TMainForm.ActiveMDIChild 속성을 넣는다는 점을 주목하기 바란다.

ActiveMDIChild 속성은 현재 활성화되어 있는 자식 폼을 뜻한다. OpenTextFile()과 OpenRTFFile()은 현재 활성화된 폼에 따라 새 폼을 열 것인지, 기존 폼에 파일을 로드할 것인지 판단한다고 했었다. 만일 활성화된 폼이 없으면 ActiveMDIChild 속성은 nil이 된다. 그리고 ActiveMDIChild가 TMdiRTFForm을 가리키는 상태에서 OpenTextFile()이 호출되어도 다음 문장 때문에 OpenTextFile()은 제대로 작동하게 된다.

```
if (EditForm <> nil) and (EditForm is TMdiEditForm) then
```

EditForm 인자로 넘어온 ActiveMDIChild 속성이 nil이거나 TMdiEditForm이 아니면 무조건 새 폼을 생성한다.

mmiExitClick() 이벤트 처리 메소드는 TMainForm.Close()를 호출한다. 이 메소드는 메인 폼을 닫으며, 따라서 애플리케이션 자체가 종료된다. 이 프로시저가 호출될 때 남아 있던 자식 폼들도 모두 종료된다.

Window 메뉴 항목들의 이벤트 처리 메소드들은 메인 폼의 클라이언트 영역에서 MDI 자식 폼들이 배치되는 방식을 변경하는 작업을 한다. 그림 16.2는 바둑판식 배열(Tile)이고, 그림 16.3은 계단식 배열(Cascade)이다.

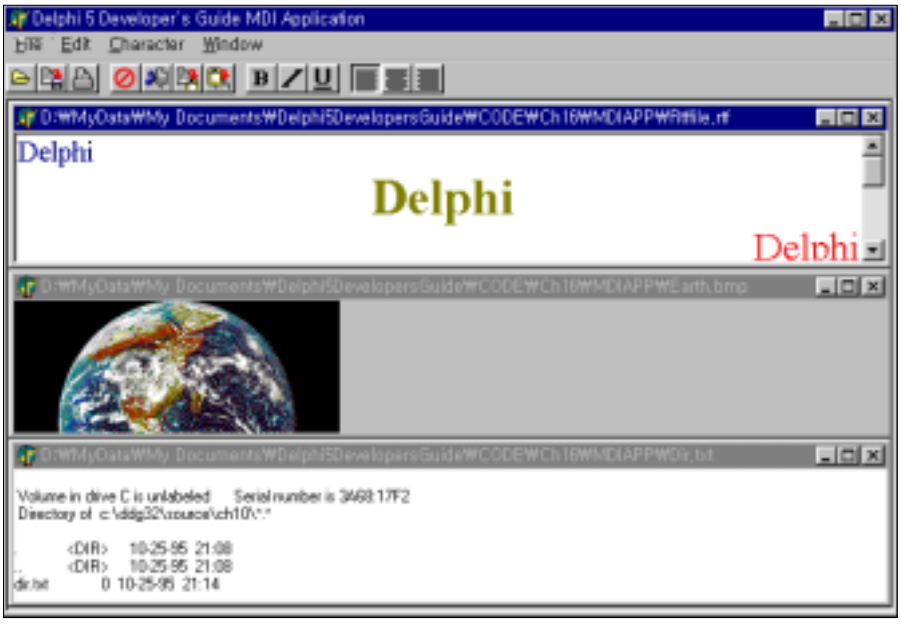


그림 16.2
바둑판식으로 배열된 자식 폼들

mmiArrangeIconsClick() 이벤트 처리 메소드는 메인 폼의 클라이언트 영역에 있는 아이콘(제목 표시줄만 있는 상태)들을 메인 폼의 클라이언트 영역의 아랫쪽에 모아서 배치한다.

mmiCloseAllClick() 이벤트 처리 메소드는 열려진 모든 MDI 자식 폼들을 닫는다. MDIChildren 배열 속성에 담긴 자식 폼들을 역순으로 닫는데, 처음부터 닫으면 자식 폼들의 순서가 하나씩 당겨지기 때문에 문제가 생긴다. MDIChildren 속성은 0부터 시작하는 배열로, 애플리케이션에 생성된 모든 MDI 자식 폼들의 인스턴스를 담고 있다. 생성된 자식 폼들의 개수는 MDIChildCount 속성에 담겨 있다.

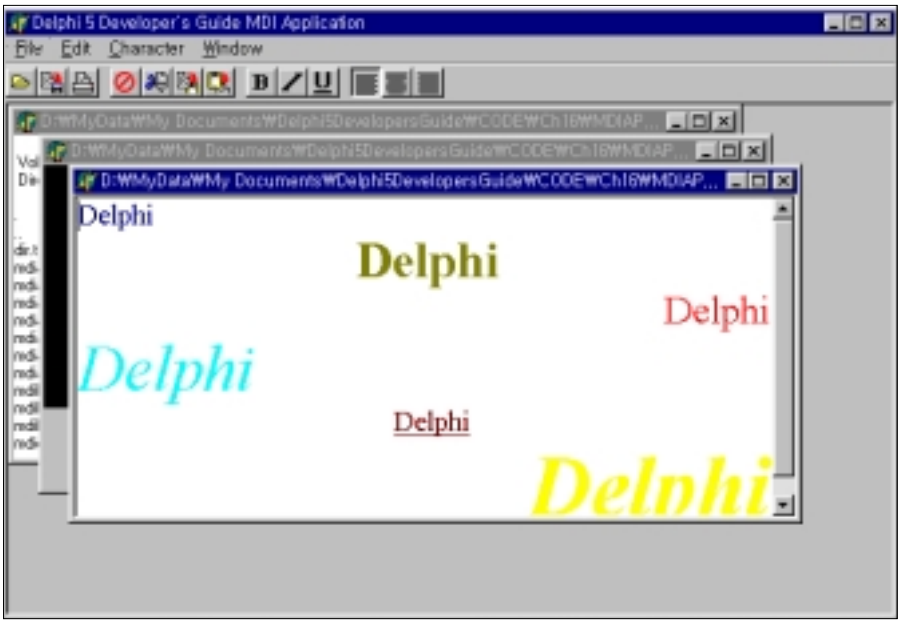


그림 16.3
계단식으로 배열된 자식 폼들

이것으로, 예제 MDI 애플리케이션의 기능에 대한 설명이 끝났다. 다음 부분은 이번 예제에서 사용한 몇 가지 기법들과 예제 애플리케이션의 여러 폼들에서 사용한 몇 가지 컴포넌트들에 대한 설명이다.

메뉴 다루기

MDI 애플리케이션이라고 해서 메뉴를 다루는 방법이 크게 다른 것은 아니지만, 몇 가지 알아둬야 할 점들이 있다. 이번 애플리케이션의 경우, 활성화된 자식 폼의 메뉴가 메인 폼의 메뉴에 동적으로 합쳐지는데, 그러한 기법을 "메뉴 병합"이라고 한다. MDI의 메뉴 병합 기법을 익힌다면 MDI가 아닌 애플리케이션에서도 메뉴들을 동적으로 합치거나 변경하는 능력이 생길 것이다.

MDI 애플리케이션의 메뉴 병합

일단 TMdiEditForm의 TMainMenu 컴포넌트를 더블클릭해서 메뉴 편집기를 띄워보자.

TMdiEditForm의 메인 메뉴는 크게 세 개의 항목, 즉 File과 Edit, Character로 구성되어 있다. 메뉴 편집기에서 각각의 항목을 클릭해서 객체 조사기(Object Inspector)에 나타난 GroupIndex 속성을 확인해 보기 바란다. File 메뉴 항목의 GroupIndex 속성의 값은 0이고, Edit와 Character는 둘 다 1이다.

TMainForm의 메인 메뉴 줄에는 File과 Window 두 개밖에 없다는 점을 주목하기 바란다. TMdiEditForm과 마찬가지로, TMainForm의 File 메뉴 항목의 GroupIndex값은 0이다. 반면, Window 메뉴 항목의 GroupIndex값은 9이다.

또한, TMainForm의 File 메뉴 항목과 TMdiEditForm의 File 메뉴 항목은 부 항목들이 다르다는 점도 주목하기 바란다. TMdiEditForm의 File 메뉴에는 TMainForm의 File 메뉴보다 더 많은 부 항목들이 있다.

폼의 메뉴들을 합치는 데 중요한 역할을 하는 것이 바로 GroupIndex 속성이다. 메인 폼이 어떤 자식 폼을 띄울 때 그 자식 폼의 메인 메뉴는 메인 폼의 메인 메뉴에 병합된다. 이때 메뉴 항목들의 순서라든지, 메인 폼의 어떤 메뉴 항목에 자식 폼의 메뉴 항목들을 합칠 것인지를 결정하는 것이 GroupIndex 속성이다. 그런데 메뉴 병합은 메뉴 줄에 나타난 주 메뉴 항목들에만 적용되며, 부 항목들에는 적용되지 않는다는 점을 기억하기 바란다.

자식 폼의 어떤 메뉴 항목의 GroupIndex 속성이 메인 폼의 어떤 메뉴 항목의 GroupIndex 속성과 같은 값을 가지면 메인 폼의 그 메뉴 항목은 자식 폼의 해당 메뉴 항목으로 대체된다. 나머지 메뉴들은 GropuIndex 속성으로 지정된 순서에 따라서 적절하게 메뉴 줄에 배치된다. 예를 들어서, TMdiEditForm이 활성화되면 메인 폼의 메뉴 줄에 File, Edit, Character, Window 순서로 메뉴 항목들이 나타난다. 이때 두 폼의 File 메뉴 항목은

GroupIndex 속성의 값이 같으므로 메인 폼에 나타난 File 메뉴는 TMdiEditForm의 File 메뉴이다. 즉, TMainForm의 File 메뉴가 TMdiChildForm의 File 메뉴로 대체된 것이다. 그리고 메뉴 줄에 나타나는 메뉴 항목들의 순서는 각 메뉴 항목들의 GroupIndex 속성의 값에 따른다. 즉, 0, 1, 1, 9의 순서로 나타나는 것이다.



이번 예제에서는 따르지 않았지만, GroupIndex 속성값 매기기에 대한 지침이 있다. 주로 OLE 컨테이너의 메뉴 합병에 관련된 것으로, Borland Delphi Library Reference Guide를 참고하면 된다.

이 MDI 예제 프로그램의 다른 자식 폼들에 대해서도 동일한 방식으로 메뉴 합병이 일어난다. 어떤 자식 폼이 활성화되면 메인 폼의 메뉴 줄에 메인 폼의 메뉴뿐만 아니라, 그 자식 폼의 메뉴도 나타나는 것이다.

MDI 애플리케이션의 메뉴 합병은 자동적으로 일어난다. 메뉴 항목들의 GroupIndex 속성을 적절히 설정하기만 하면 활성화된 자식 폼의 메뉴가 메인 메뉴에 자동으로 합병되는 것이다.

MDI가 아닌 애플리케이션의 경우에는 메뉴 합병이 자동적으로 일어나지 않으므로 코드를 통해서 직접 메뉴 합병을 지시해야 하는데, 그에 관련된 기법은 4장, “애플리케이션 프레임웍과 설계 개념들”에서 예제 애플리케이션의 TNavStatForm을 설명할 때 잠깐 선보인 바 있다. 그 애플리케이션에서는 메인 폼이 아니라 어떤 컨트롤의 자식 윈도우에 있는 메뉴를 Merge()와 Unmerge() 함수를 통해서 메인 폼의 메뉴에 병합시켰었다. 어쨌든, MDI가 아닌 애플리케이션에서는 기본적으로 메뉴 병합이 자동으로 일어나지 않는다는 점을 기억하기 바란다. 메뉴 병합이 자동으로 일어나려면 메인 폼의 메뉴에 병합시킬 다른 폼의 TMainMenu의 AutoMerge 속성을 True로 해야 하는데, 이때 메인 폼의 TMainMenu의 AutoMerge 속성은 False로 남겨두어야 한다. MDI가 아닌 애플리케이션의 메뉴 합병에 대해서는 부록 CD의 NonMDI.dpr 프로젝트를 참고하기 바란다.

열린 문서 목록을 메뉴에 추가하기

메인 폼의 Window 메뉴에는 현재 열려진 문서, 즉 자식 폼들의 목록이 나타난다. 이런 처리는 폼의 WindowMenu 속성을 이용한 것이다. 이번 예제 애플리케이션의 경우, TMainForm.WindowMenu 속성은 Window 메뉴 항목, 즉 mmiWindow로 설정되어 있다. MDI 메인 폼에 자식 폼이 하나 생성되면 WindowMenu 속성이 가리키는 메뉴 항목에 그 자식 폼을 가리키는 부 메뉴 항목이 자동으로 추가된다. 물론, 자식 폼이 닫히면 그 부 메뉴 항목도 사라진다. 그런데 WindowMenu 속성에는 주 메뉴 항목, 즉 메뉴 줄에 나타나는 메뉴 항목만 넣을 수 있다는 점을 주의하기 바란다.

기타 MDI 기법들

다음은 MDI 애플리케이션에 적용할 수 있는 일반적인 기법들이다.

MDI 클라이언트 윈도우에 비트맵 넣기

MDI 애플리케이션들은 클라이언트 윈도우에 회사 로고나 배경 그림 등의 비트맵을 넣기도 한다. MDI 폼이 아니라면 그냥 폼의 캔버스나 TImage 컴포넌트의 캔버스에 비트맵을 뿌리면 된다. TImage 컴포넌트가 클라이언트 영역에 꽉차게 하려면 TImage의 Align 속성을 alClient로 해주면 그만이다(MDI 예제 애플리케이션의 비트맵 보기 폼을 참고하기 바란다). 그러나 MDI 애플리케이션의 메인 폼에 비트맵을 표시하는 것은 그리 간단하지 않다.

MDI 애플리케이션의 클라이언트 영역, 즉 클라이언트 윈도우는 메인 폼과는 분리된 윈도우라는 것을 기억하기 바란다. 클라이언트 윈도우는 MDI 관련 작업을 처리하는 역할을 하는데, 그 역할에는 MDI 자식 폼들을 그리는 것도 포함된다.

MDI 메인 폼은 클라이언트 윈도우 위에 겹쳐진 투명한 윈도우라고 생각하면 된다. TButton이나 TImage 같은 컴포넌트를 메인 폼의 클라이언트 영역에 놓으면 실제로는 컴포넌트들이 메인 폼의 투명한 윈도우 위에 놓여지는 것이다. 클라이언트 윈도우가 자식 폼, 즉 자식 윈도우를 자신의 영역에 그리면 그 자식 폼들은 투명한 메인 윈도우 위에 놓여 있는 컴포넌트들 아래에 그려진다(그림 16.4). MDI 애플리케이션은 여러 장의 셀로판지가 겹쳐져 있는 셈이다.

따라서, 비트맵을 메인 폼의 캔버스나 TImage에 그리면 비트맵이 자식 폼들을 가리게 되므로 클라이언트 윈도우에 비트맵을 그리는 수밖에 없다. 그런데 델파이 5에는 MDI 애플리케이션의 클라이언트 윈도우를 직접 다룰 수 있는 VCL 컴포넌트가 없다. 결국, Win32 API를 사용해야 한다는 결론이 나온다. 구체적으로 말하면, 클라이언트 윈도우를 서브클래싱해서 클라이언트 윈도우의 배경을 그리는 작업에 관련된 메시지인 WM_ERASEBKGND를 가로챈 후 클라이언트 영역 표시 작업을 직접 처리해야 한다.

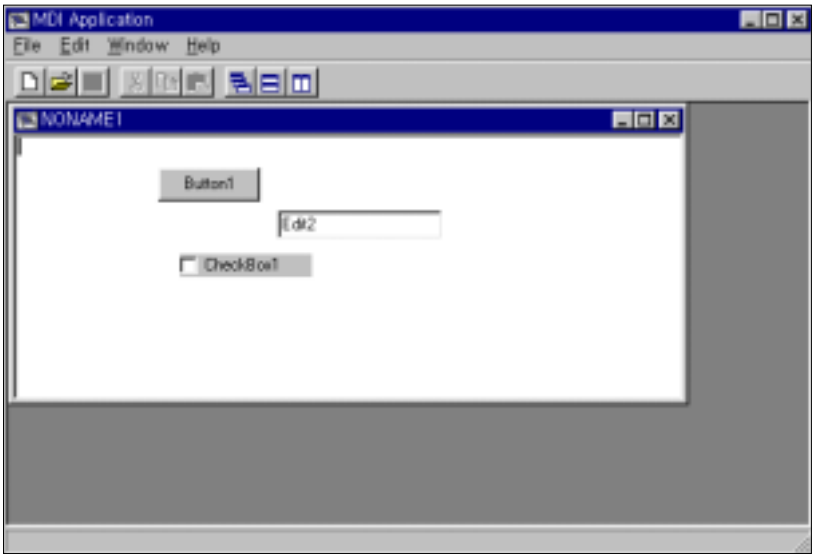


그림 16.4
메인 폼의 컴포넌트들 밑에 자식 폼들이 그려진 모습

그럼, 실제로 클라이언트 윈도우에 배경 이미지를 표시하는 예제 프로젝트를 살펴보자. 부록 CD의 MdiBknd.dpr이 바로 그 예제인데, 이 프로젝트는 TImage 컴포넌트를 이용해서 클라이언트 윈도우에 비트맵을 표시한다. 또, 메뉴를 통해서 비트맵이 나타나는 방식을 가운데, 바둑판식, 늘임(centered/tiled/streched) 중 하나로 선택할 수 있다. 그림 16.5, 그림 16.6, 그림 16.7을 보기 바란다.

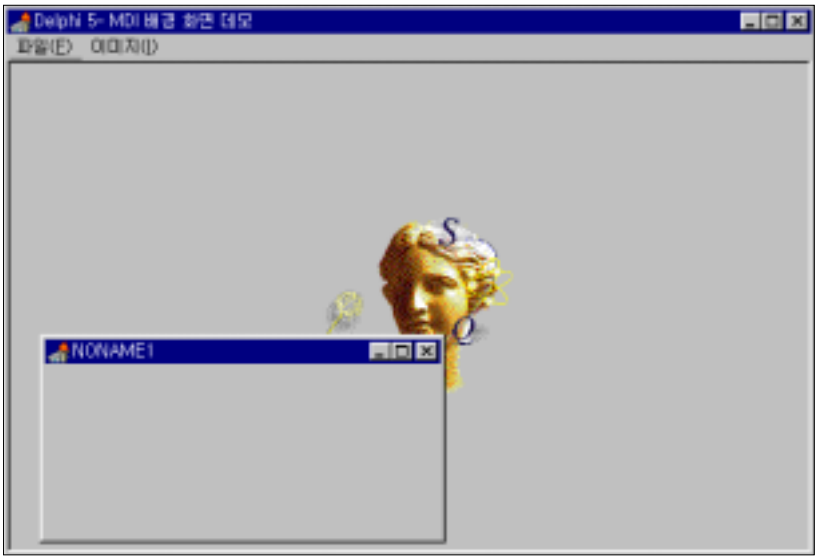


그림 16.5
비트맵 이미지가 MDI 클라이언트 윈도우의 가운데에 표시된 모습



그림 16.6
비트맵 이미지가 MDI 클라이언트 윈도우에 바둑판식으로 표시된 모습



그림 16.7
비트맵 이미지가 MDI 클라이언트 윈도우에 꽉차게 표시된 모습

목록 16.7은 비트맵 표시에 관련된 MDI 메인 폼 유닛의 소스 코드이다.

목록 16.7 Ch16\Mdibknd\MainFrm.pas

```
unit MainFrm;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Dialogs, Buttons, Messages, ExtCtrls;

type
  TMainForm = class(TForm)
    mmMain: TMainMenu;
    mmiFile: TMenuItem;
    mmiNew: TMenuItem;
    mmiClose: TMenuItem;
    N1: TMenuItem;
```

```
    mmiExit: TMenuItem;
    mmiImage: TMenuItem;
    mmiTile: TMenuItem;
    mmiCenter: TMenuItem;
    mmiStretch: TMenuItem;
    imgMain: TImage;
    procedure mmiNewClick(Sender: TObject);
    procedure mmiCloseClick(Sender: TObject);
    procedure mmiExitClick(Sender: TObject);
    procedure mmiTileClick(Sender: TObject);
  private
    FOldClientProc,
    FNewClientProc: TFarProc;
    FDrawDC: hDC;
    procedure CreateMDIChild(const Name: string);
    procedure ClientWndProc(var Message: TMessage);
    procedure DrawStretched;
    procedure DrawCentered;
    procedure DrawTiled;
  protected
    procedure CreateWnd; override;
  end;

var
  MainForm: TMainForm;

implementation

uses MdiChildFrm;

{$R *.DFM}

procedure TMainForm.CreateWnd;
begin
  inherited CreateWnd;
  // ClientWndProc 메소드를 유효한 윈도우 프로시저로 만든다.
  FNewClientProc := MakeObjectInstance(ClientWndProc);
  // 원래 윈도우 프로시저를 가리키는 포인터를 보존한다.
  FOldClientProc := Pointer(GetWindowLong(ClientHandle, GWL_WNDPROC));
```

```
  // ClientWndProc를 새 윈도우 프로시저로 설정한다.
  SetWindowLong(ClientHandle, GWL_WNDPROC, LongInt(FNewClientProc));
end;

procedure TMainForm.DrawCentered;
{ 이 프로시저는 폼의 클라이언트 영역 중앙에 이미지를 표시한다.}
var
  CR: TRect;
begin
  GetWindowRect(ClientHandle, CR);
   with imgMain do
     BitBlt(FDrawDC, ((CR.Right - CR.Left) - Picture.Width) div 2,
            ((CR.Bottom - CR.Top) - Picture.Height) div 2,
            Picture.Graphic.Width, Picture.Graphic.Height,
            Picture.Bitmap.Canvas.Handle, 0, 0, SRCCOPY);
end;

procedure TMainForm.DrawStretched;
{ 이 프로시저는 폼의 클라이언트 영역에 꽉차게 이미지를 표시한다. }
var
  CR: TRect;
begin
  GetWindowRect(ClientHandle, CR);
  StretchBlt(FDrawDC, 0, 0, CR.Right, CR.Bottom,
             imgMain.Picture.Bitmap.Canvas.Handle, 0, 0,
             imgMain.Picture.Width, imgMain.Picture.Height, SRCCOPY);
end;

procedure TMainForm.DrawTiled;
{ 이 프로시저는 폼의 클라이언트 영역에 이미지들을 바둑판식으로 채운다. }
var
  Row, Col: Integer;
  CR, IR: TRect;
  NumRows, NumCols: Integer;
begin
  GetWindowRect(ClientHandle, CR);
  IR := imgMain.ClientRect;
  NumRows := CR.Bottom div IR.Bottom;
  NumCols := CR.Right div IR.Right;
```

```
  with imgMain do
    for Row := 0 to NumRows+1 do
      for Col := 0 to NumCols+1  do
        BitBlt(FDrawDC, Col * Picture.Width, Row * Picture.Height,
               Picture.Width, Picture.Height, Picture.Bitmap.Canvas.Handle,
               0, 0, SRCCOPY);
end;

procedure TMainForm.ClientWndProc(var Message: TMessage);
begin
  case Message.Msg of
    // WM_ERASEBKGND 메시지를 잡아서 클라이언트 영역 표시 작업을 처리
    WM_ERASEBKGND:
      begin
        CallWindowProc(FOldClientProc, ClientHandle,
                  Message.Msg, Message.wParam,
          Message.lParam);
        FDrawDC := TWMEraseBkGnd(Message).DC;
        if mmiStretch.Checked then
          DrawStretched
        else if mmiCenter.Checked then
          DrawCentered
        else DrawTiled;
        Message.Result := 1;
      end;
    { 스크롤 메시지들이면 InvalidateRect를 호출해서 클라이언트 영역을
          다시 그린다. }
    WM_VSCROLL, WM_HSCROLL:
      begin
        Message.Result := CallWindowProc(FOldClientProc, ClientHandle,
                  Message.Msg, Message.wParam, Message.lParam);
        InvalidateRect(ClientHandle, nil, True);
      end;
    else
    // 나머지 메시지들은 원래의 윈도우 프로시저에게 넘긴다.
      Message.Result := CallWindowProc(FOldClientProc,
               ClientHandle, Message.Msg,
        Message.wParam, Message.lParam);
  end; { case }
```

```
end;

procedure TMainForm.CreateMDIChild(const Name: string);
var
  MdiChild: TMDIChildForm;
begin
  MdiChild := TMDIChildForm.Create(Application);
  MdiChild.Caption := Name;
end;

procedure TMainForm.mmiNewClick(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
end;

procedure TMainForm.mmiCloseClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    ActiveMDIChild.Close;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.mmiTileClick(Sender: TObject);
begin
  mmiTile.Checked := false;
  mmiCenter.Checked := False;
  mmiStretch.Checked := False;
  { 이 프로시저를 실행시킨 메뉴 항목의 Checked 속성을 전환한다. }
  if Sender is TMenuItem then
    TMenuItem(Sender).Checked := not TMenuItem(Sender).Checked;
  { 클라이언트 영역을 다시 그린다.}
  InvalidateRect(ClientHandle, nil, True);
end;

end.
```

MDI 애플리케이션의 클라이언트 윈도우에 비트맵 이미지를 표시하려면 서브클래싱이라는 기법을 사용해야 한다. 서브클래싱에 대해서는 5장, "메시지의 이해"에서 이야기했다. 클라이언트 윈도우를 서브클래싱하려면 우선 클라이언트 윈도우의 원래의 윈도우 프로시저에 대한 포인터를 보존해야 한다. 그래야 기본적인 메시지 처리를 그 윈도우 프로시저에게 넘겨줄 수 있다. 폼 변수 FOldClientProc에 원래 윈도우 프로시저에 대한 포인터를 저장하고, 변수 FNewClientProc에는 새 윈도우 프로시저에 대한 포인터를 저장한다.

ClientWndProc()는 FNewClinetProc가 가리키는 프로시저이다. ClientWndProc()는 원래 TMainForm의 메소드이므로 MakeObjectInstance() 함수를 이용해서 Win32 API에 유효한 윈도우 프로시저형의 포인터를 얻는다. 이에 대한 것은 13장, "고난도 기법들"에서 이미 이야기한 대로이다.

TMainForm.CreateWnd() 메소드는 TForm에서 상속받은 메소드를 재정의한 것으로, 이 메소드에서 Win32 API 함수 GetWindowLong()과 SetWindowLong()을 이용해서 클라이언트 윈도우를 서브클래싱한다. 새로운 윈도우 프로시저는 앞에서도 말했듯이 ClientWndProc()이다.

TMainForm에는 DrawCentered(), DrawTiled(), DrawStretched()라는 세 개의 private 메소드들이 있다. 이 세 메소드들은 Win32 GDI 함수들을 이용해서 각기 다른 방식으로 클라이언트 윈도우에 비트맵을 그린다. VCL로는 클라이언트 윈도우의 장치 문맥(DC)을 얻을 수 없으므로 델파이의 표준 그리기 메소드들을 사용할 수 없다. 그래서 Win32 API 함수들을 사용하는 것이다. 물론, API로 DC를 얻어서 TCanvas에 배정한 후 TCanvas의 메소드들을 이용할 수도 있겠지만, Win32 GDI 함수들의 사용법을 익히는 기회라고 생각하기 바란다.

배경 비트맵 표시를 위해서는 WM_ERASEBKGND, WM_VSCROLL, WM_HSCROLL 메시지를 잡아야 한다. WM_ERASEBKGND 메시지는 윈도우가 지워지려고 할 때 발생한다. 항상 나타날 배경 이미지를 그리기에 딱 좋은 메시지인 것이다. 서브클래싱한 새 윈도우 프로시저는 이 메시지가 오면 선택된 메뉴 항목에 기반해서(Tile, Center, Stretch) 적절한 그리기 메소드를 호출한다. 그리고 사용자가 메인 폼을 스크롤하면 배경 이미지를 다시 그려야 하므로 WM_VSCROLL이나 WM_HSCROLL 메시지를 잡아서 배경 이미지를 적절히 다시 그린다. 마지막으로, 새 윈도우 프로시저가 처리하지 않는 나머지 모든 메시지들은 다음과 같은 코드를 통해서 원래의 윈도우 프로시저에게 넘겨준다.

```
Message.Result := CallWindowProc(FOldClientProc, ClientHandle,
                Message.Msg, Message.wParam, Message.lParam);
```

이처럼 VCL이 지원하지 않는 기능이라도 Win32 API를 적절히 사용하면 얼마든지 구현할 수 있다.

숨겨진 MDI 자식 폼 만들기

다음과 같은 문장을 통해서 MDI 자식 폼을 숨기려고 하면 델파이 5는 에러를 낸다.

```
ChildForm.Hide;
```

앞에서 윈도우즈의 MDI 구현에는 문제가 있다고 말한 바 있는데, 이 부분에서도 역시 그렇다. 델파이 개발진들은 MDI 자식 폼을 숨기면 자식 윈도우들의 Z 순서(Z-order, 객체들이 화면에 겹쳐지는 순서를 뜻한다)가 엉킨다는 사실을 알았기 때문에 아예 자식 폼을 숨기지 못하게 만들었다. 따라서, 윈도우즈의 MDI 구현을 그대로 사용하는 한 억지로 자식 폼을 숨기려고 하면 예기치 못한 사태가 발생할 수 있다. 그렇다고는 해도 MDI 자식 폼들을 꼭 숨겨야 할 경우가 생길 수 있다. MDI 자식 폼을 숨기는 방법은 두 가지. 어쨌든, MDI 자식 폼을 숨긴다는 것은 위험을 내포한다는 점을 기억하기 바란다.

MDI 자식 폼을 숨기는 첫 번째 방법은 클라이언트 윈도우가 아예 자식 폼을 그리지 않도록 하는 것이다. Win32 API 함수 LockWindowUpdate()를 이용해서 MDI 클라이언트의 그리기 작업을 막으면 된다. 예를 들어, 초기화 시간이 긴 자식 폼이 있다고 할 때 초기화 작업을 완전히 마친 후에 폼이 화면에 나타나도록 하고 싶다면 이 기법이 유용할 것이다. 초기화 작업에서 데이터베이스 질의 같은 시간이 긴 작업을 수행하다가 만일 질의가 실패하면 자식 폼을 화면에 나타내지 않고 제거해야 한다. 만일 자식 폼이 화면에 표시된 상태에서 초기화 작업을 수행하면 화면이 껌벅거릴 수도 있고, 또 초기화 작업에서 에러가 난 경우 폼이 나타났다가 그냥 사라지는 모습을 사용자에게 보이는 것도 좋지 않은 일이다. LockWindowUpdate() 함수는 인자로 지정된 윈도우가 그려지지 않도록 하는 것이다. 한 번에 단 하나의 윈도우만 잠글 수 있다는 점도 기억하기 바란다. 인자에 0을 넣어서 LockWindowUpdate()를 호출하면 이전에 LockWindowUpdate()로 잠근 윈도우가 풀린다.

또 하나의 방법은, Win32 API 함수 ShowWindow()를 이용해서 자식 폼을 실제로 숨기는 것이다. 이 함수에 SW_HIDE 플래그를 지정하면 된다. 폼을 다시 나타나게 할 때에는 SetWindowPos() 함수를 사용한다. 이 기법은 이미 생성되어서 화면에 나타난 MDI 자식 폼을 숨기거나 다시 나타나게 할 때 유용하다.

목록 16.8은 지금까지 이야기한 기법을 보여주는 예제 MdiHide.dpr의 메인 폼 유닛이다.

목록 16.8 Ch16\Mdihide\MainFrm.pas

```
unit MainFrm;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Dialogs, Buttons, Messages, ExtCtrls, ComCtrls, MdiChildFrm;

type
  TMainForm = class(TForm)
    mmMain: TMainMenu;
    mmiFile: TMenuItem;
    mmiNew: TMenuItem;
    mmiClose: TMenuItem;
    mmiWindow: TMenuItem;
    N1: TMenuItem;
    mmiExit: TMenuItem;
    mmiHide: TMenuItem;
    mmiShow: TMenuItem;
    mmiHideForm: TMenuItem;
    procedure mmiNewClick(Sender: TObject);
    procedure mmiCloseClick(Sender: TObject);
    procedure mmiExitClick(Sender: TObject);
    procedure mmiHideClick(Sender: TObject);
    procedure mmiShowClick(Sender: TObject);
    procedure mmiHideFormClick(Sender: TObject);
  private
    procedure CreateMDIChild(const Name: string);
  public
    HideForm: TMDIChildForm;
```

```
    Hidden: Boolean;
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.CreateMDIChild(const Name: string);
var
  MdiChild: TMDIChildForm;
begin
  MdiChild := TMDIChildForm.Create(Application);
  MdiChild.Caption := Name;
end;

procedure TMainForm.mmiNewClick(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
end;

procedure TMainForm.mmiCloseClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    ActiveMDIChild.Close;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.mmiHideClick(Sender: TObject);
begin
  if Assigned(HideForm) then
    ShowWindow(HideForm.Handle, SW_HIDE);
  Hidden := True;
```

```
end;

procedure TMainForm.mmiShowClick(Sender: TObject);
begin
  if Assigned(HideForm) then
    SetWindowPos(HideForm.handle, HWND_TOP, 0, 0, 0, 0, SWP_NOSIZE
       or SWP_NOMOVE or SWP_SHOWWINDOW);
  Hidden := False;
end;

procedure TMainForm.mmiHideFormClick(Sender: TObject);
begin
  if not Assigned(HideForm) then
  begin
    if MessageDlg('Create Hidden?', mtConfirmation,
              [mbYes, mbNo], 0) = mrYes then
    begin
       LockWindowUpdate(Handle);
      try
        HideForm := TMDIChildForm.Create(Application);
        HideForm.Caption := 'HideForm';
        ShowMessage('Form created and hidden. Press OK to show form');
      finally
        LockWindowUpdate(0);
      end;
    end
    else begin
      HideForm := TMDIChildForm.Create(Application);
      HideForm.Caption := 'HideForm';
    end;
  end
  else if not Hidden then
      HideForm.SetFocus;
end;

 end.
```

이 프로젝트는 간단한 MDI 애플리케이션이다. mmiHideFormClick() 이벤트 프로시저는 숨겨진 채로 생성되거나, 나중에 사용자가 숨길 수 있는 자식 폼을 생성한다.

mmiHideFormClick()은 THideForm의 인스턴스가 생성되어 있는지 점검한다. 만일 THideForm의 인스턴스가 생성되어 있으며, 숨겨져 있지 않은 상태이면 그 인스턴스를 화면에 표시한다. 만일 생성되어 있는 인스턴스가 없으면 사용자에게 숨겨진 채로 폼을 생성할 것인지 묻는다. 사용자가 좋다고 하면 폼을 생성하기 전에 LockWindowUp() 함수를 이용해서 클라이언트 윈도우의 그리기 작업을 잠근다. 따라서, 자식 폼이 생성되어도 화면에는 나타나지 않는다. 자식 폼을 생성한 이후에는 메시지 상자를 통해서 사용자에게 폼이 생성되었다고 알린다. 사용자가 메시지 상자를 닫으면 클라이언트 윈도우의 그리기 작업을 다시 가능하게 한다. 그렇게 하면 자식 폼도 화면에 나타난다. 실제 애플리케이션의 경우에는 메시지 상자를 띄우는 대신, 자식 폼의 초기화 작업을 수행하면 될 것이다. 사용자가 No를 선택하면 숨겨진 상태로 자식 폼을 생성하는 것이 아니라, 일반적인 방식대로 그냥 폼을 생성한다.

이미 생성된 자식 폼들을 Win32 API 함수 ShowWindow()로 숨긴다. ShowWindow()에 자식 폼의 핸들과 SW_HIDE를 넣어서 호출하면 된다. 숨겨진 자식 폼을 다시 표시할 때에는 Win32 API 함수 SetWindowPos()에 자식 폼의 핸들과 적절한 인자들을 넣어서 호출한다. SetWindowPos()는 윈도우의 크기, 위치, Z 순서를 변경할 때 사용하는 함수이다. 이번 예제에서는 Z 순서를 변경해서 폼을 다시 나타나게 한다. Z 순서를 HWND_TOP으로 해서, 다시 말해 자식 폼을 최상위 윈도우로 만들어서 화면에 다시 나타나게 하는 것이다.

모든 MDI 자식 폼들을 한꺼번에 최소화, 최대화, 또는 이전 크기로 하려면

MDI 메인 폼에 있는 모든 자식 폼들을 한꺼번에 다루는 일도 필요하다. 개별 폼의 최소화/최대화/이전 크기 상태는 그 폼의 WindowState 속성으로 변경하면 된다. 현재 존재하는 모든 자식 폼들의 상태를 변경하려면 모든 자식 폼들에 접근할 수 있어야 하는데, 메인 폼의 MDIChildren 배열 속성에 모든 자식 폼들의 인스턴스를 가리키는 포인터들이 들어 있다. 그리고 자식 폼들의 개수는 MDIChildCount 속성에 들어 있다. 두 속성을 이용해서 루프를 돌리고, 각각의 자식 폼의 WindowState를 원하는 값으로 설정하면 끝난다. 부록 CD의 Min_Max.dpr이 모든 자식 폼들을 한꺼번에 최소화, 최대화, 이전 크기로 변경하는 예제이다. 목록 16.9는 Min_Max.dpr의 메인 폼 유닛의 소스 코드인데, 메뉴 항목들의 이벤트 처리 메소드들을 유심히 보기 바란다.

목록 16.9 Ch16\Min_max\MainFrm.pas

```
unit MainFrm;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Dialogs, Buttons, Messages, ExtCtrls, ComCtrls;

type
  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
    mmiFile: TMenuItem;
    mmiNew: TMenuItem;
    mmiClose: TMenuItem;
    mmiWindow: TMenuItem;
    N1: TMenuItem;
    mmiExit: TMenuItem;
    mmiMinimizeAll: TMenuItem;
    mmiMaximizeAll: TMenuItem;
    mmiRestoreAll: TMenuItem;
    procedure mmiNewClick(Sender: TObject);
    procedure mmiCloseClick(Sender: TObject);
    procedure mmiExitClick(Sender: TObject);
    procedure mmiMinimizeAllClick(Sender: TObject);
    procedure mmiMaximizeAllClick(Sender: TObject);
    procedure mmiRestoreAllClick(Sender: TObject);
  private
    { Private 선언부 }
    procedure CreateMDIChild(const Name: string);
  public
    { Public 선언부 }
  end;

var
  MainForm: TMainForm;

implementation
```

```
uses MdiChildFrm;

{$R *.DFM}

procedure TMainForm.CreateMDIChild(const Name: string);
var
  Child: TMDIChildForm;
begin
  Child := TMDIChildForm.Create(Application);
  Child.Caption := Name;
end;

procedure TMainForm.mmiNewClick(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
end;

procedure TMainForm.mmiCloseClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    ActiveMDIChild.Close;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.mmiMinimizeAllClick(Sender: TObject);
var
  i: integer;
begin
  for i := MDIChildCount - 1 downto 0 do
    MDIChildren[i].WindowState := wsMinimized;
end;

procedure TMainForm.mmiMaximizeAllClick(Sender: TObject);
var
  i: integer;
```

```
begin
  for i := 0 to MDIChildCount - 1 do
    MDIChildren[i].WindowState := wsMaximized;
end;

procedure TMainForm.mmiRestoreAllClick(Sender: TObject);
var
  i: integer;
begin
  for i := 0 to MDIChildCount - 1 do
    MDIChildren[i].WindowState := wsNormal;
end;

 end.
```

요약

이번 장에서는 델파이 5로 MDI 애플리케이션을 만드는 방법을 살펴보았다. 또한, MDI 애플리케이션에 사용할 수 있는 고급 기법들도 몇 가지 나왔다. 이번 장의 내용을 잘 이해한 독자라면 멋진 MDI 애플리케이션을 만들 수 있을 것이다.

쉬어가는 페이지