【A4】Delphiテクニカルセッション

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

開発効率を飛躍的に高めるコンポーネント自作テクニック

（株）シリアルゲームズ
細川　淳

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

コンポーネントとは？

コンポーネントとは

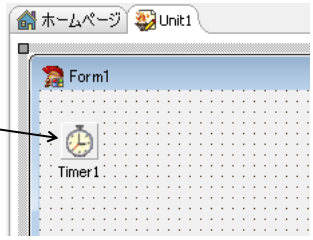
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- コンポーネントとは
  - プログラムに使える部品
    - 1つの機能を表した**再利用可能**なプログラム
      - 単体ではなく、他のプログラムやコンポーネントと組み合わせて使う
      - 例えば、TMemo を TForm に乗せて使うなど
    - Delphi では特にデザイナでドロップできる物を指す
      - TComponent が継承元のもの
      - 例えば、TStringList や TFileStream などはコンポーネントとは呼ばない
  - Delphi では、次の2つに大別される
    - ビジュアルコンポーネント
      - フォームにドロップしたとき、自身が描画されるコンポーネント
    - 非ビジュアルコンポーネント
      - フォームにドロップしたとき、アイコンが表示されるコンポーネント

Delphi のコンポーネント(1)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

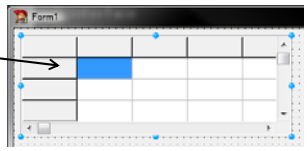
- 非ビジュアルコンポーネント
  - ドロップするとアイコンが表示される
    - 例えば TTimer
  - 見た目はない
  - 継承元は基本的に TComponent



Delphi のコンポーネント（2）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ビジュアルコンポーネント
  - ドロップすると、コンポーネントの実行時と同じ見た目が表示される
    - 例えば TStringGrid



  - 見た目がある
  - 継承元は、いくつかある
    - TControl
      - 全てのコントロールの継承元
    - TGraphicControl
      - ユーザーの操作が必要ないコントロールの継承元
        - » TLabel など
    - TWinControl
      - Windows のコントロールの継承元（ハンドルを持つ）
        - » Windows コントロールである TMemo など
    - TCustomControl
      - 一般的なコントロールの継承元
        - » TDrawGrid など

コントロールとは

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- コントロール
  - ユーザーとやり取りできるコンポーネント
    - ビジュアルコンポーネントであることが必須
- TControl - 全てのコントロールの先祖
  - TControl
    - TGraphicControl
    - TWinControl
      - TCustomControl
        - » 実際に使われるコントロール

EMBARCADERO TECHNOLOGIES

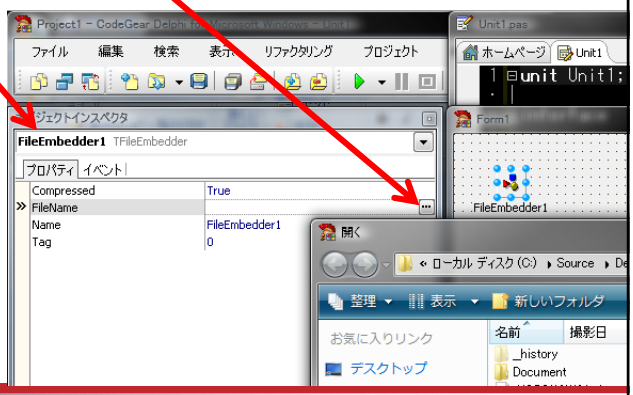
DEVELOPER CAMP

非ビジュアルコンポーネントの作成

非ビジュアル系コンポーネントの作成

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 実際に非ビジュアルコンポーネントを作成してみよう
  - 機能
    - ファイルを埋め込む
    - 埋め込むファイルを選択できる
  - 名前
    - TFileEmbedder



TFileEmbedder の実装(1)ー動作の策定

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 必要な機能
  - ファイルの指定
    - →ファイルダイアログを開いて指定させたい
  - ファイルの中身を保存する
    - →指定されたファイルを読み込んで保存しておきたい

  - 流れ
    ファイルプロパティの横のボタン(・・・)を押す
    ↓
    ファイルダイアログでファイルを開く
    ↓
    ファイルを読み込む
    ↓
    ファイルを保存する

TFileEmbedder の実装(2)ークラス定義

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
unit uFileEmbedder;

interface

uses
  Windows, Classes;

type
  TFileEmbedder = class(TComponent)
  private
    // Variables
    FFileName: String;
    FData: TMemoryStream;
    FCompressed: Boolean;
    // Methods
    procedure ReadData(vStream: TStream);
    procedure WriteData(vStream: TStream);
    procedure SetFileName(const Value: String);
  protected
    // Methods
    procedure DefineProperties(vFiler: TFiler); override;
  public
    // Constructor & Destructor
    constructor Create(vOwner: TComponent); override;
    destructor Destroy; override;
    // Methods
    procedure SaveToFile(const vFileName: String);
    procedure SaveToStream(const vStream: TStream);
    // Property
    property Data: TMemoryStream read FData;
  published
    // Properties
    property FileName: String read FFileName write SetFileName;
    property Compressed: Boolean
      read FCompressed write FCompressed default True;
  end;
```

非ビジュアルコンポーネントなので TComponent から継承する

ファイルの中身を保持

格納するファイルの名前

published(黄枠内) の
プロパティが
オブジェクトインスペクタに
表示される

TFileEmbedder の実装(3)ー方法

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- published のプロパティは自動的に保存される
  - 今回の FileName プロパティについては、何もする必要がない

※ただし、一般的な型のデータに限る

- ファイルの保存を考える
  - published プロパティではない Data プロパティを保存するには？

→TComponent のメソッド「DefineProperties」を利用する

- 「DefineProperties メソッドは，オブジェクトの非 published データをフォームファイルなどのストリームに格納するためのメソッドを指定します。」(ヘルプより)

定義

```
procedure DefineProperties(Filer: TFiler); override;
```


TFileEmbedder の実装(4)ーDefineProperties

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- DefineProperties の使い方
  - **定義：procedure** DefineProperties(Filer: TFiler);
  - DefineProperties は、自動的に呼ばれる
  - コンポーネント固有のデータを格納するための「コンポーネントストリーム」にアクセスする方法が、引数の Filer により提供される
- TFiler の使い方ー提供されるメソッド
  - DefineProperty　　Integer や String といった一般的な型のデータ
    - → TWriter.WriteString, TReader.ReadInteger などが使える場合
  - DefineBinaryProperty　　画像などの一般的ではないデータ
    - → TStream.Read, Write として読み書きする場合

TFileEmbedder の実装(5)ーDefinePropertiesの実装

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure TFileEmbedder.DefineProperties(vFiler: TFiler);
begin
  inherited;
```
必ず inherited する

今回は、ファイルの中身がどんなものでも平気なようにDefineBinaryProperty を使用する

```
  vFiler.DefineBinaryProperty(
    'EmbededData',
    ReadData,
    WriteData,
    (FFileName <> ''));
end;
```

保存するデータの名前を指定する(疑似プロパティ名)

保存したデータを読み出す時に呼ばれるメソッドを指定する

データを保存時に呼ばれるメソッドを指定する

実際に保存したいかどうか。ファイルが未指定名場合は保存しない

第2、3引数の型は TStreamProc

```
type TStreamProc = procedure(Stream: TStream) of object;
```

DefineProperty の場合、第2，3引数の型が違い、TReaderProc, TWriterProc となる

```
type TReaderProc = procedure(Reader: TReader) of object;
type TWriterProc = procedure(Writer: TWriter) of object;
```


TFileEmbedder の実装(6)ー WriteData

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure TFileEmbedder.WriteData(vStream: TStream);
var
  Size: Integer;
  Buff: Pointer;
  CompressedBuf: Pointer;
  CompressedSize: Integer;

  procedure WriteO;
  begin
    Size := 0;
    vStream.Write(Size, SizeOf(Integer));
  end;

begin
  FData.Clear;

  if (not FileExists(FFileName)) then begin
    WriteO;
    Exit;
  end;

  FData.LoadFromFile(FFileName);

  Size := FData.Size;

  if (Size = 0) then begin
    WriteO;
    Exit;
  end;

  if (FCompressed) then begin
    Buff := FData.Memory;

    ZCompress(Buff, Size, CompressedBuf, CompressedSize);
    try
      vStream.Write(CompressedSize, SizeOf(Integer));
      vStream.Write(PByte(CompressedBuf)^, CompressedSize);
    finally
      FreeMem(CompressedBuf);
    end;
  end
  else begin
    vStream.Write(Size, SizeOf(Integer));
    vStream.CopyFrom(FData, 0);
  end;
end;
```

グレーアウト部分はファイルの圧縮処理。
コンポーネントの説明とはあまり関係がないためグレーアウトとした

- vStream に対してデータを保存する

TFileEmbedder の実装(7)—ReadData

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure TFileEmbedder.ReadData(vStream: TStream);
var
  Size: Integer;
  Buff: Pointer;
  DecompressedBuf: Pointer;
  DecompressedSize: Integer;
begin
  vStream.Read(Size, SizeOf(Integer));

  FData.Clear;

  if (FCompressed) then begin
    GetMem(Buff, Size);
    try
      vStream.Read(PByte(Buff)^, Size);

      ZDecompress(Buff, Size, DecompressedBuf, DecompressedSize, 0);
      try
        FData.Size := DecompressedSize;

        CopyMemory(FData.Memory, DecompressedBuf, DecompressedSize);
      finally
        FreeMem(DecompressedBuf);
      end;
    finally
      FreeMem(Buff);
    end;
  end
  else
    FData.CopyFrom(vStream, Size);

  FData.Position := 0;
end;
```

グレーアウト部分はファイルの解凍処理。コンポーネントの説明とはあまり関係がないためグレーアウトとした

- vStream からデータを読み込む

TFileEmbedder の実装(7)—コンポーネントの登録

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
unit uFileEmbedder;

interface

uses
  Windows, Classes;

type
  TFileEmbedder = class(TComponent)
    :
    :
  end;

procedure Register;

implementation

uses
  SysUtils, ZlibEx, uMessageDialog;

procedure Register;
begin
  RegisterComponents('Sample', [TFileEmbedder]);
end;
```

- コンポーネントのインストール時に interface 部で定義されている「Register」という名前の関数が自動的に呼ばれる
- RegisterComponents でコンポーネントを登録する
  RegisterComponents('パレットの名前', [登録するコンポーネント]);

TFileEmbedder の実装(8)ーまとめ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- published プロパティは自動的に保存される
- それ以外のデータはコンポーネントストリームに格納する
  プロパティ格納用の主要な3つのメソッド
  - TComponent.DefineProperties
  - TFiler.DefineProperties
  - TFiler.DefineBinaryProperties
- コンポーネントの登録には下記の関数を使う
  - Register　　コンポーネントインストール時呼ばれる
  - RegisterComponents　　Register の中で呼び
    コンポーネントを登録する

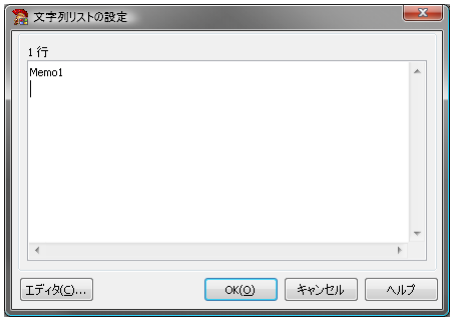
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

プロパティエディタの作成

プロパティエディタとは？

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

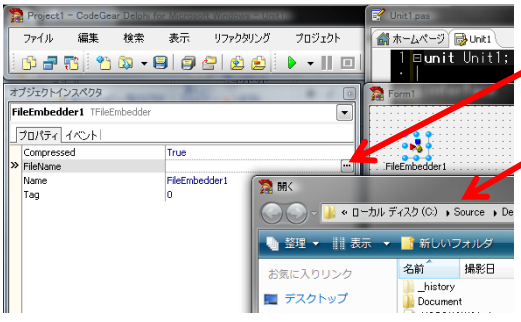
- オブジェクトインスペクタから呼び出されるプロパティの編集用エディタのこと
  - 例えば TMemo で言えば Lines を押したときに出てくるエディタ



TFileEmbedderEditor の実装（1）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 今回は、ボタンが押されるとファイルダイアログを開くようにしたい



オブジェクトインスペクタの
このボタンが押されたら
ファイルダイアログを開きたい

TFileEmbedderEditor の実装(2)—TPropertyEditor

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
unit uFileEmbedderEditor;   コンポーネントとは、別のユニット！

interface
                            DesignIntf, DesignEditors には、
                            デザイン時に必要なインターフェース、プロパティエディタクラスなどが宣言されている
uses
  DesignIntf, DesignEditors;

type
  TFileEmbedderEditor = class(TStringProperty)
  public
    procedure Edit; override;
    function GetAttributes: TPropertyAttributes; override;
  end;
```

- プロパティエディタは TPropertyEditor の下位クラスから継承する
  - TIntegerProperty や TMethodProperty など
  - 今回は文字列(ファイル名)のため TStringProperty から継承する
  - 下位クラスから継承すると、プロパティの保存を自分で行う必要がない
  - そのため、今回は Edit メソッドと GetAttributes メソッドを実装するだけで良い
- プロパティエディタは、本体のコンポーネントとソースファイルを必ずわけること
  - わけないと、実行ファイルに不必要なプロパティエディタやユニットが含まれてしまう

TFileEmbedderEditor の実装(3)—Edit

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure TFileEmbedderEditor.Edit;
begin
  with TOpenDialog.Create(Application) do
    try
      FileName := GetStrValue;   GetStrValue でファイル名を TOpenDialog に設定する
      Filter := 'すべてのファイル (*.*)|*.*';

      if (Execute) then
        SetStrValue(FileName);   SetStrValue でファイル名をプロパティの値とする
    finally
      Free;
    end;
end;
```

- TPropertyEditor.Edit メソッドは「・・・」ボタンを押されたとき呼ばれる
  - ここで、必要ならば自作のフォームを開くこともできる
  - 今回は TOpenDialog を開き、ファイル名を取得・設定している
- GetStrValue, SetStrValue
  - この2つのメソッドは TStringProperty が提供するメソッドでプロパティの値を取得・設定できる

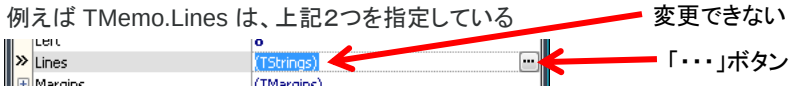
TFileEmbedderEditor の実装(4)—GetAttributes

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
function TFileEmbedderEditor.GetAttributes: TPropertyAttributes;
begin
  Result := [paDialog];
end;
```

- TPropertyEditor.GetAttributes はプロパティの属性を返す

  代表的な属性
  - paDialog　　「・・・」ボタンを表示し Edit メソッドを呼び出す
  - paReadOnly　　値は変更できない
    - 例えば TMemo.Lines は、上記2つを指定している



    - 他には、paValueList, paSortList, paSubProperties, paMultiSelect, paAutoUpdate, paRevertable, paFullWidthName, paVolatileSubProperties, paVCL, paNotNestable がある

TFileEmbedderEditor の実装(5)—登録

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure Register;

implementation

uses
  Dialogs, uFileEmbedder;

procedure Register;
begin
  RegisterPropertyEditor(
    TypeInfo(String),      このプロパティの型情報を渡す
    TFileEmbedder,         プロパティエディタと関連づけるクラス
    'FileName',            プロパティの名前
    TFileEmbedderEditor);  プロパティエディタのクラス
end;
```

- コンポーネントと同じように Register 関数が呼ばれる
- RegisterPropertyEditor 関数でプロパティエディタを登録する

TFileEmbedderEditorの 実装（6）ーまとめ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 編集したいプロパティの型にあったプロパティエディタクラスから派生する
- プロパティの属性を適切に設定する
  - フォームを表示したいならば paDialog を指定する
- paDialog なら Edit メソッドをオーバーライドしてフォームを表示する
- Register 関数で RegisterPropertyEditor 関数を呼び、プロパティエディタとして設定する

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

ビジュアルコンポーネントの作成

ビジュアルコンポーネントの作成

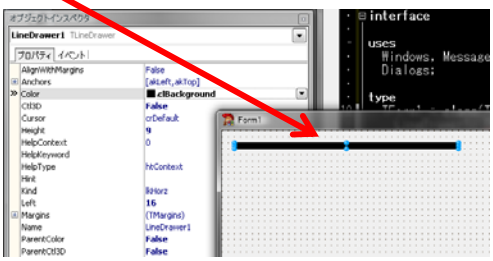
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 基本的には非ビジュアルコンポーネントと同じ
- 違うところは継承元
  - ユーザーの操作が
    - 必要なければ TGraphicControl
    - 必要であれば TCustomControl
      - ただし、ユーザー操作が必要なくても独自の Canvas が必要な場合は TCustomControl を使う
      - TGraphicControl は、Parent のデバイスコンテキストを使用して描画されるため Parent の背景に影響してしまう
  - を使う
- 見た目を表示するために TControl.Paint が呼ばれる
  - この中で Canvas を使ってコンポーネントの外観を描く

ビジュアルコンポーネントの作成

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 実際にビジュアルコンポーネントを作成してみよう
  - 機能
    - 線を引く
  - 名前
    - TLineDrawer



TLineDrawer の実装(1)ー動作の策定

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 必要な機能
  - 線を描く
  - 流れ
    Paint が呼ばれたら線を描く

TLineDrawer の実装(2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
unit uLineDrawer;

interface

uses
  Messages, Classes, Controls;

type
  TLineDrawerKind = (lkHorz, lkVert);

  TLineDrawer = class(TCustomControl)
  private
    FKind: TLineDrawerKind;
    procedure SetKind(const Value: TLineDrawerKind);
  protected
    // Methods
    procedure Paint; override;
    // Message Handlers
    procedure CMCtl3DChanged(var vMsg: TMessage); message CM_CTL3DCHANGED;
  public
    // Constructor
    constructor Create(vOwner: TComponent); override;
  published
    // Properties
    property Kind: TLineDrawerKind read FKind write SetKind default lkHorz;
    // Inherited Properties
    property Anchors;
    property Color;
    property Ctl3D;
    property ParentColor;
    property ParentCtl3D;
    property Visible;
    property OnClick;
    property OnDblClick;
  end;
```

ビジュアルコンポーネントで
独自 Canvas を使用するため
TCustomControl から派生する

Paint メソッドをオーバーライド
して自身を描画する

CM_CTL3DCHANGED
コントロールメッセージに
応答する
(付録A参照)

property に default を付けると
そのプロパティのデフォルト値に
できる
(値が設定されるわけではない)

継承元の protected プロパティを
published で再宣言すると
オブジェクトインスペクタに表示
できるようになる

TLineDrawer の実装(3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
procedure TLineDrawer.Paint;
var
  Half: Integer;
  Dark, Light: TColor;
begin
  with Canvas do begin
    Brush.Style := bsSolid;
    Dark := Darker(Color);
    Light := Lighter(Color);

    if (Ctl3D) then
      case FKind of
        lkHorz: begin
          Half := Height div 2;

          Brush.Color := Dark;
          FillRect(Rect(0, 0, Width, Half));

          Brush.Color := Light;
          FillRect(Rect(0, Half, Width, Height));
        end;
```

Ctl3D の値で描画する線の形状を変更する

水平線を3Dっぽく描く

```
        lkVert: begin
          Half := Width div 2;

          Brush.Color := Dark;
          FillRect(Rect(0, 0, Half, Height));

          Brush.Color := Light;
          FillRect(Rect(Half, 0, Width, Height));
        end;
      end
    else begin
      Brush.Color := Color;
      FillRect(ClientRect);
    end;
  end;
end;
```

垂直線を3Dっぽく描く

3Dじゃなければ、コントロール矩形いっぱいに四角を描く

- Paint メソッドで Canvas プロパティを使って自身を描画する

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

コンポーネントの登録

アイコンの作成

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Bitmap 16x16 で作成する
(上部にコンポーネントパレットがあった時代は 24x24 だった)

画像の左下と同じ色が透過色となる

XN Resource Editor

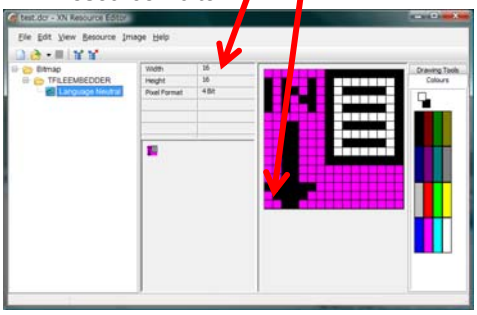
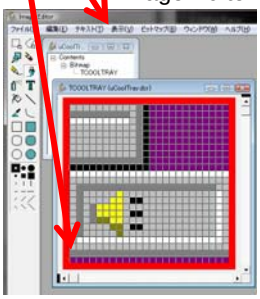


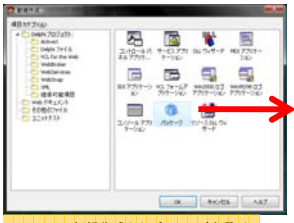
Image Editor



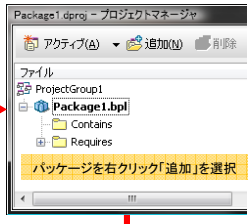
- XN Resource Editor や 旧Borland Image Editor などを使い Delphi Component Resource (dcr) ファイルを作成する
  - 現在 Image Editor は提供されていないため、コンパニオンCDに入っている XN Resource Editor を使用する
- dcr が無い場合はデフォルトのアイコンで表示される

新規パッケージの作成と登録

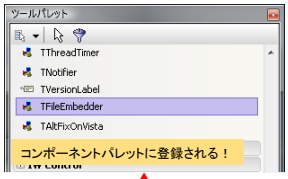
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



ファイル→新規作成からパッケージを選ぶ

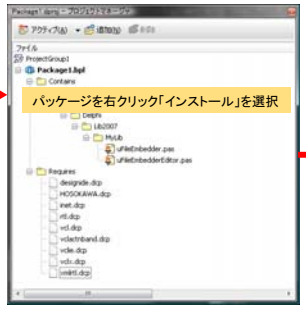


パッケージを右クリック「追加」を選択



コンポーネントパレットに登録される！

追加するコンポーネントファイルを選択。
コンポーネント自身・コンポーネントエディタ・アイコン(dcr)を選ぶ



パッケージを右クリック「インストール」を選択

インストール後にエラーがあったら？

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- インストールしたパッケージ内のコンポーネントにバグがあり、読み込み違反などを起こした場合、Delphi 自身の実行に支障がでてしまう。
- しかし、パッケージは起動時に読み込まれるため、そのままでは Delphi を正常に使用することができない。

- そう言った場合は、以下のようにする
  1. 一旦 Delphi を終了する
  2. 当該パッケージを削除する（名前を変更する）
  3. Delphi を起動する
     - 起動時にエラーが表示され「次回は読み込まないようにするか？」と聞かれるが、これに YES と答えると、存在しても読み込まれなくなってしまうため、必ず NO とする。
  4. 修正する
  5. 再度インストールする

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

付録A　～その他の機能～

プロパティのデフォルト値

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
TLineDrawer = class(TCustomControl)
  :
  :
published
  property Kind: TLineDrawerKind read FKind write SetKind default lkHorz;
  :
  :
end;
```

- TLineDrawer の Kind プロパティにはデフォルト値が設定されている
- published のプロパティにデフォルト値を設定すると、プロパティの値がデフォルト値と同じ場合、コンポーネントストリームに保存されない。
- 例えば TLineDrawer.Kind が lkHorz だった場合、値は保存されない
- 値がストリームに保存されて居なくても、コンポーネント生成時に値が設定される（はず）からである
- これによって、容量を削減できる
- コンポーネントの作成者は、デフォルト値を指定した場合、コンストラクタなどで値を指定する必要がある
  - TLineDrawer.Kind のデフォルト値は lkHorz (=0) なので、保存していない
  - これが lkVert だった場合はコンストラクタや Loaded メソッドで指定する必要がある
    - **TComponent.Loaded メソッド**は、コンポーネントの全てのプロパティやデータが読み終わったあとに呼ばれるメソッド。
    - あるプロパティAが、他のプロパティBに依存しているときに、プロパティAの初期化に使うことができる

状態や属性を示すプロパティ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TComponent.ComponentState
  - コンポーネントの現在の状態を示すフラグ
  - csDesigning　　デザイン時
  - csDestroying　　コンポーネントが破棄中である
- TControl.ControlState
  - 実行時にコントロールの現在の状態を示すフラグ
  - csLButtonDown　　マウスの左ボタンが押されている
  - csCreating　　現在コントロールを作成中
- TControl.ControlStyle
  - コントロールの属性を示すフラグ
  - csCaptureMouse　　マウスをキャプチャする
このフラグをセットすると左ボタンダウンで自動的にキャプチャされる
  - csSetCaption　　コントロールの Caption は明示的に指定されない限り
Name プロパティと同じ値になる

※各プロパティの値は、代表的な物のみ記した
その他の値は Classes.pas, Controls.pas を参照

その他のメソッド・関数

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TComponent.Notification メソッド
  - コンポーネントが追加・削除されたときに自動的に呼び出されるメソッド
    - 定義:

```
procedure TComponent.Notification(Component: TComponent; Operation: TOperation);
type TOperation = (opInsert, opRemove);
```

  - 例えば、Form 上に載っている TMenu のインスタンスが削除されると、Form が所有しているコンポーネントの Notification を呼び出す。これによって、TMenu のインスタンスを参照しているコンポーネントがあった場合、適切な処理(参照を nil にするなど)を行うことができる。

- TComponent.Loaded メソッド
  - コンポーネントストリームから全てのプロパティやデータを読み終わると呼ばれるメソッド。
    - 定義:

```
procedure TComponent.Loaded;
```

  - あるプロパティAが、他のプロパティBに依存しているときに、プロパティAの初期化に使うことができる

- RegisterNoIcon 関数
  - コンポーネントを登録するが、ツールパレットには登録しない場合に使う
  - 例えば TMenuItem は TMenu, TPopupMenu で作成できるが、ツールパレット上にアイコンは存在しない
  - コンポーネント登録をしておかないとコンポーネントストリームからの読み込み時に失敗してしまう。

※メソッドは、派生先コンポーネントで override して使う

コントロールメッセージ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- コントロールメッセージとは何らかの事態が起きたときにコントロールに通知されるメッセージである
  - デザインだけではなく、実行時も通知される。
  - これらに反応することで見た目の変化などに対応できる

- 例えば下記のような物がある
  - CM_FONTCHANGED　フォントが変更された
  - CM_MOUSEENTER　マウスがコントロールの中に入った
  - CM_MOUSELEAVE　マウスがコントロールの中から出た
  - CM_CTL3DCHANGED　3D状態が変更された
  - CM_COLORCHANGED　色が変更された

※Controls.pas に全メッセージが定義されている

コンポーネントの名前・可視性

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TCustom～
  - コンポーネントの先頭に Custom と付いているコンポーネントは継承元クラスとしてデザインされている
    - 例えば TCustomEdit から TEdit や TMemo が派生している
  - 何故 Custom～ というコンポーネントが必要なのか？

→ プロパティやメソッドの可視性を変更するため

例えば TLineDrawer では Ctr3D などを published にしていた
これらの絶対必要ではないプロパティは TCustomControl では
全て protected レベルに定義されている
protected に定義することで、派生コンポーネントは、必要なプロパティだけ公開できる

- 自分でコンポーネントを作るときも、この規則に沿うと良い

コンポーネントは機能の集まりであり、自分だけが使うわけではない
そのため、規則に沿った方が他人にもわかりやすい

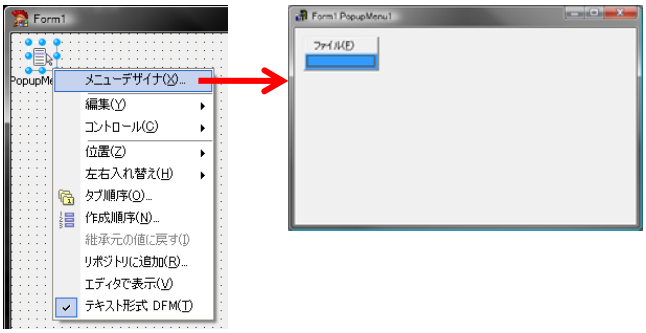
- 可視性
  - published レベルで定義されたプロパティには実行時型情報が生成される.
  - strict private, strict protected が Delphi 8 で追加された
    - 同じユニット内からの可視性を制御出来るようになった

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

付録B　～コンポーネントエディタ～

コンポーネントエディターTComponentEditor

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



- 上図のようにコンポーネントを右クリックしたとき独自メニューを表示し、何らかの機能を実行するにはどうすればよいのだろうか？

→TComponentEditor を使うと TMenu を右クリックしたときのような動作を実装できる

コンポーネントエディタの例

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

```
  TTitleEditor = class(TComponentEditor)
  public
    procedure Edit; override;
    function GetVerb(vIndex: Integer): String; override;
    function GetVerbCount: Integer; override;
    procedure ExecuteVerb(vIndex: Integer); override;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponentEditor(TTitleBar, TTitleEditor);
end;
```

- GetVerb
  - コンテキストメニューに表示する文字列を返す
- GetVerbCount
  - コンテキストメニューの数を返す
- ExecuteVerb
  - メニューがクリックされると呼ばれる
- Edit
  - PropertyEditor と同じでここでダイアログを表示したりする

```
function TTitleEditor.GetVerb(vIndex: Integer): String;
begin
  case vIndex of
    0:
      Result := 'Title の編集';
    else
      Result := '';
  end;
end;

function TTitleEditor.GetVerbCount: Integer;
begin
  Result := 1;
end;

procedure TTitleEditor.ExecuteVerb(vIndex: Integer);
begin
  case vIndex of
    0:
      Edit;
  end;
end;

procedure TTitleEditor.Edit;
begin
  if (FTitleEditor = nil) then
    FTitleEditor := TfrmTitleEditor.Create(nil);

  FTitleEditor.Designer := Designer;
  FTitleEditor.Initialize(Component);
  FTitleEditor.Show;
end;
```

コンテキストメニューの一番上(0番目)に 'Title の編集' と表示する

メニューに表示する数を返す

独自のエディットフォームを表示する

TComponentEditor.Designer インターフェースを渡している
IDesigner にはデザイン時に必要な様々なメソッドが定義されている
例えば Modified メソッドを呼べば変更されたことになり「保存」ボタンが Enable になったりする
詳しくは DesignIntf.pas を参照