4

BAB 4.
VISUAL COMPONENT LIBRARY

Yang akan dibahas pada bab ini adalah:

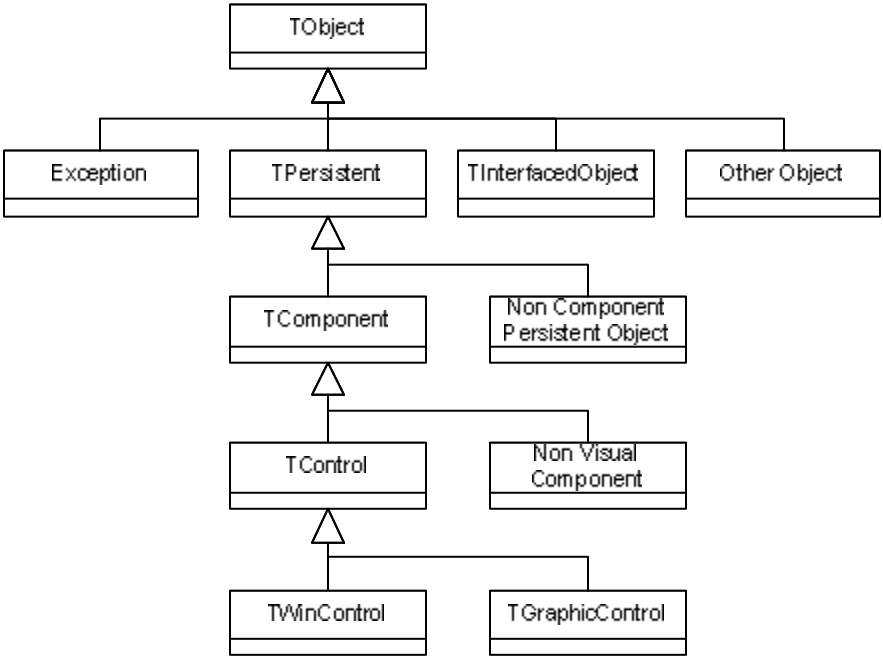
- Struktur Hirarki VCL
- Delphi Component (TComponent)
- Delphi Visual Component (TControl)
- Bekerja dengan Clipboard

Salah satu elemen penting dalam pemrograman menggunakan bahasa dan ***development tools*** tertentu adalah memahami ***application framework*** dari development tools tersebut. Application framework adalah library yang berisi objek-objek yang dapat digunakan oleh programmer dalam membuat program. Delphi menyediakan dua macam application framework yang dapat digunakan, yaitu CLX (Component Library for Cross Platform) yang digunakan untuk

membangun program yang portable antar Delphi dan Kylix, serta VCL (Visual Component Library) yang digunakan hanya untuk mengembangkan aplikasi dengan platform Windows. Mengingat tempat yang terbatas, buku ini hanya akan membahas VCL saja.

Struktur Hirarki dalam VCL

VCL memiliki ribuan objek yang bila digambarkan dengan lengkap akan memerlukan sebuah poster berukuran besar. Pada bagian ini kita akan membahas struktur VCL dalam kerangka yang sangat global. Berikut ini adalah skema yang menggambarkan struktur VCL secara global.



pelajari sebelumnya. Turunan dari Exception misalnya adalah EAbort yang merupakan ***silent exception***, EDivByZero, EAccessViolation, dll.

TInterfacedObject adalah objek yang menyokong (*support*) interface. Fasilitas yang disediakan TInterfacedObject adalah mekanisme reference counting yang memungkinkan interface terhapus dari memory ketika tidak lagi digunakan.

TPersistent adalah turunan lain dari TObject yang berperan sangat penting dalam VCL. Objek ini dicompile dengan compiler directive {$M+}, sehingga objek ini dan turunannya memiliki RTTI (Run-time Type Information), dan memiliki sifat-sifat berikut ini

- dapat memiliki property atau method yang bersifat **published**
- tersedia mekanisme streaming untuk menyimpan objek ke dalam stream
- tersedia sarana untuk melakukan copy dari satu objek ke objek lain (dengan method Assign dan AssignTo)
- tersedia mekanisme untuk menyimpan unpblished data

Untuk objek yang ingin bersifat persistent tetapi juga menyediakan kemampuan untuk menyokong interface, moyangnya adalah **TInterfacedPersistent** yang juga turunan dari TPersistent. Turunan TPersistent lainnya yang sangat penting adalah **TComponent**, yaitu semua objek yang merupakan **component**. Salah satu sifat penting component adalah dapat diinstall di Component Palette (dalam IDE). Selain itu, berbeda dengan TPersistent dimana mekanisme streamingnya masih agak sulit digunakan, mekanisme streaming pada TComponent sudah sangat mudah digunakan. Sifat dan perilaku TComponent dan turunannya akan dibahas dalam satu bagian khusus dalam bab ini. Turunan TPersistent lainnya yang sangat penting adalah **TCanvas**, yang juga akan dibahas secara khusus. Objek objek yang merupakan turunan TPersistent tetapi bukan turunan TComponent biasanya disebut **non component persistent object**. Contohnya TCanvas, TCollection, TCollectionItem, TClipboard, dll.

TComponent menurunkan dua kelompok object, yaitu **visual component**, yaitu semua turunan dari TControl dan **non visual component** (turunan dari TComponent, tapi bukan turunan dari TControl).

TControl menurunkan dua kelompok objek, yaitu TWinControl dan TGraphicControl. **TWinControl** membungkus objek window (dalam MS Windows). Ciri dari objek ini diantaranya adalah memiliki **window handle**, yaitu suatu bilangan integer yang merupakan identifikasi untuk

window (dalam MS Windows). Contoh TWinControl adalah TForm, TPanel, TButton, dll. **TGraphicControl** adalah visual control yang ringan karena tidak menggunakan resource untuk window. Contoh turunan TGraphicControl adalah TImage, TLabel, TPaintBox, dll.

Delphi Component (TComponent)

Seperti telah disebutkan sebelumnya, semua turunan TComponent adalah Delphi Component. Kemampuan yang dimiliki oleh TComponent adalah sebagai berikut

- **Delphi IDE Integration**, TComponent dapat di-install di component palette.
- **Ownership**, yaitu kemampuan untuk mengelola komponen lain. Jika komponen A adalah owner dari komponen B maka komponen A bertanggung jawab untuk men-destroy komponen B ketika komponen A di-destroy.
- **Streaming dan Filing**, yaitu kemudahan dalam menulis ke stream maupun ke file.

Selain itu dalam objek TComponent diperkenalkan beberapa property penting, yaitu

- **Name**, nama komponen.
- **Tag**, property bebas bernilai integer yang tidak digunakan oleh Delphi, sehingga dapat kita manfaatkan untuk berbagai kebutuhan.
- **ComponentCount**, yaitu berisi jumlah komponen yang dimiliki oleh komponen tersebut.
- **Components**, yaitu daftar komponen yang dimiliki component tersebut.
- **ComponentIndex**, yaitu indeks suatu komponen pada dalam list komponent pemiliknya.

Memahami Konsep Owner

Constructor Create yang dimiliki komponen didefinisikan sebagai

```
constructor Create(AOwner: TComponent); virtual;
```

Parameter AOwner dari constructor tersebut adalah komponen yang akan memiliki komponen yang di-create tersebut. Perhatikan contoh berikut ini

```
// Kode 1
procedure TForm1.Button1Click(Sender: TObject);
begin
  with TOpenDialog.Create(Self) do begin
    Name := 'MyDialog1';
    Filter := 'Delphi File (*.pas)|*.pas';
  end;
end;

// Kode 2
procedure TForm1.Button1Click(Sender: TObject);
begin
  with TStringList.Create do begin
    Add ('Testing');
  end;
end;
```

Kode pertama **aman**, karena TOpenDialog yang sudah di-create akan dibuang dari memory ketika Form dibuang (Ownernya adalah instance dari TForm1, karena Self pada konteks procedure tersebut adalah instance dari TForm1). Kode kedua **tidak aman**, karena TStringList bukan komponen sehingga tidak memiliki owner, sehingga setelah objek TStringList di-create tidak ada yang membuangnya.

Untuk mengakses daftar Component yang dimiliki oleh suatu komponen tertentu, kita dapat menggunakan property **ComponentCount** dan **Components**. Contoh berikut ini akan menampilkan semua komponen yang dimiliki oleh Form.

```
procedure TForm1.Button1Click(Sender: TObject);
var i: Integer;
begin
  for i:=0 to ComponentCount-1 do begin
    Memo1.Lines.Add(Components[i].ClassName+' - '+
      Components[i].Name);
  end;
end;
```

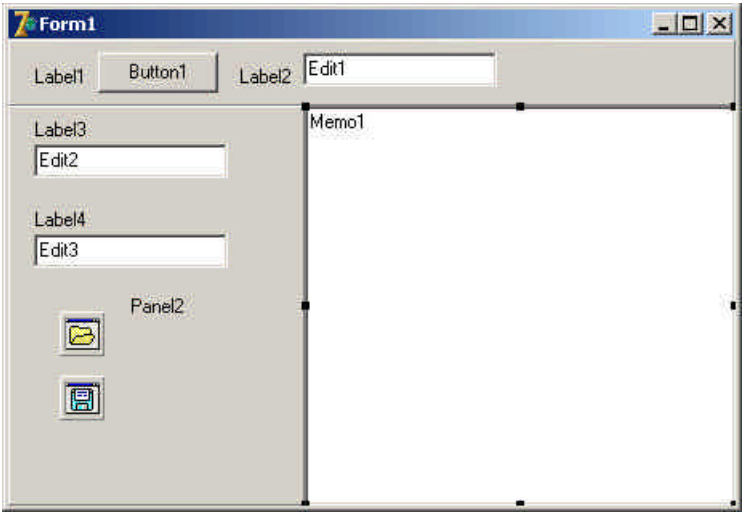
Ketika kita meletakkan sebuah component dalam Form, baik itu langsung ataupun diatas komponen lain (misalnya Edit diatas Panel), pada saat run-time semua komponen tersebut ownernya adalah Form.

Contoh 4-1 Bereksperimen dengan Components

Dalam contoh ini kita akan bereksperimen dengan component. Kita akan mendaftarkan semua komponen yang dimiliki oleh Form dalam sebuah Memo menggunakan property Components.

- Buat aplikasi baru, pada form tambahkan beberapa visual komponen ataupun non visual komponen. Susunannya terserah, tetapi paling

tidak ada sebuah tombol yang event OnClick-nya akan kita gunakan. Misalnya adalah sebagai berikut



- Kemudian pada event OnClick tombol Button1 diatas, tuliskan kode berikut ini

```
procedure TForm1.Button1Click(Sender: TObject);
var i: Integer;
begin
  Memo1.Clear;
  for i:=0 to ComponentCount-1 do begin
    Memo1.Lines.Add(Components[i].ClassName+' - '+
      Components[i].Name);
  end;
end;
```

- Jalankan Program, tekan Button1, maka dalam Memo1 akan muncul hasil berikut ini

```
TPanel - Panel1
TLabel - Label1
TLabel - Label2
TButton - Button1
TEdit - Edit1
TPanel - Panel2
TLabel - Label3
TLabel - Label4
TEdit - Edit2
TEdit - Edit3
TMemo - Memo1
TOpenDialog - OpenDialog1
TSaveDialog - SaveDialog1
```

✍ Selanjutnya kita akan mencoba meng-create beberapa non visual komponen pada event OnCreate dari Form, sebagai berikut

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Komponen yang dicreate adalah komponen dialog
  // sebagai contoh paling mudah untuk non visual component
  with TColorDialog.Create(Self) do begin
    Name := 'Komponen1';
    Color := clRed;
  end;
  with TFontDialog.Create(Self) do begin
    Name := 'Komponen2';
  end;
  // variabel FMyComponent berikut ini harus didefinisikan
  // dulu di bagian private dari Form
  FMyComponent := TPrintDialog.Create(Self);
  TPrintDialog.Create(Self);
end;
```

✍ Setelah program dijalankan, maka hasilnya adalah sebagai berikut

```
TPanel - Panel1
TLabel - Label1
TLabel - Label2
TButton - Button1
TEdit - Edit1
TPanel - Panel2
TLabel - Label3
TLabel - Label4
TEdit - Edit2
TEdit - Edit3
TMemo - Memo1
TOpenDialog - OpenDialog1
TSaveDialog - SaveDialog1
TColorDialog - Komponen1
TFontDialog - Komponen2
TPrintDialog -
TPrintDialog –
```

Perhatikan dua baris terakhir, dimana dua buah TPrintDialog namanya kosong. Dua informasi penting dari kedua baris ini adalah

- Nama komponen boleh kosong
- Nama komponen yang tidak kosong tidak boleh sama

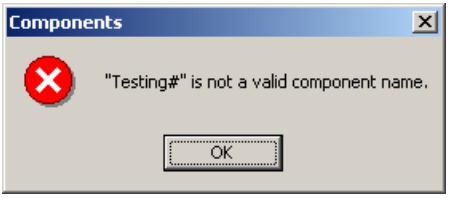
Selanjutnya kita akan bereksperimen dengan nama komponen. Kita akan memerlukan sebuah tombol yang event OnClick-nya akan kita gunakan untuk mengubah property name dari salah satu komponen dalam form kita, misalnya adalah komponen Panel2. Untuk contoh ini kita pilih Panel2, karena komponen nii memiliki caption yang terlihat ketika running.

✍ Tambahkan sebuah tombol (mis Button2), ketik event OnClick dengan kode berikut ini

```
procedure TForm1.Button2Click(Sender: TObject);
var vStr: string;
begin
  vStr := Panel2.Name;
  if InputQuery('Test dengan Komponen', 'Nama Komponen ',
    vStr) then begin
    Panel2.Name := vStr;
  end;
end;
```

✍ Jalankan program, dan tekan Button2. Isikan dengan nama komponen lain yang ada (misalnya Label1) pada input Dialog. Hasilnya akan muncul Error sebagai berikut

- Selanjutnya kita isikan nama dengan karakter aneh (#@'!) atau angka di bagian depan, maka akan muncul error juga.



- Selanjutnya isikan dengan nama yang belum ada (misalnya Test1). Secara visual akan langsung nampak bahwa tulisan yang muncul pada Panel1 akan beruba. Kemudian tekan tombol Button1, maka terlihat bahwa tidak ada lagi TPanel dengan nama Panel2, tetapi yang ada sekarang adalah TPanel dengan nama Test1.
- Bila kita perhatikan komponen Panel2, terlihat bahwa caption dari komponen ini berubah ketika kita mengubah nama komponen. Hal ini terjadi karena Delphi secara otomatis akan mengubah caption ketika name berubah bila caption dan name sama.
- Bila kita tekan lagi tombol Button2, maka akan muncul error access violation

Error diatas muncul karena perubahan nama komponen kita lakukan terhadap field dari form yang bersifat published. Bila nama yang kita ubah adalah dari variabel yang bersifat private (misalnya FMyComponent), maka error diatas tidak akan muncul. Penjelasan teknisnya adalah karena property published dari form sangat erat kaitannya dengan design time, maka secara internal delphi

menyediakan mekanisme tertentu yang melakukan sinkronisasi antara nama field dengan nama komponen. Bila sebuah field published dari form tidak menemukan komponen dari form dengan nama yang sama, maka field tersebut akan diisi dengan nil.

Nama Komponen

Dari contoh yang kita buat diatas, kita memperoleh beberapa pelajaran penting tentang nama komponen, yaitu

- Nama komponen boleh kosong ('')
- Sekumpulan komponen yang ownernya sama, namanya harus unik, kecuali bila namanya kosong ('')
- Penamaan komponen memiliki ketentuan yang sama dengan penamaan identifier, misalnya karakter pertama hanya huruf dan underscore (_), sedangkan karakter berikutnya bisa huruf, angka atau underscore. Karakter lain seperti @#$ dll tidak dapat digunakan.
- Untuk komponen yang memiliki Caption, bila isi Caption sama dengan isi Name dan ketika Name berubah secara otomatis Caption akan ikut berubah sesuai dengan nama baru komponen tersebut. Tapi hal ini tidak terjadi bila Caption tidak sama dengan Name.
- Hindari mengubah nama (saat run-time) dari komponen yang dibuat pada saat design-time (berada pada bagian published dari form), karena sinkronisasi internal dalam Delphi, field published tersebut akan bernilai nil bila nama komponen diubah.

Selain ketentuan diatas yang telah kita lihat dari contoh, ada beberapa hal penting lainnya yang perlu ditambahkan tentang nama komponen, yaitu

- Kita dapat mencari komponen tertentu berdasarkan nama komponen dengan method (function) FindComponent.
- Owner dari AutoCreate Form adalah **Application**, yaitu sebuah variabel untuk objek TApplication yang membungkus windows application. Kita dapat lihat dalam project source ada mekanisme create form oleh aplikasi

```
program Components;

uses
  Forms,
  FMain in 'FMain.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

? Jika kita secara manual meng-create dua buah form yang sama atau lebih, maka secara otomatis form kedua dan seterusnya namanya akan ditambahi dengan nomor urut. Misalnya pada OnCreate dari objek TForm1 ada kode berikut

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with TForm2.Create(Application) do begin
    Caption := Name;
    Show;
  end;
  with TForm2.Create(Application) do begin
    Caption := Name;
    Show;
  end;
  with TForm2.Create(Application) do begin
    Caption := Name;
    Show;
  end;
end;
```

Maka form pertama akan bernama Form2, form kedua akan bernama Form2_1, dan form ketiga akan bernama Form2_2.

Component Streaming

Component menyediakan mekanisme standard untuk menuliskan dan membaca data dari dan ke file. Caranya sangat mudah yaitu dengan menggunakan method dari Stream yang bernama **WriteComponent**. Demikian juga untuk membacanya dengan menggunakan method ReadStream.

```
// Membaca Komponen
vStream1.ReadComponent(FMyComponent);

// Menulis Komponen
vStream.WriteComponent(FMyComponent);
```

Setelah ditulis di stream, komponen tersebut akan berada dalam stream dengan format biner. Kita dapat mengubah format biner tersebut menjadi teks dengan menggunakan method **ObjectBinaryToText** dan sebaliknya untuk mengubah dari teks ke binary kita gunakan **ObjectTextToBinary**. Perlu digarisbawahi bahwa property yang akan ditulis ke dalam stream hanya property yang bersifat **published**.

Berikut ini adalah contoh kode lengkap untuk membaca dan menulis komponen.

Membaca Komponen

```
procedure TForm1.Button1Click(Sender: TObject);
var vStream, vStream1: TMemoryStream;
begin
  if OpenDialog1.Execute then begin
    vStream := TMemoryStream.Create;
    vStream1 := TMemoryStream.Create;
    try
      vStream.LoadFromFile(OpenDialog1.FileName);
      vStream.Position := 0;
      ObjectTextToBinary(vStream, vStream1);
      vStream1.Position := 0;
      vStream1.ReadComponent(FMyComponent);
    finally
      vStream.Free;
      vStream1.Free;
    end;
  end;
end;
```

Menulis Komponen

```
procedure TForm1.Button2Click(Sender: TObject);
var vStream, vStream1: TMemoryStream;
begin
  if SaveDialog1.Execute then begin
    vStream := TMemoryStream.Create;
    vStream1 := TMemoryStream.Create;
    try
      vStream.WriteComponent(FMyComponent);
      vStream.Position := 0;
      ObjectBinaryToText(vStream, vStream1);
      vStream1.Position := 0;
      vStream1.SaveToFile(SaveDialog1.FileName);
    finally
      vStream.Free;
      vStream1.Free;
    end;
  end;
end;
```

Setelah komponen ditulis ke dalam file, kita dapat membacanya kembali kapan saja kita inginkan. Akan tetapi akan muncul masalah bila definisi komponen kita berubah. Sebagai contoh bila sebelumnya kita memiliki definisi komponen berikut ini

```
TMyComponent = class(TComponent)
private
  FTest1: Integer;
  FTest2: Integer;
  FTest3: Integer;
published
  property Test1: Integer read FTest1 write FTest1;
  property Test2: Integer read FTest2 write FTest2;
  property Test3: Integer read FTest3 write FTest3;
end;
```

Kemudian karena sudah tidak dibutuhkan lagi property Test2 kita hapus. Ketika kita baca file yang disimpan ketika property Test2 masih ada, akan muncul Error Message sebagai berikut

Dengan munculnya error ini property-property lainnya setelah terjadinya error ini tidak akan dipanggil. Masalah ini mudah diatasi karena method **ReadComponent** menggunakan objek TReader dalam

melakukan tugasnya. Sementara itu, objek **TReader** tersebut memiliki event **OnError** yang tipenya adalah **TReaderError**

```
type TReaderError = procedure(Reader: TReader; const
  Message: string; var Handled: Boolean) of object;
```

Bila kita ingin mengabaikan error yang muncul ketika kita baca suatu property dari komponen, dalam event handler yang kita buat kita dapat mengeset parameter Handled dengan True.

Untuk melakukannya kita definisikan sebuah function dengan judul SafeReadComponent, sebagai berikut

```
function SafeReadComponent(Stream: TStream; Instance: TComponent;
  ErrHandler: TReaderError): TComponent;
var vReader: TReader;
begin
  vReader := TReader.Create(Stream, 4096);
  vReader.OnError := ErrHandler;
  try
    result := vReader.ReadRootComponent(Instance);
  finally
    vReader.Free;
  end;
end;
```

Contoh pemanggilannya adalah

```
SafeReadComponent(vStream1, FMyComponent, IgnoreError);
```

Dimana procedure IgnoreError adalah sebagai berikut

```
procedure TForm1.IgnoreError(Reader: TReader; const Message:
string;
  var Handled: Boolean);
begin
  // kita bisa juga memeriksa eror apa yang terjadi
  // berdasarkan teks dalam message
  Handled := True;
end;
```

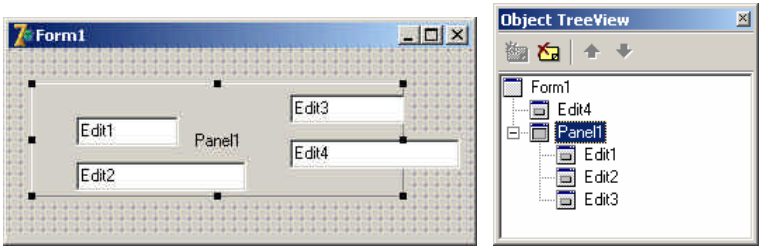
Bila kita jalankan program dan kita baca data yang salah, exception tetap ada, tetapi sudah di-handle oleh IgnoreError. Jadi bila error muncul, itu disebabkan karena option Stop On Delphi Exception diaktifkan. Pada saat program running, exception ini tidak akan muncul. Bila masih belum jelas, baca lagi penjelasan tentang Exception pada bab terdahulu.

Delphi Visual Component (TControl)

Visual Component juga sering disebut Control adalah semua objek yang merupakan turunan dari TControl. Kelas TControl menyediakan berbagai mekanisme berkaitan dengan tampilan di layar dan interaksi dengan user interface.

Property Parent

Setiap control bisa berada di dalam control lain. Ketika kita tambahkan sebuah panel dalam form, maka panel tersebut berada di dalam form. Jika kita tambahkan sebuah komponen edit diatas panel, maka komponen edit tersebut berada di dalam Panel. Perhatikan gambar berikut ini



Pada contoh diatas, Edit1, Edit2, dan Edit3 berada dalam Panel1, tetapi Edit4 tidak. Seperti terlihat dalam gambar diatas, Edit3 tidak dapat digeser keluar dari Panel1. Ini berarti Edit3 berada dalam Panel1. Berbeda dengan Edit4 yang dapat digeser keluar dari Panel1, maka Edit4 tidak di dalam Panel1. Cara lain untuk mengetahui apakah control berada dalam control lain ataukah tidak, kita dapat menggunakan ObjectTreeView untuk melihatnya. Pada gambar diatas, terlihat jelas bahwa Edit4 dan Panel1 berada dalam Form1. Sedangkan Edit1, Edit2, dan Edit3 berada dalam Panel1.

Property Parent berkaitan dengan keberadaan sebuah control dalam control lain. Parent sebuah control adalah tempat dimana control tersebut berada. Pada contoh diatas parent dari Edit4 dan Panel1 adalah Form1. Sedangkan Parent dari Edit1, Edit2, dan Edit3 adalah Panel1.

Kita jangan bingung antara Parent dengan Owner. Owner berkaitan dengan tanggung jawab untuk mendestroy component yang dimilikinya, sedangkan Parent berkaitan dengan penampilan sebuah control di dalam control lain.

Yang memiliki property parent adalah semua turunan dari TControl. Tetapi yang bisa menjadi parent adalah turunan dari TWinControl.

Agar dapat muncul di layar, semua Control kecuali Form harus memiliki Parent. Biasanya Form tidak memiliki parent. Akan tetapi sebagai turunan TControl, Form juga bisa memiliki parent.

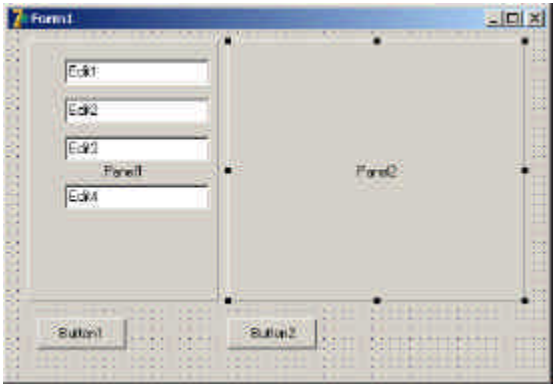
Ketika kita berada dalam Design-Time, kita tidak dapat meletakkan sebuah Edit diatas Button. Tetapi Button adalah turunan dari TWinControl juga, jadi kontrol apapun juga bisa berada di dalam Button (meskipun tidak biasa).

Untuk mengakses semua control yang ada dalam sebuah control, kita dapat menggunakan property **ControlCount** (bertipe Integer) dan **Controls** (bertipe array). Perlu dicatat juga disini bahwa posisi control (property Left dan Top) adalah relatif terhadap parent-nya.

Contoh 4-2 Eksperimen dengan Property Parent

Dalam eksperimen ini, kita akan mencoba memindahkan beberapa control dari satu panel ke panel lainnya.

- Buat sebuah aplikasi baru, tambahkan dua buah panel yang agak lebar dan posisinya berdampingan
- Tambahkan beberapa komponen edit pada Panel1. Tampilannya kurang lebih akan menjadi sebagai berikut



- Selanjutnya tombol Button1 akan kita gunakan untuk memindahkan semua control dalam Panel1 ke Panel2, dan tombol Button2 akan kita gunakan untuk memindahkan semua control dalam Panel2 ke Panel1. Kode yang harus kita tuliskan adalah sebagai berikut

```
procedure TForm1.Button1Click(Sender: TObject);
var i: Integer;
begin
  for i:=Panel1.ControlCount-1 downto 0 do begin
    Panel1.Controls[i].Parent := Panel2;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var i: Integer;
begin
  for i:=Panel2.ControlCount-1 downto 0 do begin
    Panel2.Controls[i].Parent := Panel1;
  end;
end;
```

✍ Jalankan program, dan akan kita lihat bahwa ketika Button1 ditekan maka semua control akan pindah ke Panel2. Demikian juga bila Button2 ditekan maka control akan pindah ke Panel1.

Bila kita perhatikan kode diatas, terlihat bahwa kita mulai loop dari ControlCount menuju 0. Bila kita lakukan sebaliknya maka akan terjadi error List Out of Bound. Penjelasan teknisnya adalah sebagai berikut

? Perhatikan kode berikut ini

```
for i:=0 to Panel1.ControlCount-1 do begin
  Panel1.Controls[i].Parent := Panel2;
end;
```

? Misalnya seperti contoh yang kita buat (lihat gambar), terdapat 4 control dalam Panel1. Jadi selama loop nilai i akan bergerak dari 0 sampai 3.

? Pada saat i=0, Controls[0] (yaitu Edit1) akan dipindahkan ke Panel2, control lainnya akan naik keatas. Dalam hal ini Edit2 menjadi Controls[0], Edit3 menjadi Controls[1] dan Edit4 menjadi Controls[2].

? Pada saat i=1 (putaran loop berikutnya), Controls[1], yaitu Edit3 akan dipindahkan ke Panel2. Ini berarti Edit4 akan menjadi Controls[1].

? Pada saat i=2 (putaran loop berikutnya), Controls[2] tidak ada, karena Edit4 (kontrol terakhir) indeksnya adalah 1, dan tinggal ada dua Control. Sehingga ketika program mencoba mengakses Controls[2] akan terjadi Error List Out of Bound.

Problem ini adalah problem umum untuk semua loop yang bekerja untuk suatu List dan dalam loop tersebut ada elemen list yang

dibuang. Contoh lainnya misalnya kita akan mendestroy semua komponen yang dimiliki oleh komponen tertentu, maka loopnya juga harus menurun seperti diilustrasikan pada kode berikut ini

```
for i:=MyContainer.ComponentCount-1 downto 0 do begin
  MyContainer.Components[i].Free;
end;
```

Masih menggunakan aplikasi yang sama, akan ditunjukkan bahwa suatu form bisa berada diatas panel.

- Tambahkan sebuah tombol (Button3) di dekat Button2.
- Tambahkan sebuah Form baru (Form2) dan isi Form2 tersebut dengan beberapa control. Ubah warna form dengan warna lain, misalnya clCream atau ClWhite. Contoh tampilan Form2 kurang lebih sebagai berikut

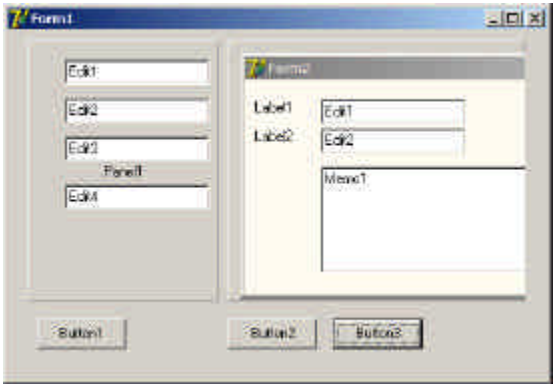
- Save Form2, agar pada saat uses tidak bermasalah, misalnya kita beri nama FForm2.pas.
- Kembali ke Form1, Pada bagian implementation kita tambahkan uses, seperti kode berikut ini

```
uses FForm2;
```

- Tambahkan kode pada event OnClick dari Button3, sebagai berikut

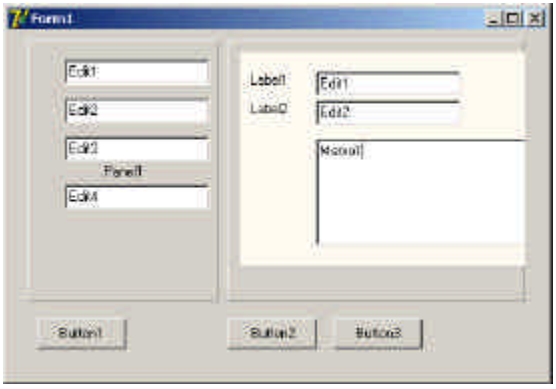
```
procedure TForm1.Button3Click(Sender: TObject);
begin
  // Poisi Form2 diatur agar bisa terlihat
  Form2.Top := 10;
  Form2.Left := 10;
  Form2.Parent := Panel2;
  Form2.Visible := True;
end;
```

- Jalankan program, tekan tombol Button3, maka Form2 akan berada dalam Panel2, seperti ditampilkan pada gambar berikut ini



Setelah form dua berada dalam Panel2, coba kita click ke Edit1. Ternyata fokus tidak dapat masuk ke Edit1. Hal ini dikarenakan ada mekanisme messaging internal yang tidak bekerja. Kalau kita drag header dari Form2, Form ini bisa kita gerakkan. Bila kita tekan Tab beberapa kali (sekitar 5 – 6 kali pada contoh diatas) maka Fokus akan bergerak ke Edit1 dari Form2 dan teks dalam Edit1 tersebut bisa kita ubah.

- Untuk mengatasi masalah diatas, ubah property BorderStyle dari Form2 menjadi bsNone (Form2 tidak memiliki border dan juga header). Hasilnya adalah sebagai berikut



Perlu diberikan catatan disini, mengubah **BorderStyle** menjadi **bsDialog** dari Form2 akan menyebabkan munculnya **Exception** yang berulang-ulang. Bila kita berada pada kondisi seperti ini, untuk

menghentikan program dapat kita gunakan tombol Ctrl-F2 (Program Reset).

Contoh 4-1 Manageable Multi-tabed User Interface

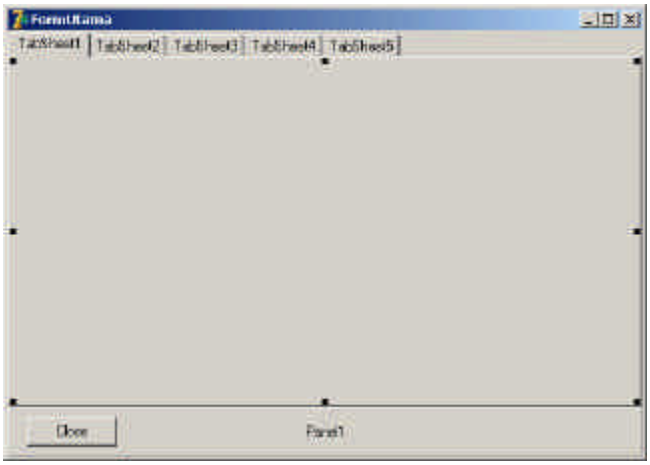
Pada contoh ini kita akan membuat suatu aplikasi yang tampilannya berbentuk multi-tabed. Kita masih menggunakan komponen PageControl yang ada pada page Win32 dalam ComponentPalette. Tetapi berbeda dengan apa yang pernah kita pelajari, pada contoh ini masing masing page adalah sebuah form. Dengan cara seperti ini kode program akan lebih manageable.

Sebagai ilustrasi, misalkan kita akan membuat sebuah aplikasi yang tampilannya hanya satu form, tetapi form tersebut terdiri atas belasan Page. Bila kita masih menggunakan cara biasa, maka kode dan disain untuk setiap page akan berada dalam satu form yang sangat besar (source code-nya mungkin ribuan baris). Program seperti ini sangat tidak manageable. Dan untuk program seperti ini ***maintenance cost***-nya sangat mahal (waktu dan tenaga), serta hampir mustahil dikerjakan lebih dari seorang.

Kita perlu pendekatan lain. Setelah memahami contoh sebelum ini dimana meletakkan sebuah form didalam panel sangatlah mudah dilakukan, secara alamiah solusi yang kita pikirkan adalah dengan memisahkan isi setiap page masing-masing kedalam sebuah Form. Pendekatan seperti inilah yang akan kita lakukan, dan cara melakukannyapun sangat mudah.

Katakanlah Form utama kita terdiri atas 5 page. Kita akan memerlukan 5 buah Form tambahan, jadi seluruhnya kita perlu 6 Form. Form Pertama kita beri nama FormUtama, form lainnya kita beri nama FormPage1 sampai dengan FormPage5.

✍ Buatlah sebuh Form, beri nama form tersebut menjadi FormUtama. Tambahkan sebuah PageControl (dari page Win32). Click kanan PageControl tersebut, pilih tambahkan 5 buah Page kedalamnya. Bila perlu tambahkan sebuah Panel dengan property Align = alBottom dan dengan sebuah tombol diatasnya. PageControl tadi kita set property Align-nya dengan alClient. Tampilannya kurang lebih sebagai berikut



- Buatlah 5 buah form baru, masing masing diberi nama FormPage1 s.d. FormPage5. Isi dan atur setiap form dengan control yang bervariasi sesuai keinginan. Semua form disave dulu agar nanti usesnya tidak bermasalah. Kita simpan form 1 sampai 5 dengan nama FPage1, FPage2, FPage3, FPage4, dan FPage5.
- Kembali ke FormUtama, pada bagian implementation, tambahkan uses untuk mengakses form-form lain, sebagai berikut

```
implementation
uses FPage1, FPage2, FPage3, FPage4, FPage5;
```

- Pada event OnShow dari FormUtama, kita pindahkan semua form dalam Page yang sesuai, dengan kode sebagai berikut.

```
procedure TFormUtama.FormShow(Sender: TObject);
// procedure untuk meletakkan form ke dalam TabSheet
// dibuat agar tidak melakukan pengulangan yang tidak perlu
procedure PlaceFormOnPage(vForm: TForm; vPage: TTabSheet);
begin
  // Kita pastikan border style dari form adalah bsNone
  vForm.BorderStyle := bsNone;
  // Align dari form kita set alClient
  vForm.Align := alClient;
  // Caption dari form kita copy ke Tabsheet
  vPage.Caption := vForm.Caption;
  // Parent kita set ke TabSheet
  vForm.Parent := vPage;
  vForm.Visible := true;
end;
begin
  PlaceFormOnPage(FormPage1, TabSheet1);
  PlaceFormOnPage(FormPage2, TabSheet2);
  PlaceFormOnPage(FormPage3, TabSheet3);
  PlaceFormOnPage(FormPage4, TabSheet4);
  PlaceFormOnPage(FormPage5, TabSheet5);
end;
```

✍ Jalankan program, dan terlihat bahwa semua form sudah masuk ke dalam PageControl.

Catatan untuk Form di dalam Panel
Bila ke dalam panel atau *windowed control* lainnya kita letakkan form yang memiliki menu, maka menu tersebut tidak akan muncul.

Berkenalan dengan TCanvas

Canvas (TCanvas) adalah objek yang mewakili permukaan yang bisa digambari. Kita dapat menggunakan objek ini untuk menggambar diatas semua control yang ada. Objek ini akan dijelaskan secara lengkap pada bab tentang pemrograman grafis. Pada bab ini kita akan mempelajari sedikit tentang objek TCanvas ini karena hal ini akan kita perlukan untuk lebih memahami Visual Component seperti yang akan dijelaskan pada sub-bab setelah sub-bab ini.

Objek TCanvas memiliki beberapa kemampuan, yaitu

- Menggambar garis
- Memenuhi area kotak dengan warna tertentu
- Menggambar beberapa bentuk, seperti kotak, lingkaran, dan polygon
- Menggambar bitmap

? Menuliskan Teks

Menggambar garis

Untuk menggambar garis, kita dapat menggunakan dua method, yaitu **MoveTo** dan **LineTo**. Untuk mengatur warna garis, kita gunakan property **Canvas.Pen.Color**, sedangkan untuk mengatur tebal garis, kita gunakan property **Canvas.Pen.Width**.

```
begin
  Canvas.Pen.Width := 2;
  Canvas.Pen.Color := clRed;
  Canvas.MoveTo(100, 100);
  Canvas.LineTo(120, 200);
end;
```

Memenuhi Area Kotak dengan Warna Tertentu

Kita dapat memenuhi area berbentuk kotak dengan warna tertentu, kita dapat menggunakan method **FillRect**. Warna yang akan kita gunakan dapat kita atur dengan property **Canvas.Brush.Color**.

```
begin
  Canvas.Brush.Color := clYellow;
  Canvas.FillRect(ClientRect);
end;
```

Menggambar Kotak dan Ellipse

Untuk menggambar kotak, kita dapat menggunakan method **Rectangle**, sedangkan untuk menggambar lingkaran ataupun ellipse kita dapat menggunakan method **Ellipse**. Warna garis dari kotak atau ellipse yang akan kita gambar dapat kita atur dengan property **Canvas.Pen.Color**, sedangkan untuk ukuran garis kita gunakan **Canvas.Pen.Width**. Warna isi kotak atau ellipse kita atur dengan property **Canvas.Brush.Color**.

Baik untuk Ellipse maupun Rectangle kita dapat memilih melewatkan dua pasang bilangan integer (X1, Y1, X2, Y2) maupun sebuah variabel bertipe TRect sesuai kebutuhan kita, karena kedua method tersebut adalah method yang di-overload.

```
var vRect: TRect;
begin
  Canvas.Pen.Color := clMaroon;
  Canvas.Brush.Color := clRed;
  vRect := Rect(10, 10, 50, 70);
  Canvas.Rectangle(vRect);
  Canvas.Brush.Color := clAqua;
  Canvas.Rectangle(20, 80, 100, 200);
  Canvas.Brush.Color := clGreen;
  Canvas.Ellipse(200, 100, 300, 200);
  Canvas.Brush.Color := clRed;
  vRect := Rect(200, 220, 300, 360);
  Canvas.Ellipse(vREct);
end;
```

Menggambar Bitmap

Untuk menggambar bitmap, dapat kita gunakan method **Draw**, dengan parameter bitmap yang akan digambarkan ke Canvas. Kode berikut ini akan menggabar bitmap yang sudah ada di Image1 ke Canvas.

```
begin
  Canvas.Draw(0, 200, Image1.Picture.Bitmap);
end;
```

Menulis Teks

Untuk menuliskan teks di Canvas, kita dapat menggunakan TextOut. Untuk mengatur font dari apa yang akan kita tuliskan, kita dapat menggunakan property Font. Kita dapat mengatur nama font dari property Font.Name, warna font dengan Font.Color, serta kita juga dapat menuliskan huruf tebal dan/atau miring dengan Font.Style, sedangkan ukuran font kita gunakan property Font.Size.

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  Canvas.Brush.Color := clAqua;
  Canvas.Font.Name := 'arial';
  Canvas.Font.Style := [fsBold, fsItalic];
  Canvas.Font.Color := clNavy;
  Canvas.Font.Size := 18;
  Canvas.TextOut(120, 20, Edit1.Text);
end;
```

Bekerja dengan Owner Draw

Owner Draw adalah konsep menarik dari Delphi yang memberikan keleluasaan kepada programmer untuk mengubah tampilan dari beberapa control tertentu. Control yang menyediakan kemampuan owner draw menyediakan event yang dapat kita gunakan untuk

menggambari elemen dari control tersebut. Untuk mengetahui apakah sebuah Control memiliki kemampuan owner draw, kita dapat memeriksa apakah ada event yang namanya mirip dengan OnDraw (misalnya **OnDrawItem**, **OnDrawCell**, **OnCustomDraw**, **OnCustomDrawItem**, dll). Control tertentu, seperti ListBox atau ComboBox, dapat memiliki elemen yang ukurannya berbeda. Untuk control seperti ini, biasanya menyediakan event untuk mengatur tinggi setiap item dengan event yang namanya mirip dengan **OnMeasureItem**.

Owner Draw untuk ListBox dan ComboBox

Dalam control ListBox dan ComboBox, terdapat dua event berkaitan dengan owner draw, yaitu OnMeasureItem, untuk mendefinisikan tinggi setiap item sert OnDrawItem untuk menggambar Item. Pada kedua control ini owner draw bisa bekerja jika mengubah property style.

Untuk ListBox, style lbStandard dan lbVirtual adalah style yang bukan owner draw. Bila kita mendefinisikan kode program untuk event OnDrawItem, kode program tersebut tidak akan berpengaruh terhadap tampilan ListBox.

Bila style dari ListBox adalah lbOwnerDrawFixed atau lbVirtualOwnerDraw, maka owner draw akan bekerja. Tetapi listbox dengan style ini ukurannya akan sama untuk semua item, sehingga kode yang kita tulis dalam event OnMeasureItem tidak akan berarti.

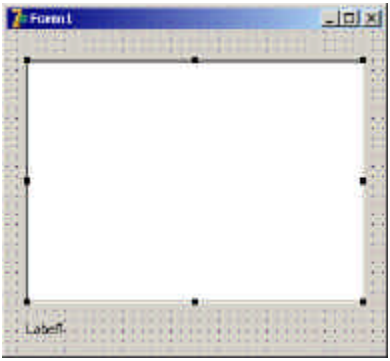
Bila style dari ListBox adalah lbOwnerDrawVariable, maka tinggi setiap item dari ListBox bisa berbeda-beda. Untuk mengaturnya kita dapat menuliskan kode dalam event OnMeasureItem.

Untuk ComboBox, style csDropDown, csSimple, maupun csDropDownList adalah non owner draw. Bila style-nya adalah csOwnerDrawFixed berarti owner draw dengan tinggi elemen yang sama, sedangkan csOwnerDrawVariable adalah owner draw dengan tinggi setiap elemen bervariasi.

Contoh 4-3 Owner Draw Pada ListBox

Contoh ini akan kita gunakan untuk meningkatkan pemahaman kita terhadap mekanisme owner draw pada ListBox. Kita akan mengisi sebuah ListBox dengan daftar nama font yang sudah terinstall di sistem. Selanjutnya setiap item akan ditampilkan dengan menggunakan ukuran dan bentuk font sesuai namanya.

✍ Buat sebuah aplikasi baru, tambahkan sebuah ListBox dan sebuah Label, seperti berikut ini



- Ubah Property ListBox.Style dengan lbOwnerDrawVariable .
- Pada event OnCreate dari Form, kita akan mengisikan property ListBox1.Items dengan daftar nama semua Font yang ada di sistem.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items := Screen.Fonts;
end;
```

- Tulis kode untuk menghitung tinggi setiap item sesuai dengan ukuran asli Font pada event OnMeasureItem dari ListBox1

```
procedure TForm1.ListBox1MeasureItem(Control: TWinControl;
Index: Integer;
  var Height: Integer);
begin
  with ListBox1.Canvas do
  begin
    Font.Name := Listbox1.Items[Index];
    // Gunakan ukuran asli Font
    Font.Size := 0;
    // Hitung tinggi Teks
    Height := TextHeight('Wg') + 2;
  end;
end;
```

- Tuliskan kode untuk menampilkan Font pade event OnDrawItem dari ListBox1, sebagai berikut.

```
procedure TForm1.ListBox1DrawItem(Control: TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState);
begin
  with ListBox1.Canvas do
  begin
    // kita warnai item selang seling antara biru muda
    // dengan cream kecuali bila item terseleksi
    if not (odSelected in State) then begin
      if (index mod 2) = 0 then begin
        Brush.Color := clCream;
      end else begin
        Brush.Color := clSkyBlue;
      end;
    end;
    FillRect(Rect);
    Font.Name := ListBox1.Items[Index];
    Font.Size := 0;    // Gunakan ukuran asli Font
    // Tulis nama font ke canvas
    TextOut(Rect.Left+1, Rect.Top+1, ListBox1.Items[Index]);
  end;
end;
```

Bentrok nama Rect

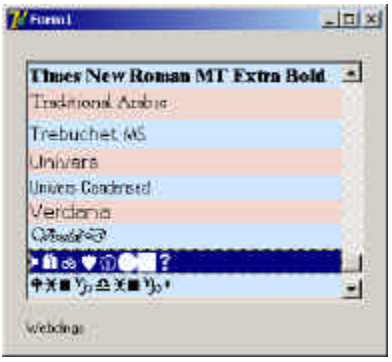
Method untuk event OnDrawItem yang otomatis di-generate oleh Delphi menggunakan sebuah parameter dengan nama Rect dengan tipe TRect. Padahal pada method untuk event seperti ini, seringkali kita perlu memanggil function Rect yang ada di unit Classes. Karena namanya sama, terjadi bentrok nama dan yang berlaku disini adalah variabel Rect. Cara umum untuk mengatasi bentrok nama seperti ini adalah dengan menambahkan nama unit sebagai specifier-nya, yaitu dengan memanggil Classes.Rect pada kasus ini.

Cara lain yang bisa dilakukan untuk kasus ini adalah dengan mengubah nama parameter di bagian definisi method (di bagian atas, yaitu pada deklarasi kelas) dan di bagian implementasinya, sehingga tidak lagi bentrok dengan function Rect. Hal penting yang ingin disampaikan disini adalah bahwa merubah **nama parameter** (asal sama antara deklarasi dengan implementasi) dari method yang di-assign-kan untuk event yang ada dalam Delphi tidaklah menjadi masalah.

✍ Terakhir, kita tampilkan nama Font yang terseleksi dengan event OnClick, sebagai berikut.

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  Label1.Caption := ListBox1.Items[ListBox1.ItemIndex];
end;
```

✍ Setelah selesai, simpan dan jalankan program. Hasilnya adalah sebagai berikut



Owner Draw pada Grid (StringGrid dan DrawGrid)

Menerapkan OwnerDraw pada Grid (StringGrid dan DrawGrid) sedikit lebih mudah, karena ukuran sel Grid adalah seragam, sehingga kita tidak perlu mengatur ukuran setiap sel. Bila yang kita inginkan adalah mengatur lebar setiap kolom, kita dapat menggunakan property **ColWidths** dan bila kita ingin mengatur tinggi setiap baris kita dapat menggunakan property **RowHeights**. Untuk melakukan owner draw, kita dapat menggunakan event OnDrawCell. Grid juga tidak memiliki property khusus yang menentukan apakah owner draw dilakukan atau tidak. Dengan kata lain, bila kita tulis kode pada event OnDrawCell, maka event tersebut pasti dijalankan.

Bekerja dengan Clipboard

Salah satu fasilitas menarik yang disediakan Windows adalah copy data antar aplikasi melalui Clipboard. Delphi menyediakan objek yang membungkus clipboard, yaitu menggunakan objek TClipboard. Aplikasi biasanya mengcopy data ke clipboard dalam beberapa format sekaligus. Misalnya dalam aplikasi teks editor seperti MS Word, ketika kita meng-copy data ke clipboard maka akan ada beberapa format yang bisa kita lihat. Cara termudah untuk melihat format apa saja yang dapat di-paste (misalnya kita menggunakan MS Word) adalah menggunakan menu

Paste Special. Dengan menggunakan Delphi, kita bisa memeriksa ada format apa saja yang dapat kita paste.

Hal lain yang biasanya dilakukan ketika kita bekerja dengan clipboard adalah menangkap notifikasi dari objek Clipboard untuk mengetahui apakah ada data yang baru di-copy ke clipboard.

Objek TClipboard dan function Clipboard

Delphi menyediakan function Clipboard yang hasilnya bertipe TClipboard untuk mengakses objek clipboard dari Windows. TClipboard adalah sebuah objek yang membungkus fungsionalitas clipboard sehingga memudahkan kita dalam bekerja dengan clipboard.

Meng-copy dan mem-paste text

Bekerja dengan clipboard menggunakan format text amat sangat mudah. Kita cukup menggunakan property AsText dari clipboard. Meng-assign property AsText sama dengan meng-copy ke clipboard. Sedangkan mengassignkan property AsText ke variabel tertentu sama dengan mem-paste dari clipboard. Bila tidak ada informasi bertipe text dalam clipboard, maka AsText akan berisi string kosong. Contoh penggunaannya adalah seperti berikut ini

```
Clipboard.AsText := Edit1.Text; //Copy

Edit2.Text := Clipboard.AsText; // Paste
```

Meng-copy dan mem-paste Gambar

Selain copy-paste teks, objek TClipboard juga sudah menyediakan copy-paste gambar asal berformat bitmap atau metafile. Caranya juga sangat mudah, tinggal menggunakan method Assign. Contoh penggunaannya adalah seperti berikut ini

```
// copy to clipboard
// bila isi Picture bukan bitmap atau metafile
// akan terjadi exception
Clipboard.Assign(Image1.Picture);

// Paste from Clipboard
Image1.Picture.Assign(Clipboard);

// copy to clipboard (langsung dari bitmap)
// MyBitmap bertipe TBitmap
Clipboard.Assign(MyBitmap);
```

Method CopyToClipboard dan PasteFromClipboard

Beberapa komponen sudah menyediakan mekanisme copy dan paste ke clipboard. Untuk copy menggunakan method CopyToClipboard, sedangkan untuk paste menggunakan method PasteFromClipboard. Komponen-komponen tersebut adalah

- TCustomEdit dan TCustomMemo beserta turunannya seperti TEdit, TMemo, TRichEdit, dan TMaskEdit.
- TDBImage
- TddeServerItem
- Turunan dari TGraphic, seperti TBitmap, dan TMetafile, menggunakan method yang bernama SaveToClipboardFormat dan LoadFromClipboardFormat.

Memeriksa Format-format dalam Clipboard

Kita dapat memeriksa format-format apa saja yang ada dalam Clipboard dengan menggunakan property yang dimiliki objek Clipboard yaitu Formats dan FormatCount. FormatCount adalah banyaknya format yang ada dalam clipboard, sedangkan Formats adalah sebuah property berindeks bertipe Word yang merupakan kode format yang ada dalam Clipboard. Untuk mengetahui nama format, kita dapat menggunakan function dari Windows API yang bernama GetClipboardFormatName. Sayangnya function ini tidak memberikan nilai bila format yang ada dalam clipboard adalah default format. Oleh sebab itu kita perlu meng-case setiap clpboard format.

Peter Below dari TeamB menunjukkan bagaimana cara memeriksa format data yang ada di clipboard dan menunjukkan arti setiap kode format standard yang ada dalam Windows. Kodenya ditulis dalam Delphi dan ada pada link ini.

http://homepages.borland.com/efg2lab/Library/UseNet/2001/1124.txt

Berikut ini adalah sedikit modifikasi dari library yang ditulis oleh Peter Below. Modifikasi dilakukan agar program lebih sederhana.

```
function GetDefaultFormat(vFormatID: Cardinal): string;
begin
  case vFormatID of
        1: result := 'CF_TEXT';
        2: result := 'CF_BITMAP';
        3: result := 'CF_METAFILEPICT';
        4: result := 'CF_SYLK';
        5: result := 'CF_DIF';
        6: result := 'CF_TIFF';
        7: result := 'CF_OEMTEXT';
        8: result := 'CF_DIB';
        9: result := 'CF_PALETTE';
       10: result := 'CF_PENDATA';
       11: result := 'CF_RIFF';
       12: result := 'CF_WAVE';
       13: result := 'CF_UNICODETEXT';
       14: result := 'CF_ENHMETAFILE';
       15: result := 'CF_HDROP (Win 95)';
       16: result := 'CF_LOCALE (Win 95)';
       17: result := 'CF_MAX (Win 95)';
    $0080: result := 'CF_OWNERDISPLAY';
    $0081: result := 'CF_DSPTEXT';
    $0082: result := 'CF_DSPBITMAP';
    $0083: result := 'CF_DSPMETAFILEPICT';
    $008E: result := 'CF_DSPENHMETAFILE';
    $0200..$02FF: result := 'private format';
    $0300..$03FF: result := 'GDI object';
  else
    result := 'unknown format';
  end;
end;
```

```
function FormatIDToString(vFormatID: Cardinal): string;
var vBuffer: array[0..60] of Char;
begin
  FillChar(vBuffer, SizeOf(vBuffer), 0);
  if GetClipboardFormatName(vFormatID, vBuffer,
    SizeOf(vBuffer))<>0 then begin
    result := vBuffer;
  end else begin
    result := GetDefaultFormat(vFormatID);
  end;
end;
```

Contoh 4-4 Memeriksa format dalam Clipboard

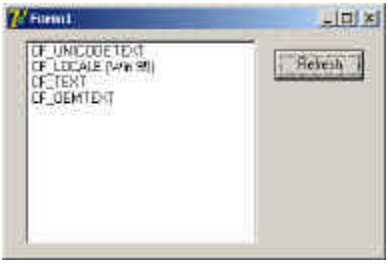
Pada contoh ini kita akan membuat aplikasi yang dapat memeriksa format-format apa saja yang ada dalam clipboard.

✍ Buat sebuah aplikasi baru, tambahkan sebuah Listbox dan sebuah tombol dengan caption Refresh.

- Bila tombol refresh
- Tambahkan Clipbrd ke uses list
- Ketikkan function FormatIDToString dan function GetDefaultFormat diatas.
- Ketikkan response untuk event OnClick dari tombol refresh diatas sebagai berikut

```
procedure TForm1.Button1Click(Sender: TObject);
var i: Integer;
begin
  ListBox1.Items.Clear;
  for i:=0 to Clipboard.FormatCount-1 do begin
    ListBox1.Items.Add(FormatIDToString(
      Clipboard.Formats[i]));
  end;
end;
```

- Jalankan Aplikasi. Jalankan aplikasi lain (misalnya Notepad), lakukan copy dari aplikasi tersebut.
- Pada aplikasi kita, tekan tombol refresh. Tampilannya kurang lebih sebagai berikut



Menangkap Notifikasi dari Clipboard

Suatu objek window (di Delphi adalah turunan dari TWinControl) bisa menerima notifikasi dari Clipboard jika didaftarkan sebagai suatu Clipboard Viewer, yang berkerja dengan dua buah Windows API function dan beberapa message untuk berkomunikasi dengan Clipbord.

Function SetClipboardViewer digunakan untuk mendaftarkan diri di awal rantai clipboard viewer dan menghasilkan suatu handle untuk clipboard viewer berikutnya.

Contoh 4-5 Menangkap Notifikasi dari Clipboard

Contoh ini masih melanjutkan contoh sebelumnya. Bila ada salah satu aplikasi di windows, kita akan me-refresh Listbox yang ada dalam Form untuk menampilkan semua format clipboard yang ada.

- Masih pada aplikasi yang sama dengan sebelumnya, tambahkan sebuah field dari objek TForm1 dengan nama NextInChain yang bertipe THandle.

- Tambahkan dua buah function yang bernama WMDrawClipboard dan WMChangeCBChain.

```
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    NextInChain: THandle;
    procedure WMDrawClipboard(var Msg: TMessage);
      message WM_DRAWCLIPBOARD;
    procedure WMChangeCBChain(var Msg: TMessage);
      message WM_CHANGECBCHAIN;
...
```

- Pada saat form di-create, kita daftarkan form tersebut untuk memperoleh notifikasi dari clipboard dengan kode seperti berikut ini

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  NextInChain := SetClipboardViewer(Handle);
end;
```

Bila ada perubahan pada clipboard, maka form yang kita buat akan menerima message dengan nomor message adalah WM_DRAWCLIPBOARD. Message tersebut akan kita respon dengan me-refresh ListBox dan kita harus melanjutkan message ke mata rantai clipboard viewer berikutnya.

- Ketikkan implementasi procedure WMDrawClipboard berikut ini

```
procedure TForm1.WMDrawClipboard(var Msg: TMessage);
begin
  Button1Click(Button1);
  if NextInChain <> 0 then begin
    SendMessage(NextInChain, WM_DrawClipboard, 0, 0)
  end;
end;
```

Selanjutnya kita harus mengimplementasi procedure yang menangani messge WM_CHANGECBCHAIN. Ketika ada message

datang, kita harus periksa apakah handle yang akan di remove (WParam) adalah NextInChain. Bila ya maka NextInChain harus diisi dengan next (LParam). Bila handle yang diremove bukan NextInChain, kita harus melanjutkan message ini ke handle NextInChain.

✍ Ketikkan kodenya seperti berikut ini

```
procedure TForm1.WMChangeCBChain(var Msg: TMessage);
var vRemove, vNext: THandle;
begin
  vRemove := Msg.WParam;
  vNext := Msg.LParam;
  if NextInChain=vRemove then begin
    NextInChain := vNext;
  end else if NextInChain <> 0 then begin
    SendMessage(NextInChain, WM_CHANGECBCHAIN,
      vRemove, vNext);
  end;
end;
```

✍ Jalankan program dan lakukan copy dari software lain, maka secara otomatis Listbox akan di-refresh.