31ST EMBARCADERO DEVELOPER CAMP

第31回　エンバカデロ・デベロッパーキャンプ

【T1】Delphi/C++テクニカルセッション

「VCL ユーザーのための FireMonkey入門」

株式会社シリアルゲームズ
アプリケーション第3開発部
取締役　細川淳

embarcadero

アジェンダ

- はじめに
- 画像付きエディットを作る
- アニメーションを見る
- まとめ

はじめに

はじめに

- 今回の資料は後から見ても解るように詳しい操作手順が書いてあります！

- 時間が余った場合、複数行の TListBoxItem を作ります
  - ですが、このセッションを最後まで見れば、誰でも作成できるようになっているはずです！

はじめに

- VCL と FMX
  - VCL - Visual Component Library
    - Windows の進化とともに発展してきたコンポーネント群
  - FMX - FireMonkey
    - マルチプラットフォームに対応したコンポーネント群

はじめに

- VCL の考え方
  - Windows のコントロールがベース
    - 見た目は Windows が用意したものになる
      - 最近は Style によって見た目を変更できる

はじめに

- FMX の考え方
  - それぞれの OS でコントロールの見た目が違う
  - …めんどくさいから全部自分で描いちゃおうZE！
    - Style で見た目をコントロール
    - OS ごとに DirectX / OpenGL / OpenGL ES で描画する

はじめに

- FMX の考え方
  - それぞれの OS でコントロールの見た目が違う
  - …めんどくさいから全部自分で描いちゃおうぜ！
  - Style で見た目をコントロール
  - OS ごとに DirectX / OpenGL / OpenGL ES で描画する

画像付きエディットを作る

画像付きエディットを作る

- VCLでは
  1. TEdit 継承したエディットを作る→TImageEdit とします
  2. TImage を作成し、Parent を TEdit にする
  3. TEdit のテキストエリアを変更する

実際に作ってみます。

TImageEdit

```
unit uImageEdit;

interface

uses
  Winapi.Windows, Winapi.Messages, System.Classes, Vcl.Graphics, Vcl.Controls,
  Vcl.StdCtrls, Vcl.ExtCtrls;

type
  TImageEdit = class(TEdit)
  private
    FImage: TImage;
    procedure ImageChange(Sender: TObject);
  protected
    procedure DoImageChanged; virtual;
    procedure SetParent(iParent: TWinControl); override;
    procedure WndProc(var Message: TMessage); override;
    procedure Resize; override;
```

Windows に特化しているので
Winapi をそのまま uses に

TEdit を継承

画像用 TImage

TImageEdit

```
  public
    constructor Create(iOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Image: TImage read FImage;
  end;

implementation

uses
  Vcl.Imaging.Jpeg, Vcl.Imaging.PngImage;
```

外から変更できるように
property として公開

Jpeg と Png を表示するために
uses

TImageEdit

```
{ TImageEdit }

constructor TImageEdit.Create(iOwner: TComponent);
begin
  inherited;

  FImage := TImage.Create(Self);
  FImage.SetBounds(1, 1, 16, 16);
  FImage.Stretch := True;
  FImage.Picture.OnChange := ImageChange;
  FImage.Parent := Self;
end;

destructor TImageEdit.Destroy;
begin
  FImage.DisposeOf;

  inherited;
end;
```

画像用 TImage を生成

画像用 TImage を廃棄

TImageEdit

```
procedure TImageEdit.DoImageChanged;
begin
  if Parent = nil then
    Exit;

  SendMessage(
    Handle,
    EM_SETMARGINS,
    EC_LEFTMARGIN or EC_RIGHTMARGIN,
    MakeLong(FImage.Width + 2, 0));

  Invalidate;
end;

procedure TImageEdit.ImageChange(Sender: TObject);
begin
  DoImageChanged;
end;
```

画像が変更されたときに、EDIT のテキストマージンを設定

テキストが入力できるエリアの左右のマージンを指定できる

TImageEdit

```
procedure TImageEdit.Resize;
begin
  inherited;

  if Parent <> nil then
    FImage.SetBounds(1, 1, ClientHeight, ClientHeight);
end;

procedure TImageEdit.SetParent(iParent: TWinControl);
begin
  inherited;

  DoImageChanged;
end;
```

TImage の大きさを
ClientHeight 四方の正方形に

TImageEdit

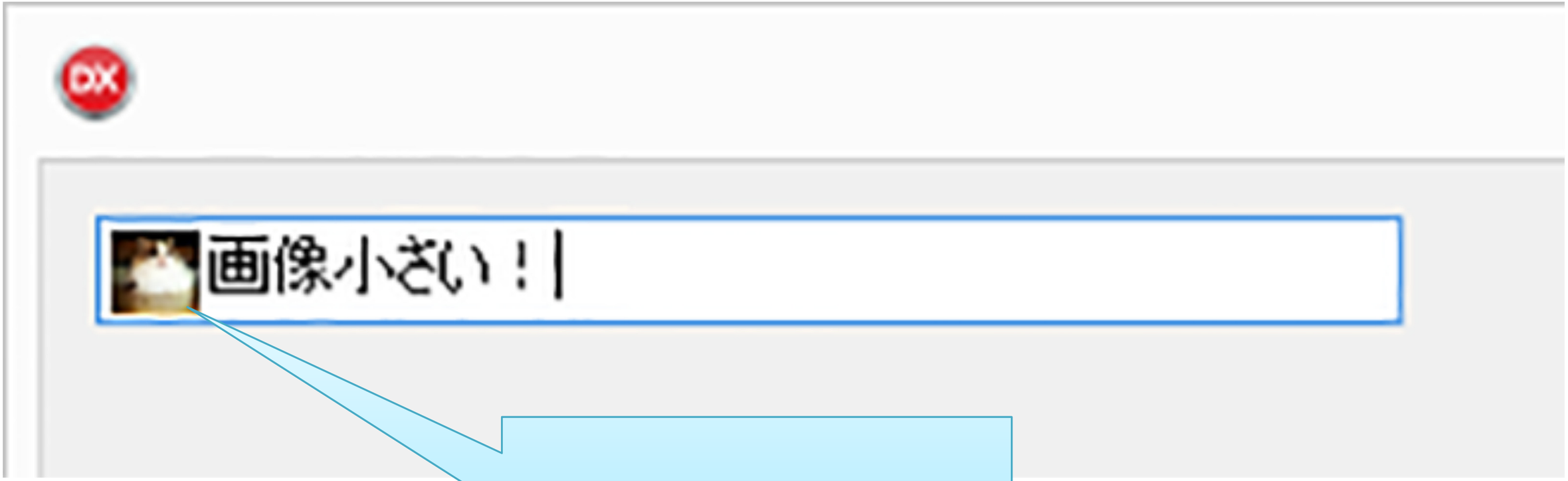
```
procedure TImageEdit.WndProc(var Message: TMessage);
begin
  case Message.Msg of
    CN_CTLCOLORSTATIC,
    CN_CTLCOLOREDIT:
      if FImage.Picture.Graphic <> nil then
        ExcludeClipRect(
          Message.WParam,
          1,
          1,
          1 + ClientHeight,
          1 + ClientHeight);
  end;

  inherited;
end;

end.
```

TImage の部分を描画しないようにクリッピング
（クリッピングしないと Text の背景色で塗りつぶされてしまう）

TImageEdit を使う側

```
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    FImageEdit: TImageEdit;
  public
  end;

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  FImageEdit := TImageEdit.Create(Self);
  FImageEdit.Image.Picture.LoadFromFile('C:¥temp¥luna6.jpg');
  FImageEdit.SetBounds(10, 10, 250, FImageEdit.Height);
  FImageEdit.Parent := Self;
end;
```

実行結果



読み込んだ画像が確かに
表示されている

画像付きエディットを作る

- FMX では…

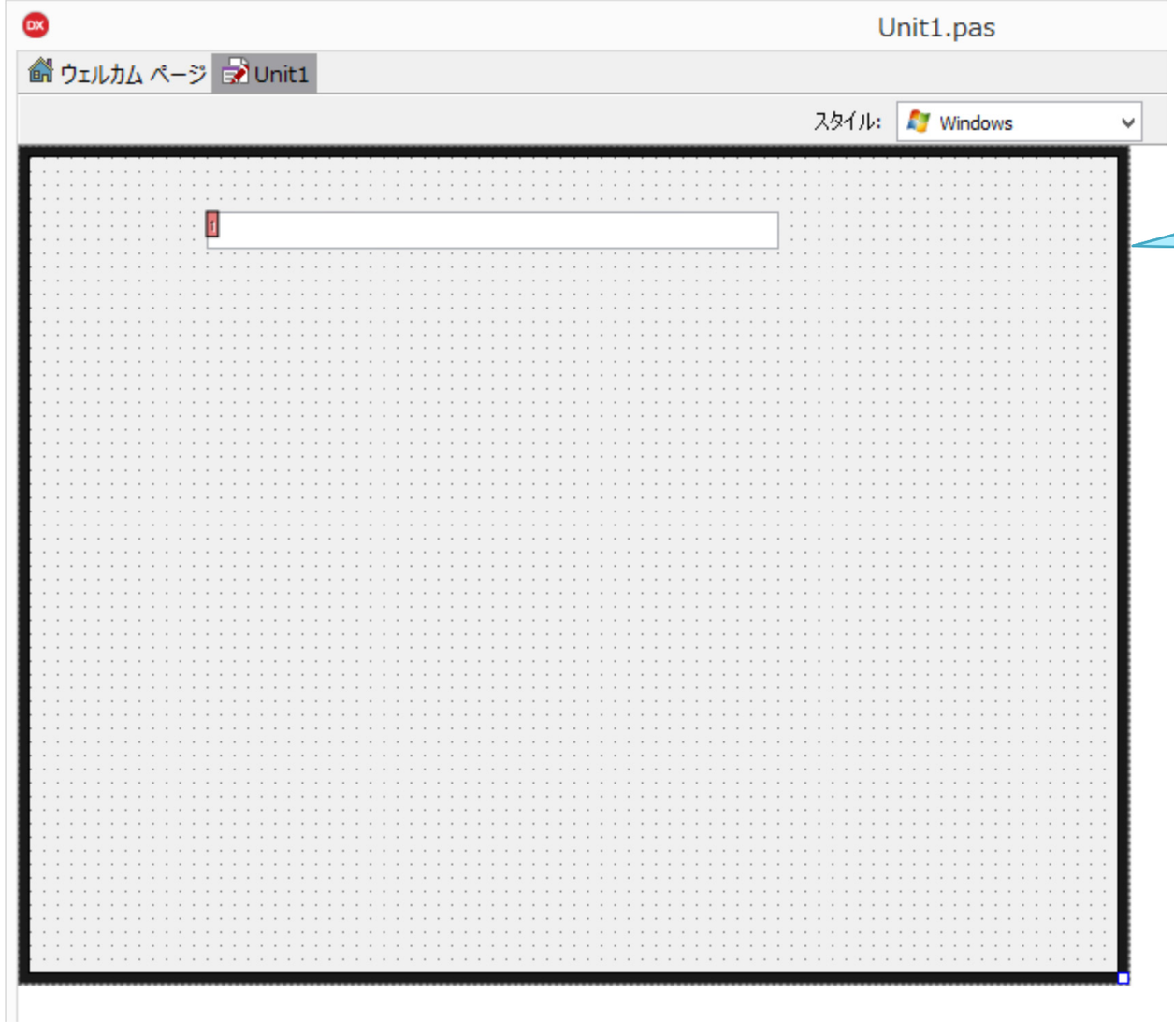
画像付きエディットを作る

- FMX では、どうするか？

スタイルを使います！

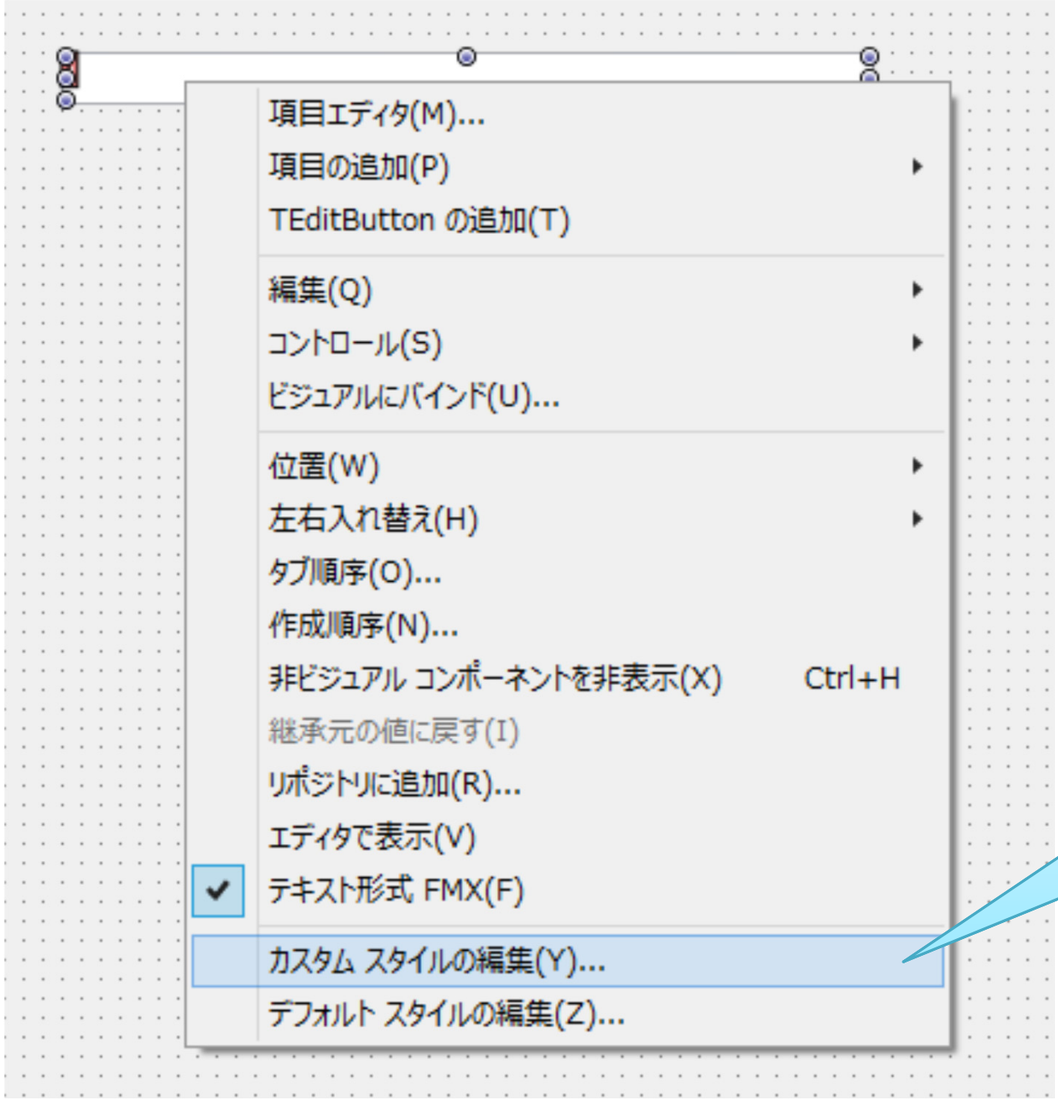
実際に作ってみます。

画像付きエディットを作る



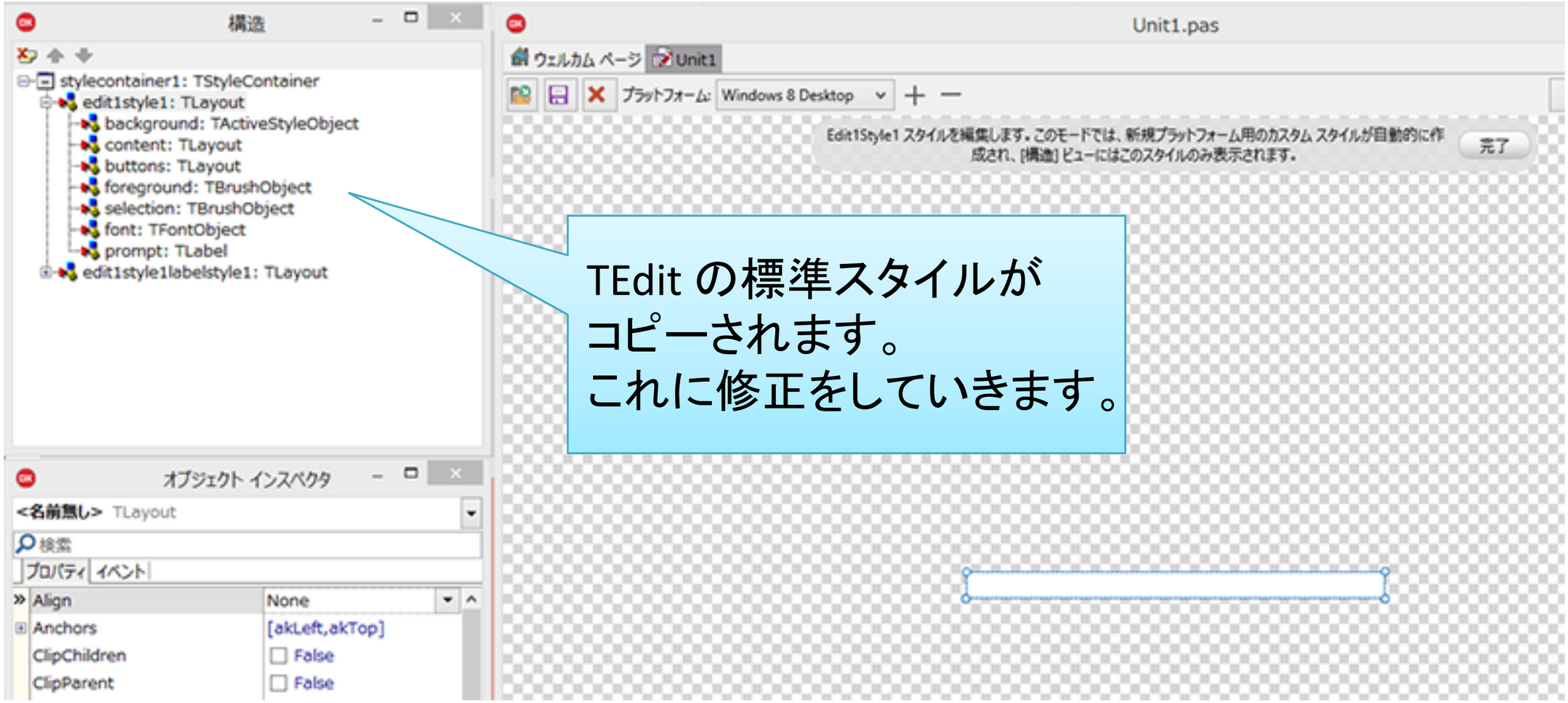
マルチデバイスアプリケーションを新規作成
TEdit をドロップします。

画像付きエディットを作る

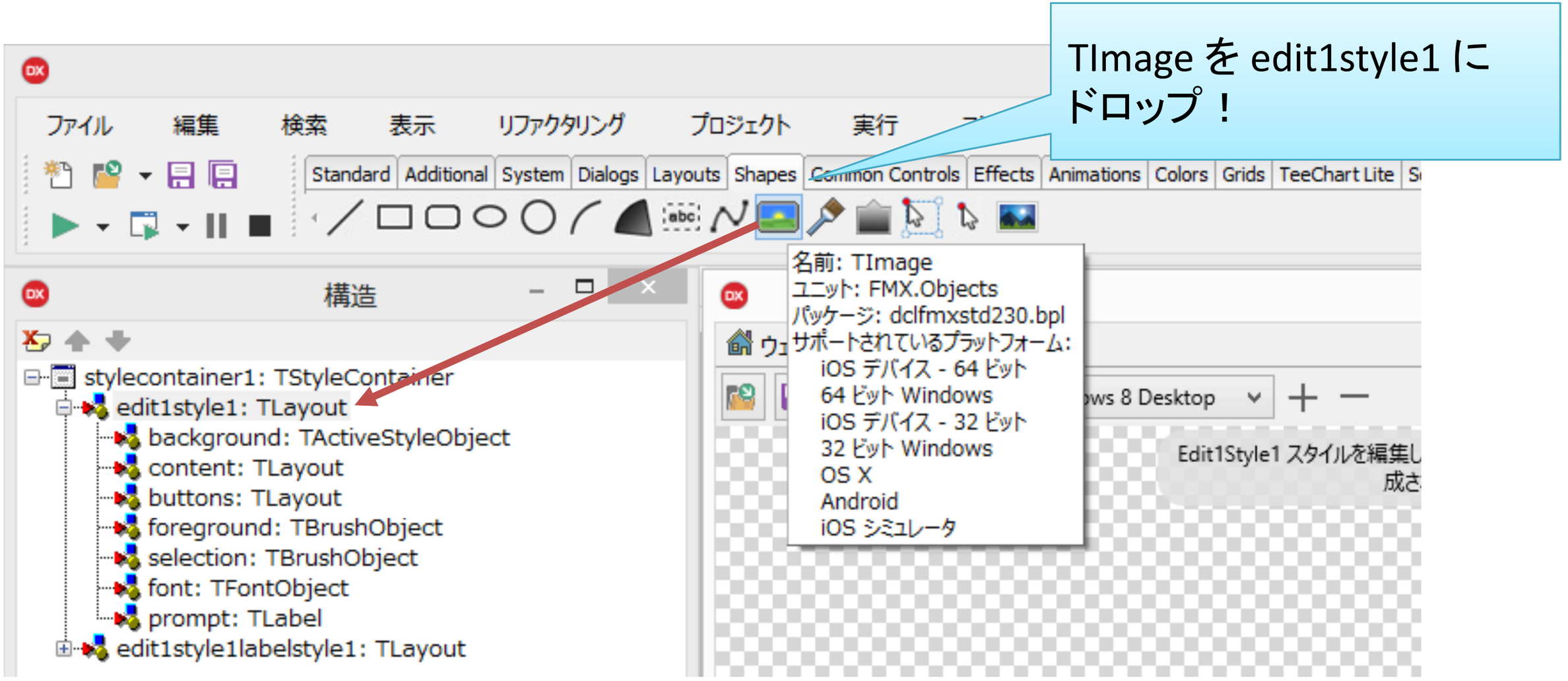


TEdit を右クリックして
「カスタムスタイルの編集」
を選びます。

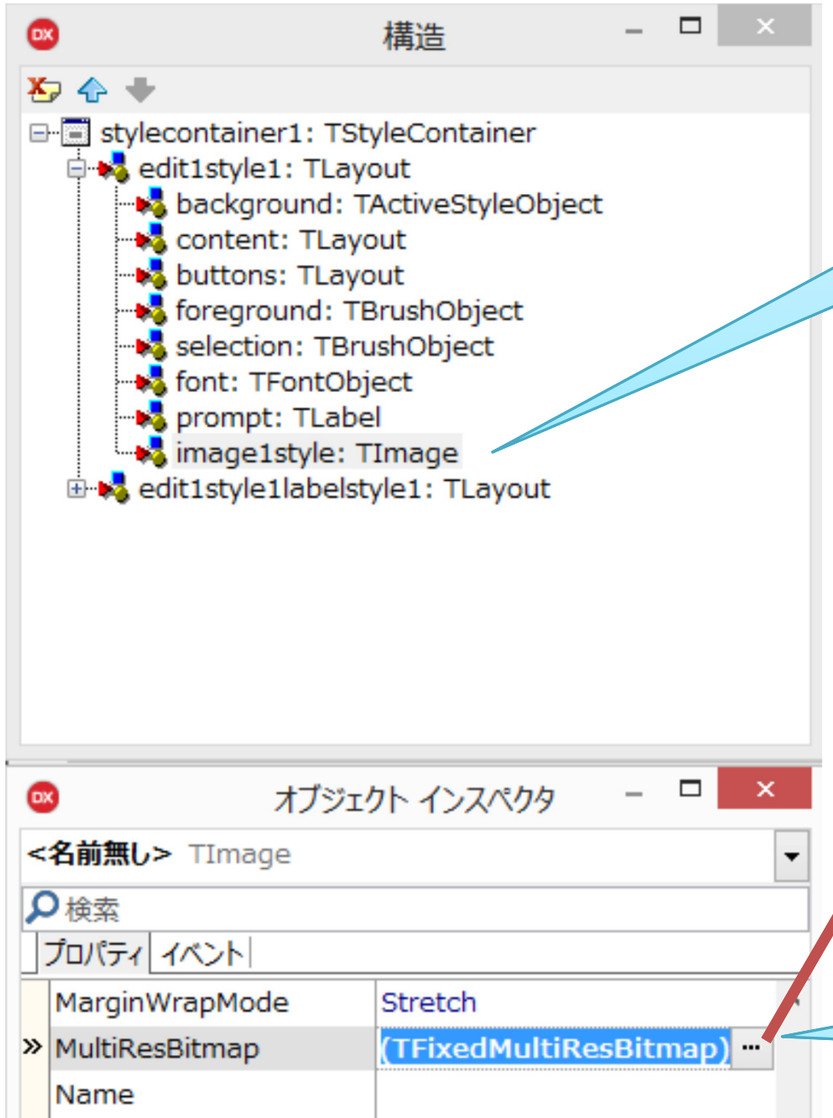
画像付きエディットを作る



画像付きエディットを作る

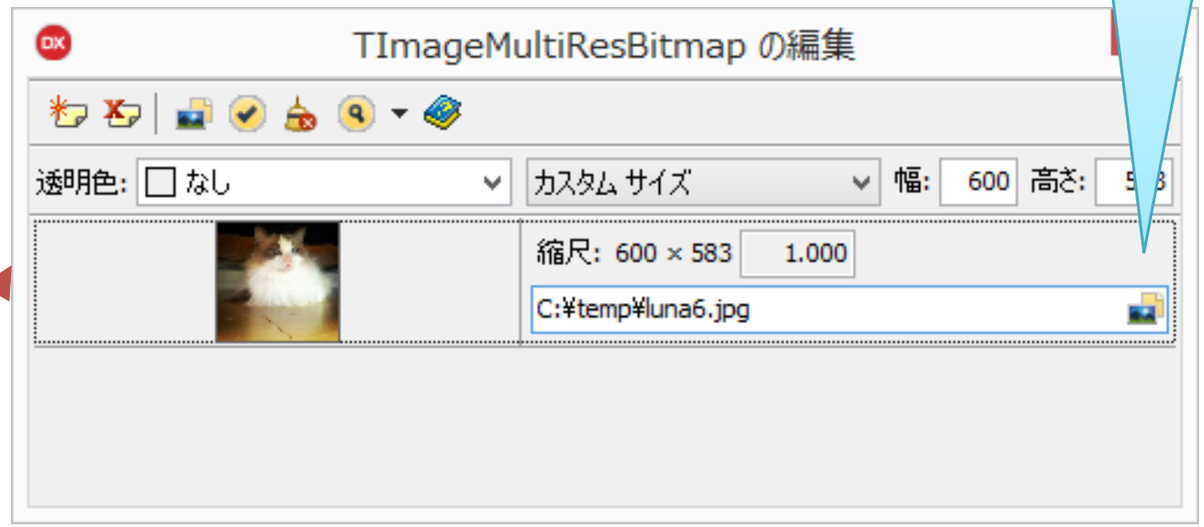


画像付きエディットを作る



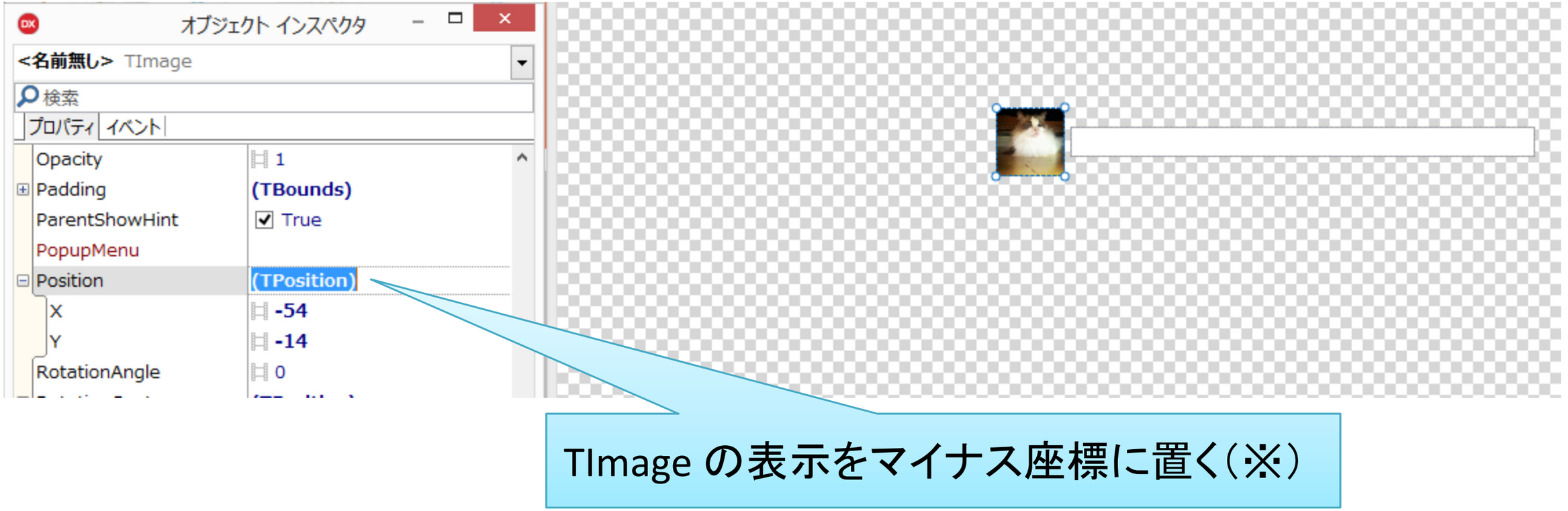
ドロップされた
TImage

画像を選ぶ



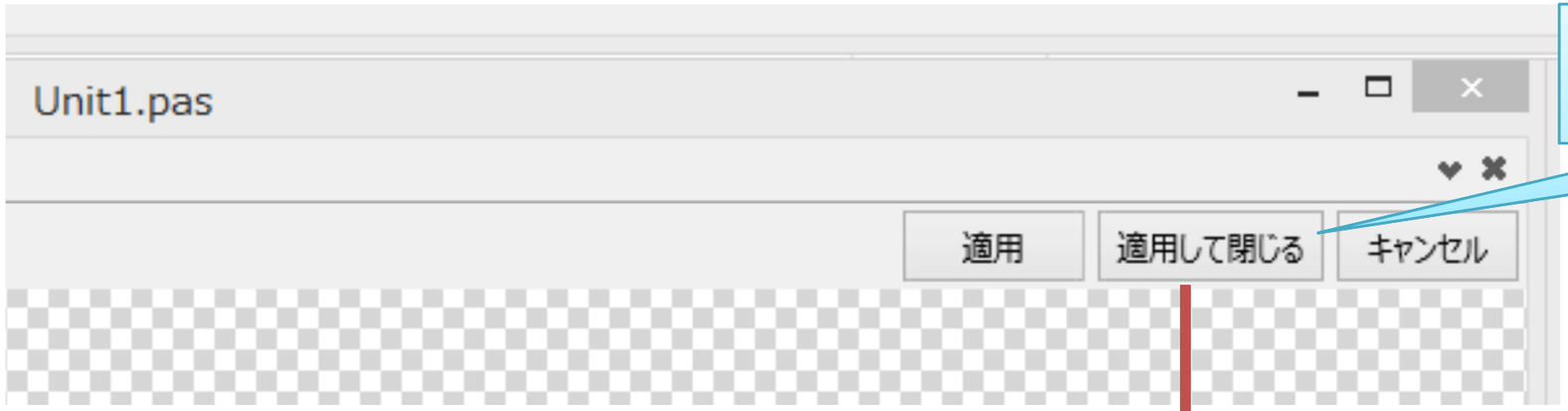
ドロップされた
TImage のプロパティ
MultiResBitmap

画像付きエディットを作る



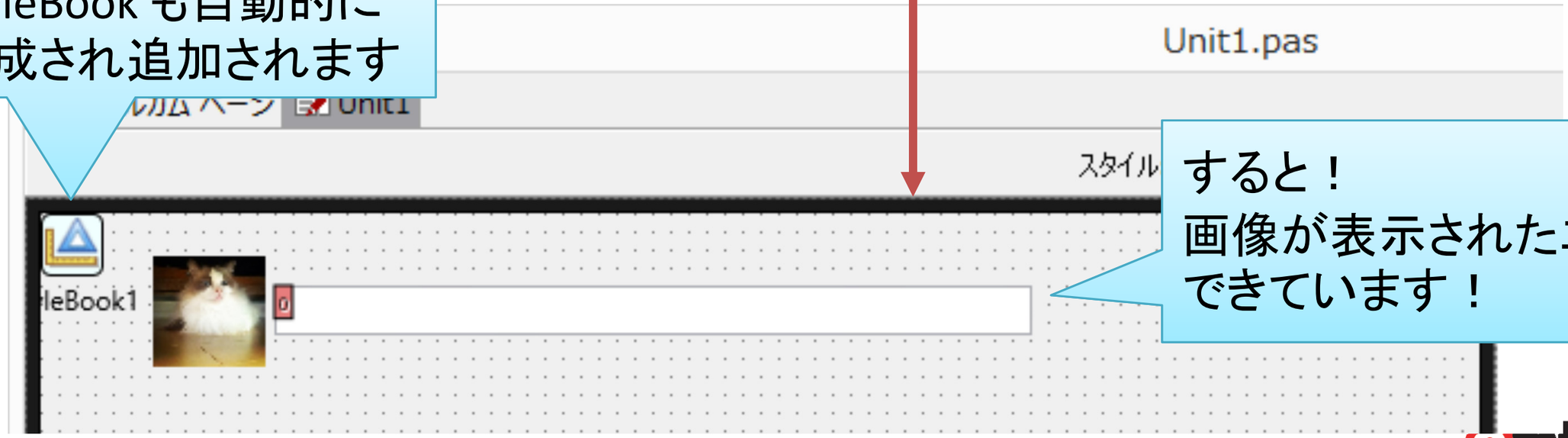
※本来なら親レイアウトに TAlignLayout.Left などで置きますが、
TEdit がプレゼンテーションレイヤーなどに分かれており、正しく作らないとキャレット位置がずれたりします。
そのため、ここでは簡単にこのようしています。

画像付きエディットを作る



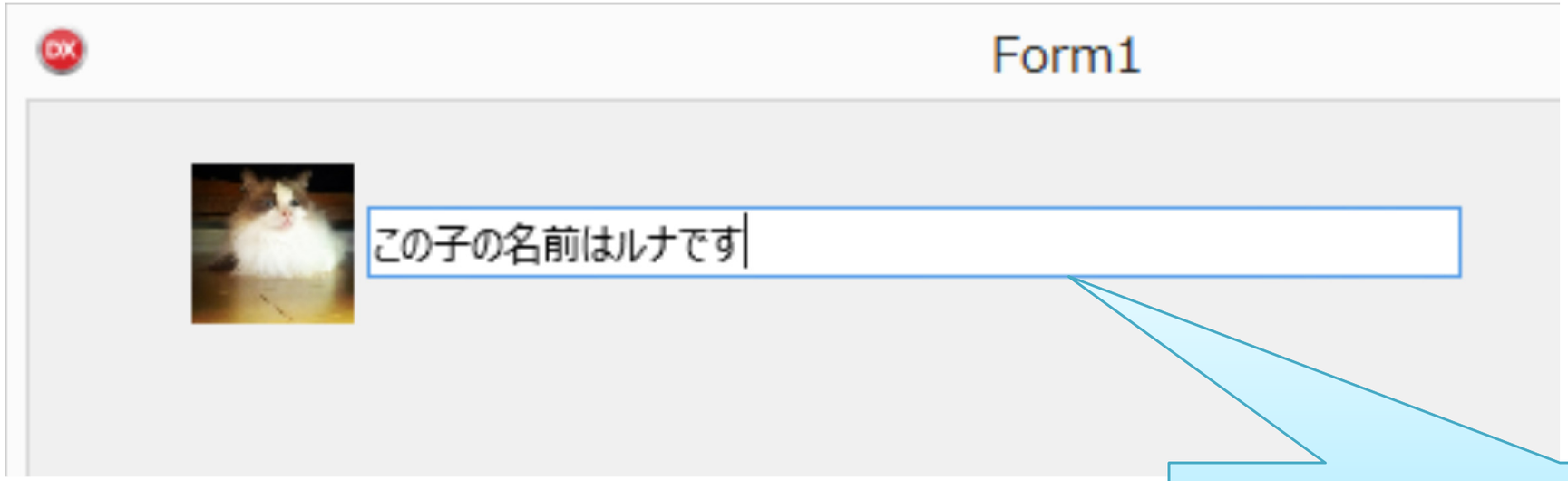
適用して閉じる、を押します。

StyleBook も自動的に
作成され追加されます



すると！
画像が表示されたエディットが
できています！

画像付きエディットを作る



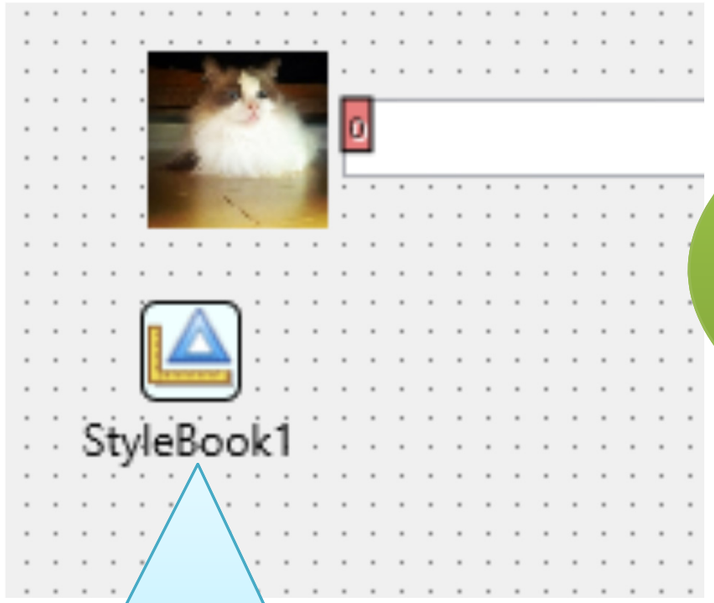
実行すると、正しく画像が表示されたエディットになっています。

画像付きエディットを作る

VCL と違って FMX 版は
画像を変えられない？？

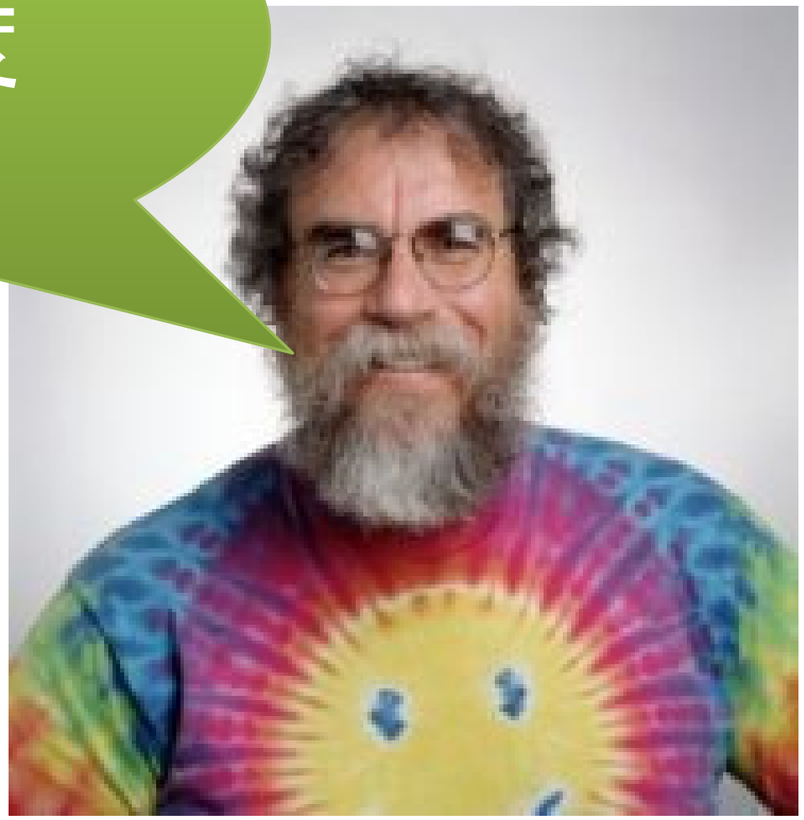
VCL 版は TImage がプロパティとして出ていたので
Image.Picture.LoadFromFile などが使えた！

画像付きエディットを作る



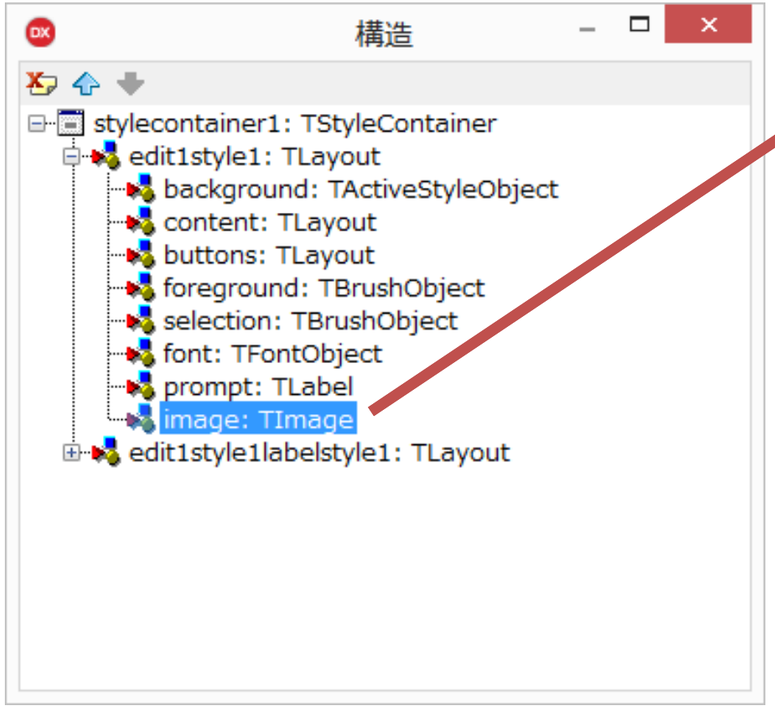
スタイルを編集するためには StyleBook1 をダブルクリックします。

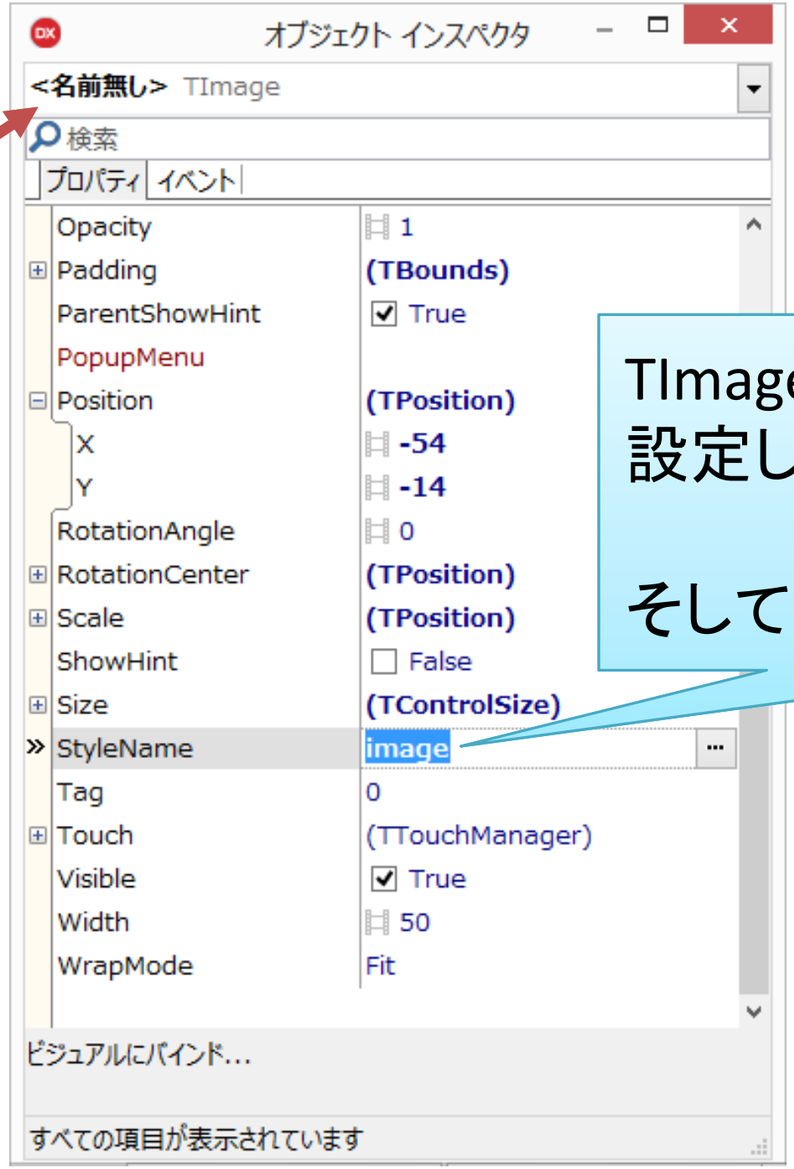
心配無用じゃ！！
スタイルをもう一度
開くんじゃ



＿＿人人人人人人人＿＿
＞　突然のDavid I.　＜
￣Y^Y^Y^Y^Y^Y^Y^Y^Y￣

画像付きエディットを作る

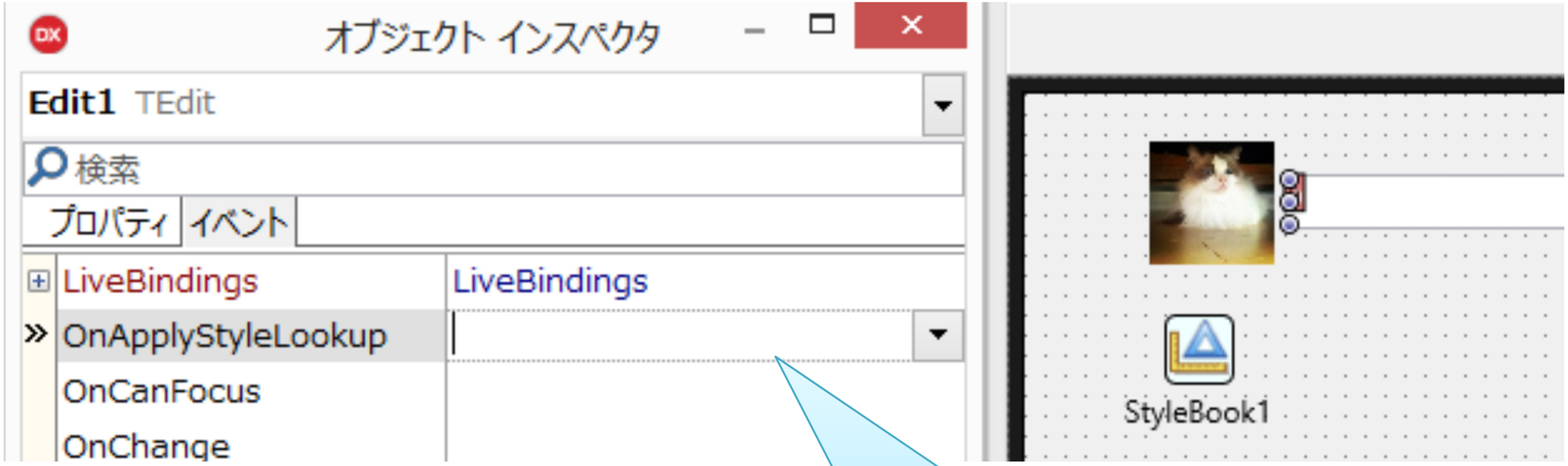




TImage の StyleName を image に設定します

そして、適用して閉じる、で閉じます。

画像付きエディットを作る



OnApplyStyleLookup
イベントハンドラを書きます

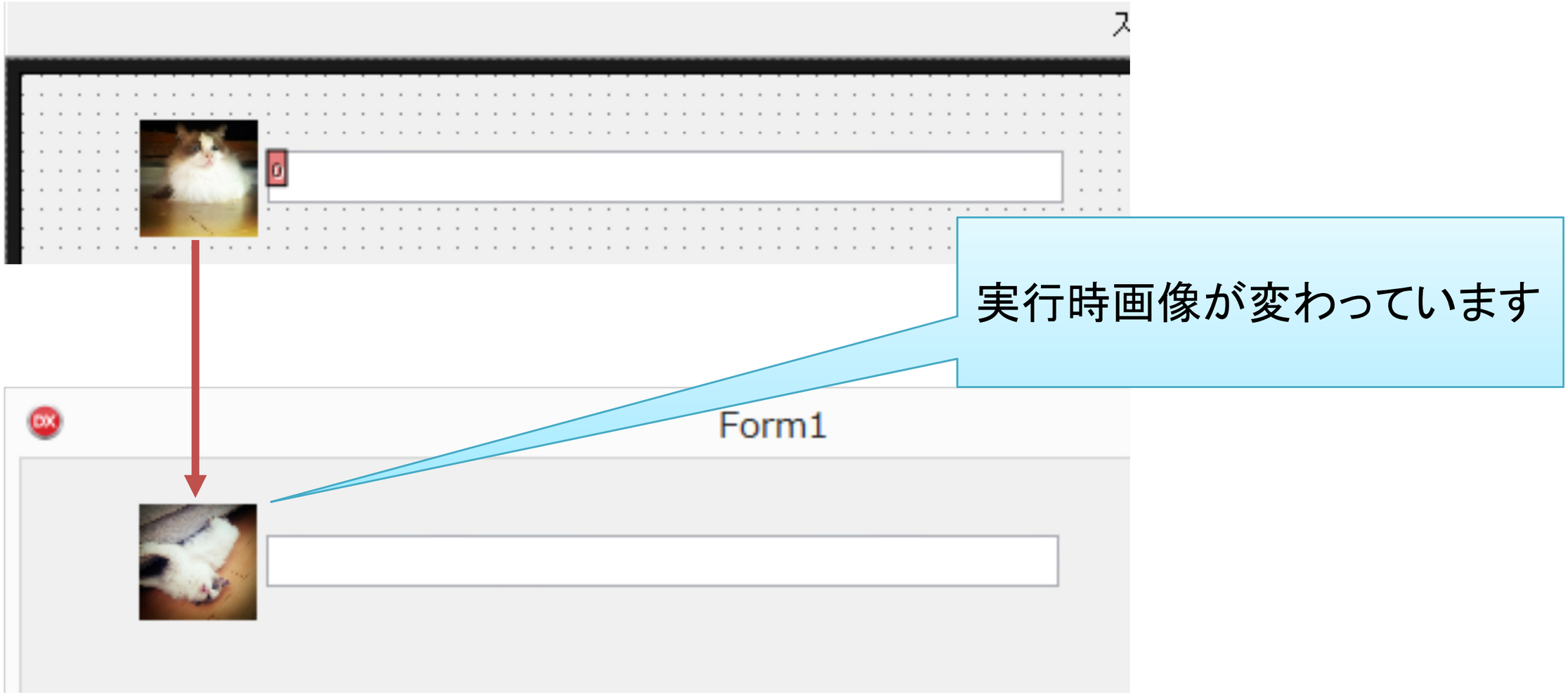
画像付きエディットを作る

```
implementation

{$R *.fmx}

uses
  FMX.Objects;

procedure TForm1.Edit1ApplyStyleLookup(Sender: TObject);
var
  Image: TImage;
begin
  Image := Edit1.FindStyleResource('image') as TImage;
  Image.Bitmap.LoadFromFile('C:¥temp¥LunaGiro.jpg');
end;
```

TImage は FMX.Objects に
宣言されているので uses します

StyleName で指定した名前で
スタイル中のコントロールを
取り出します。

画像付きエディットを作る



画像付きエディットを作る

TStyledControl の子孫は全て

OnApplyStyleLookup

を持ちます。

FindStyleResource メソッドでスタイル内のコントロールを取得できます。

画像付きエディットを作る

注意したいのは、OnApplyStyleLookup イベントより前ではスタイルのコントロールは生成されていない！ということです。

たとえば、TForm.OnCreate で FindStyleResource を呼んでも値はとれません。

画像付きエディットを作る

Style を理解する前、StyleLookup と StyleName の
２つが存在する理由がわかりませんでした。
いかがでしょう？

画像付きエディットを作る

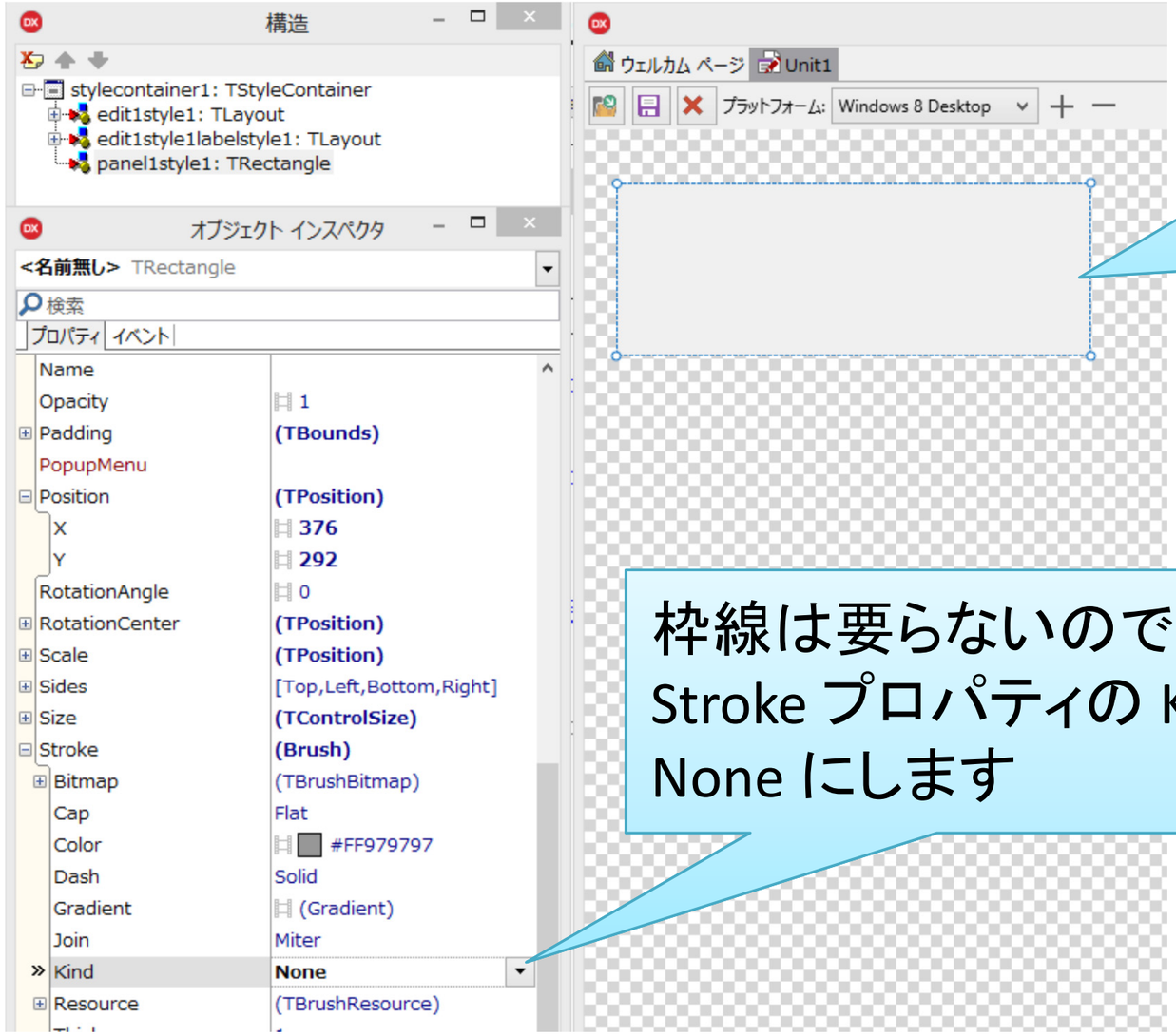
- StyleName
  - その名の通り、スタイルの名前
  - StyleLookup や FindStyleResource で指定します
- StyleLookup
  - 適用したいスタイルの名前を指定します

画像付きエディットを作る

もう1つの冴えたやり方を紹介します

...たった1つではない！！

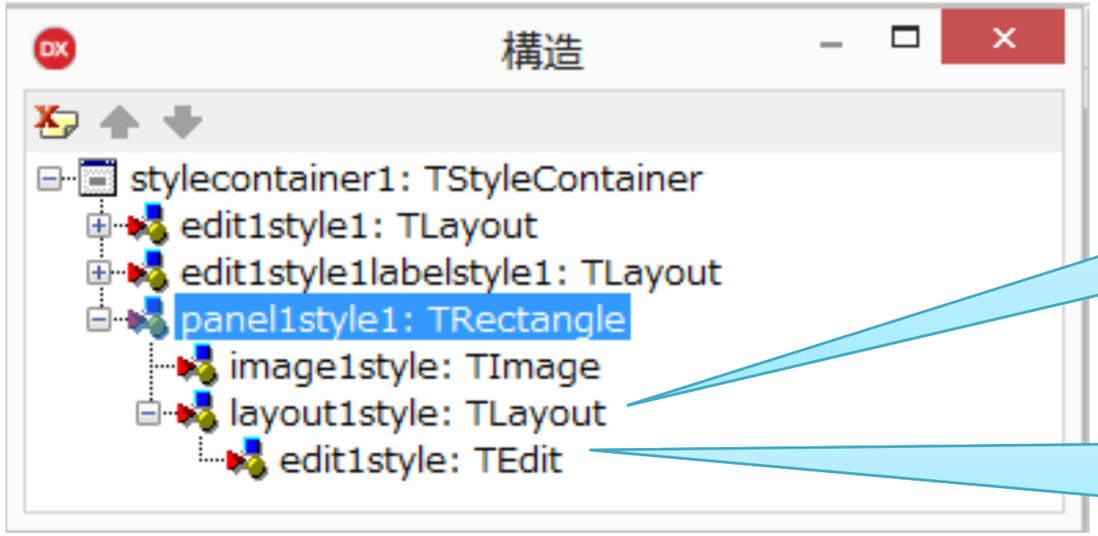
画像付きエディットを作る2



パネルを置いて
カスタムスタイルの編集をクリックして、
スタイルエディタを開きます。

枠線は要らないので
Stroke プロパティの Kind を
None にします

画像付きエディットを作る2



TLayout は何もしないコンテナ

TEdit も普通にスタイルにおける！！！

TImage, TLayout, TEdit を上のように置きます。
TEdit は TLayout の子供に！
FMX は全てのコントロールが親（コンテナ）に
なれます！！

画像付きエディットを作る2

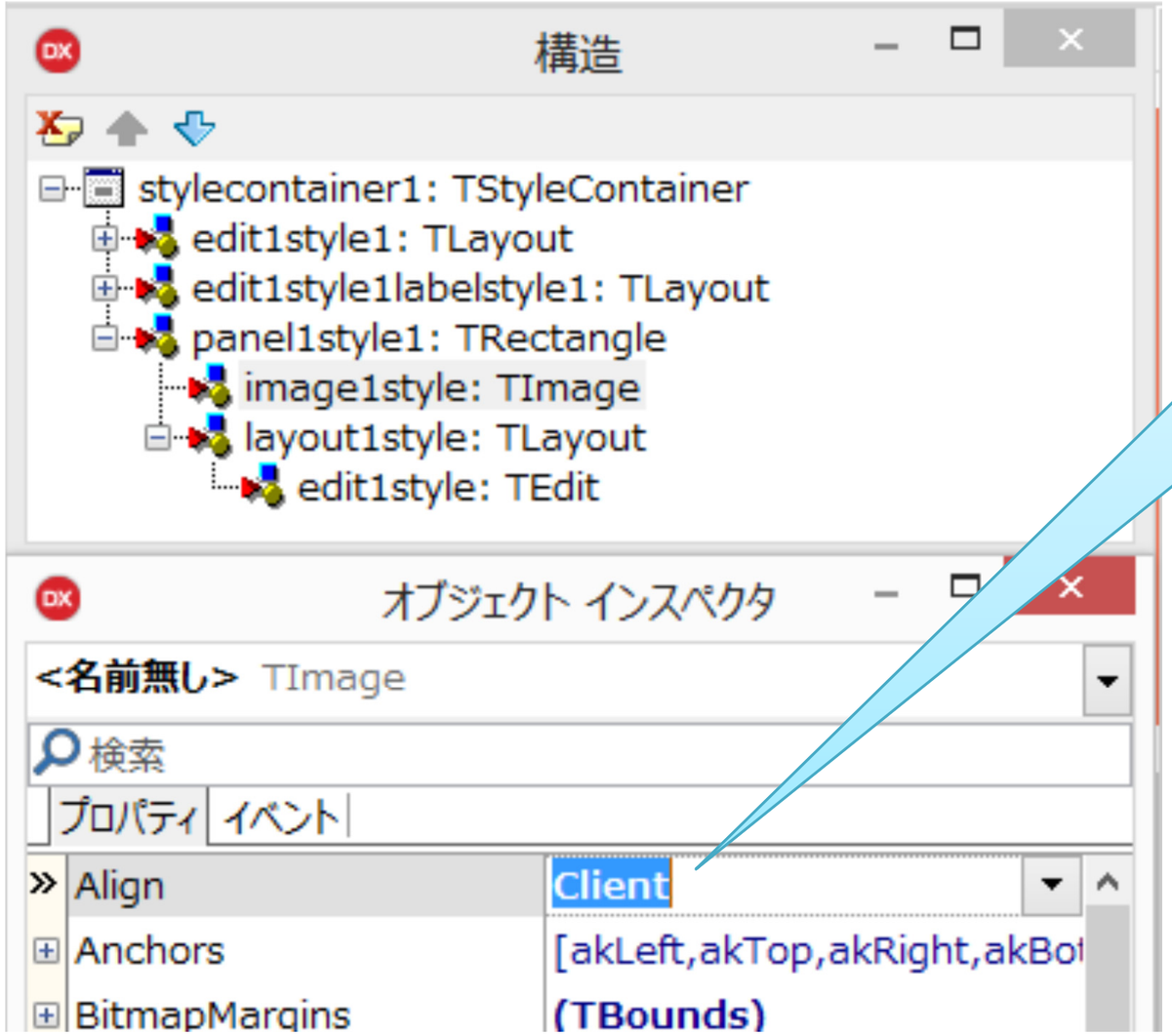
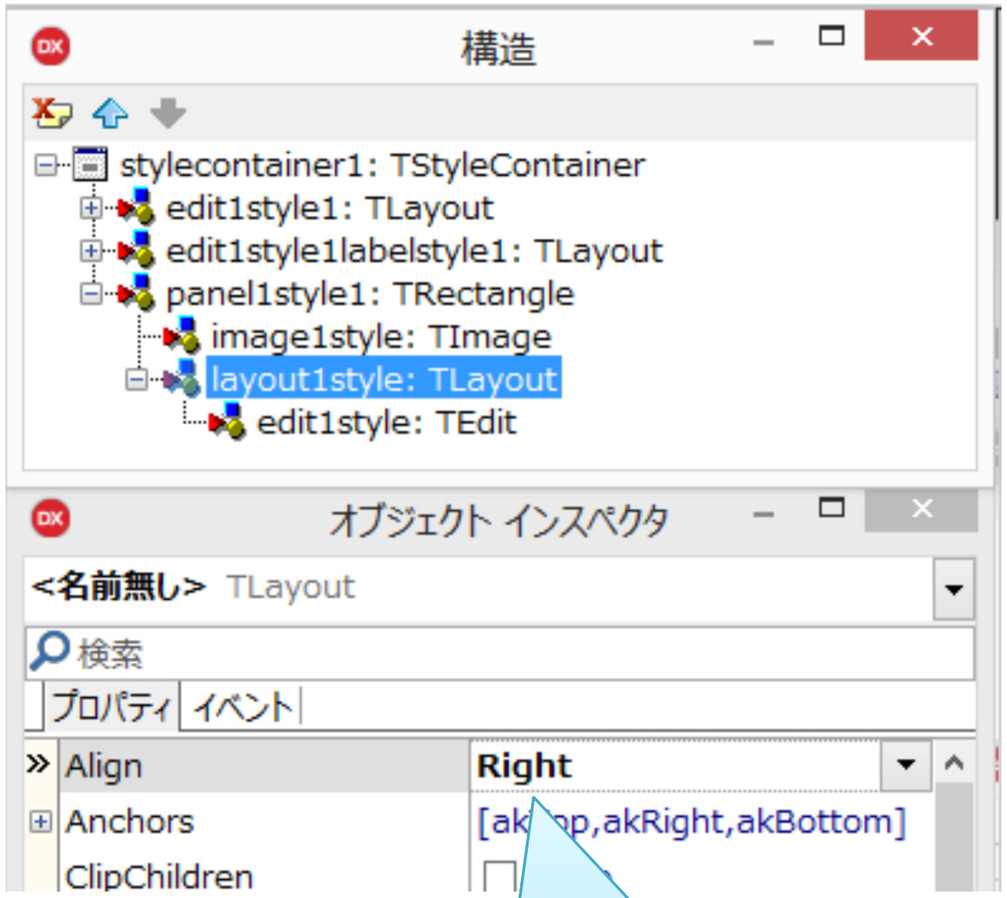
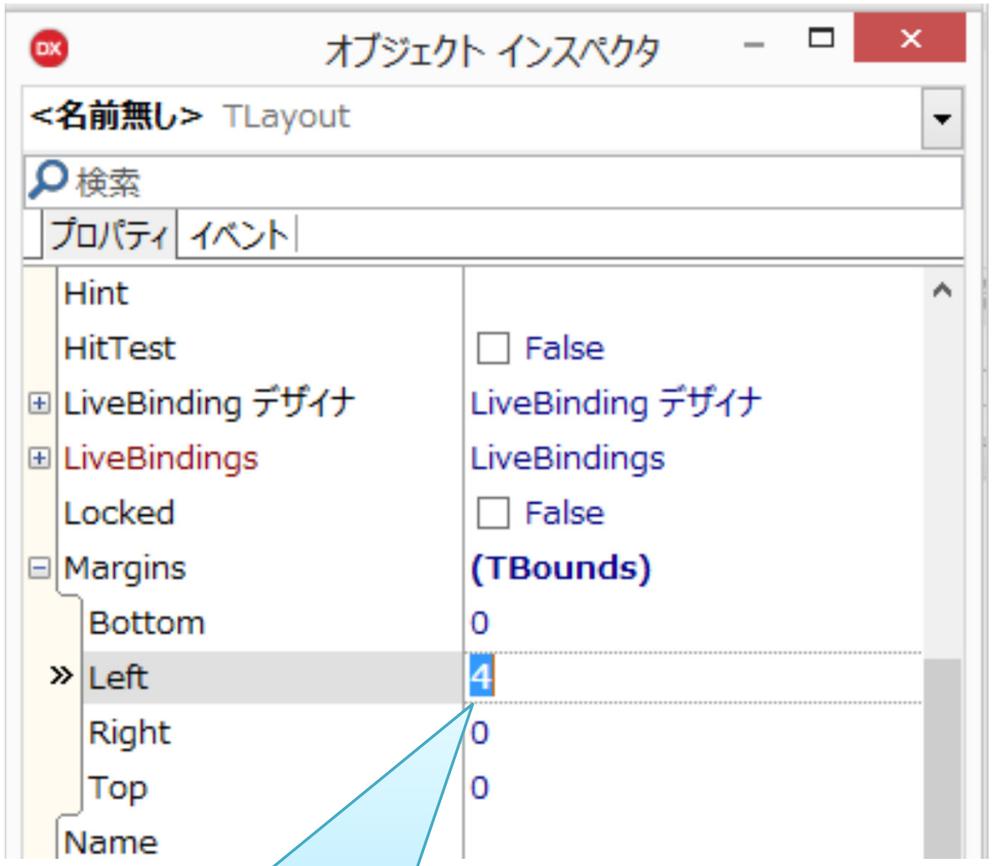


Image の
Align を Client に

画像付きエディットを作る2

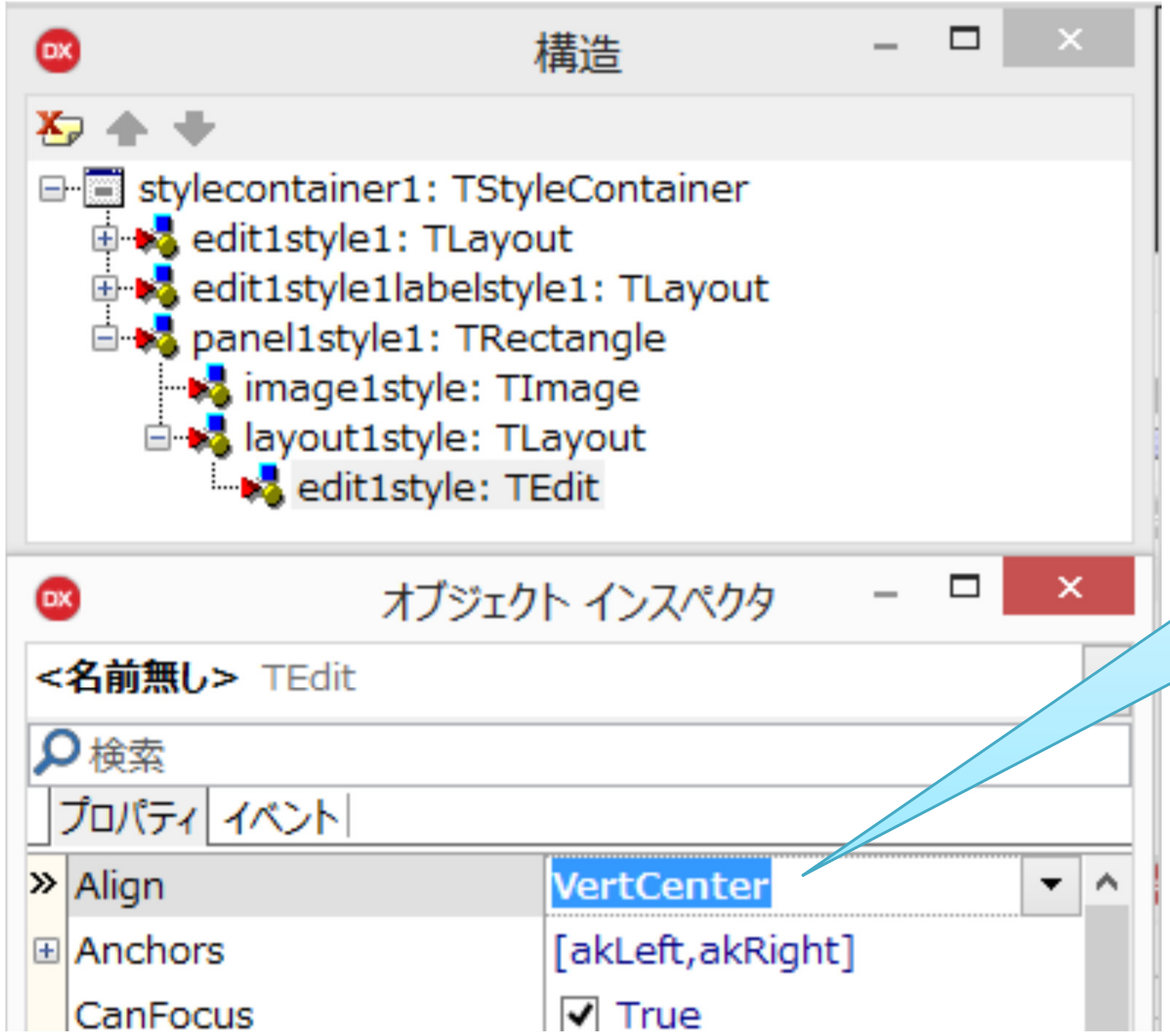


Layout の
Align を Right に



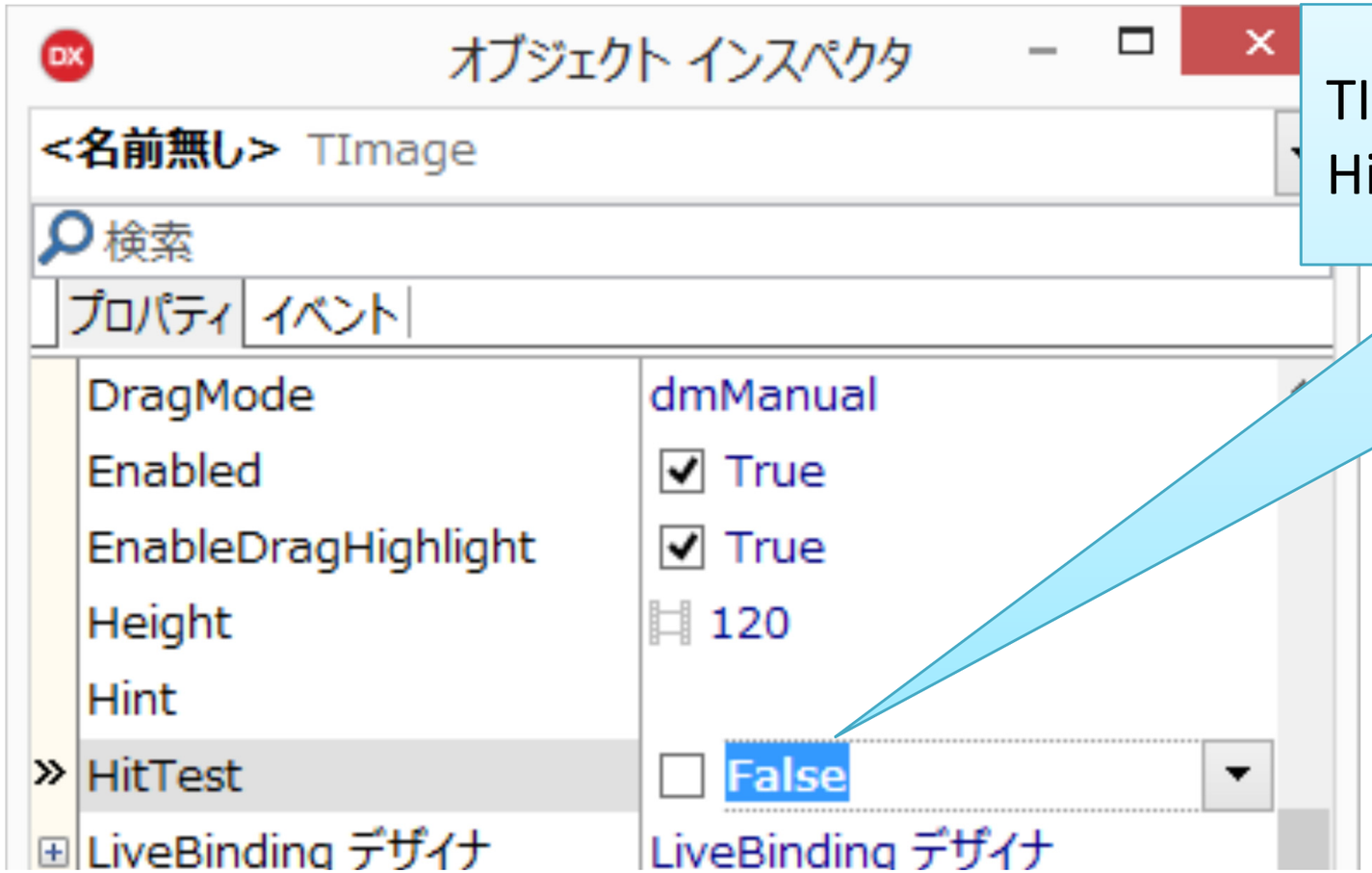
Layout の
Margins.Left に 4 を指定

画像付きエディットを作る2



Edit の
Align を VertCenter に

画像付きエディットを作る2



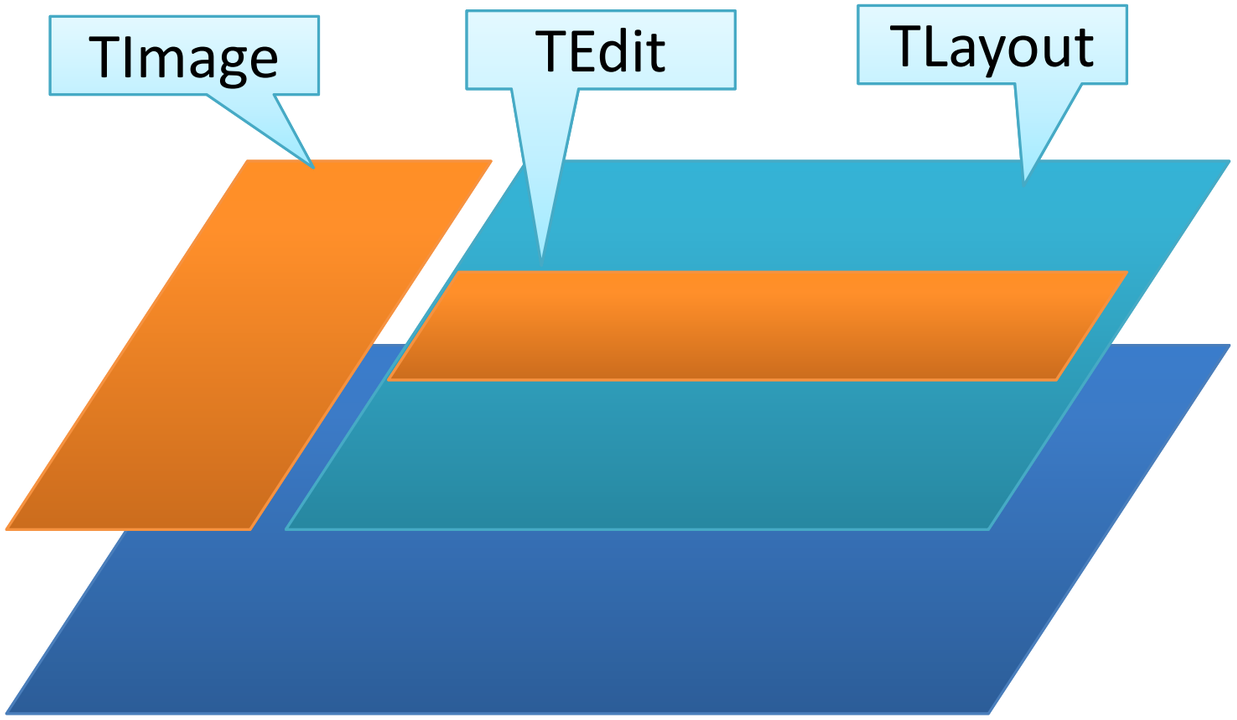
TImage と TLayout の
HitTest プロパティは Flase にする！

HitTest プロパティが True の場合
マウスの押下がコントロールに
通知されます。
親には通知されません。
特に必要ではない時は False に
します。

画像付きエディットを作る2

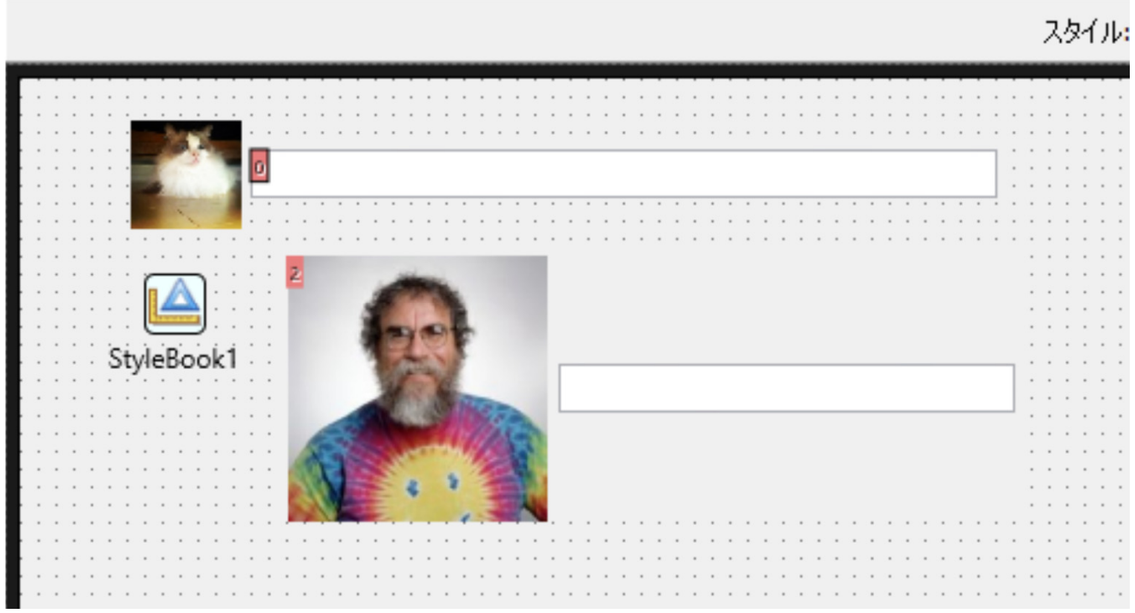
ここまでの操作で、次のようになります。
ついでに、TImage.MultiResBitmap に画像を読み込んでおきます。

Margins.Left による空隙

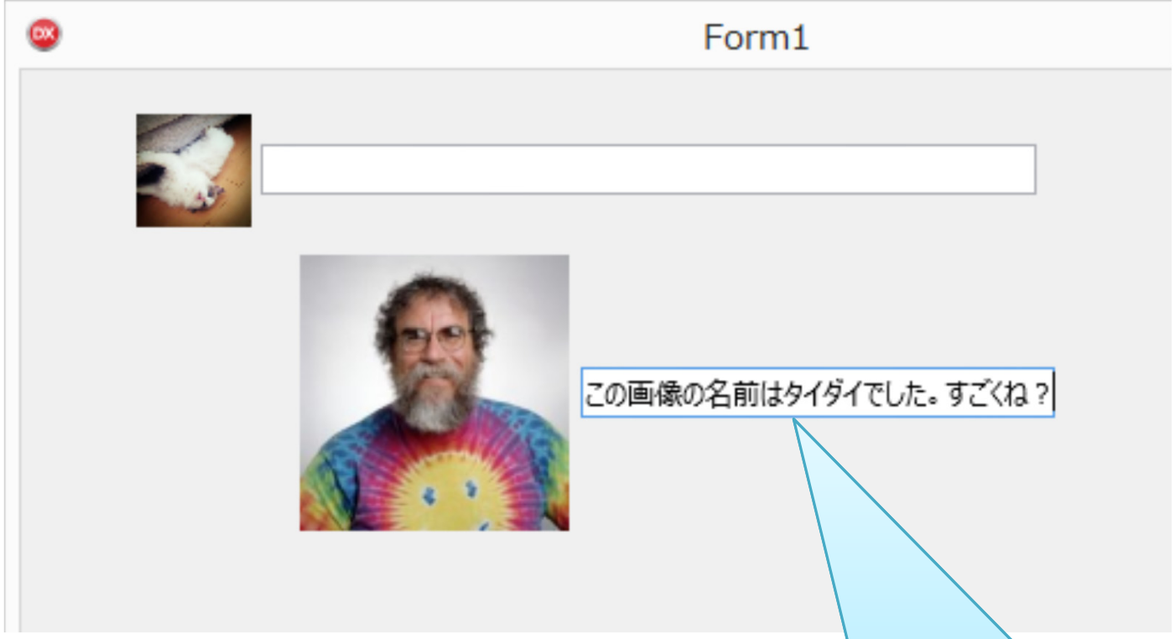


画像付きエディットを作る2

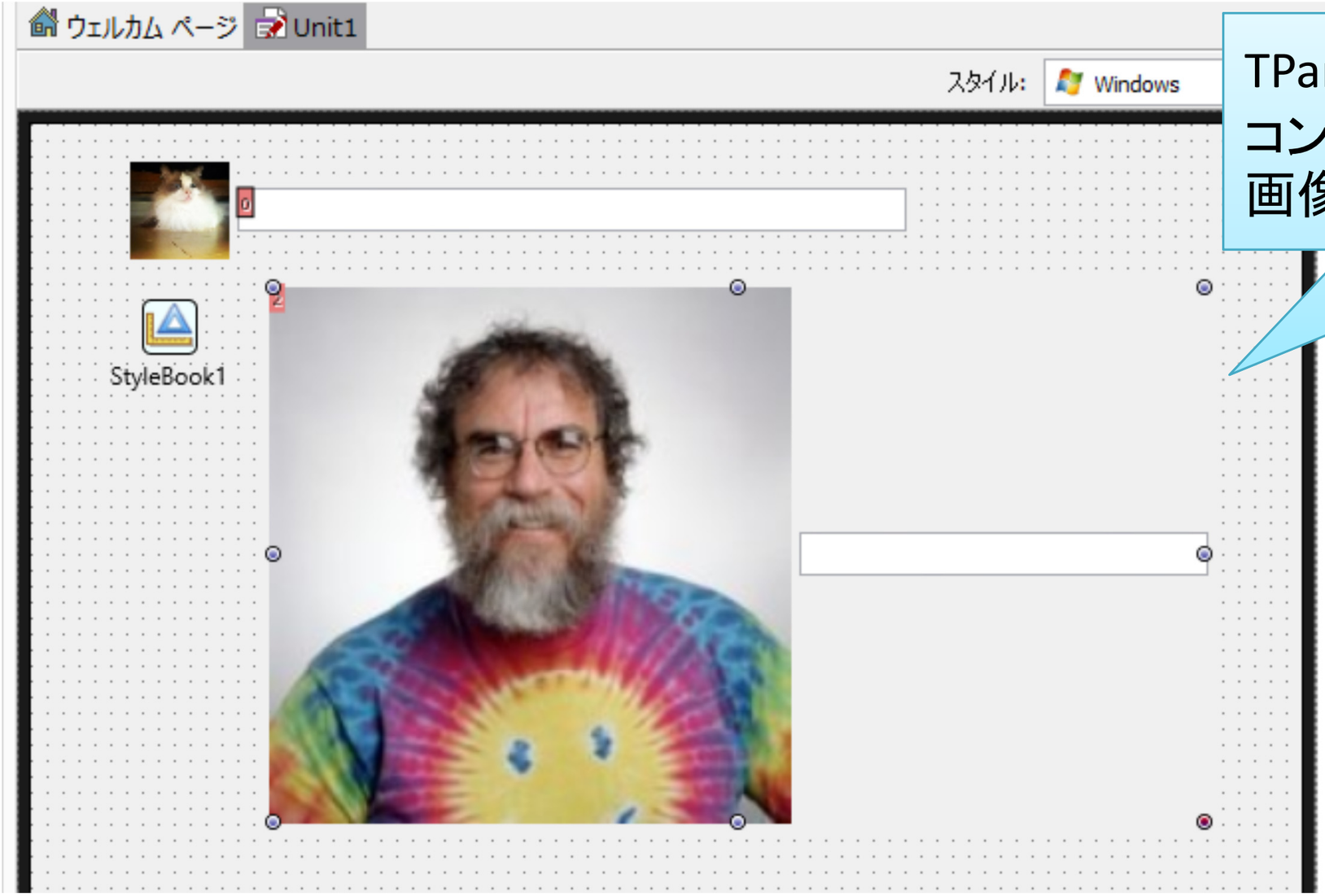
設計時



実行時



TPanel なのに！
文字が入力できている！



TPanel のスタイルにする利点は
コントロールのリサイズに追随し
画像のサイズも変わるところ

画像付きエディットを作る2

- FMX にとってコントロールとは「素体」でしかない
- スタイルが変わると「機能」も変わってしまう
  - TPanel のスタイルに TEdit を付与できる

- VCL のコントロールは機能そのものであった！

画像付きエディットを作る2

ここで冒頭のスライドを思い出してください

はじめに

- FMX の考え方
  - それぞれの OS でコントロールの見た目が違う
  - …めんどくさいから全部自分で描いちゃおうぜ！
  - Style で見た目をコントロール
  - OS ごとに DirectX / OpenGL / OpenGL ES で描画する

画像付きエディットを作る2

FireMonkey.Style は
見た目だけでなく
機能を持たせられる！

アニメーションを使う

アニメーションを使う

- 最近のリッチなアプリケーションでは当たり前のマウスオーバーで色が変わる仕組みはどうやっているのでしょうか？

アニメーションを使う

- VCL では
  1. TTimer を置く
  2. OnEnter
     1. Brush.Color を初期カラーに変更し描画
     2. Timer.Enabled := True にする
  3. Timer.OnTimer で Brush.Color を計算し描画
  4. OnLeave で TTimer.Enabled := False

このような手順が必要になります。

アニメーションを使う

- FMX では！！

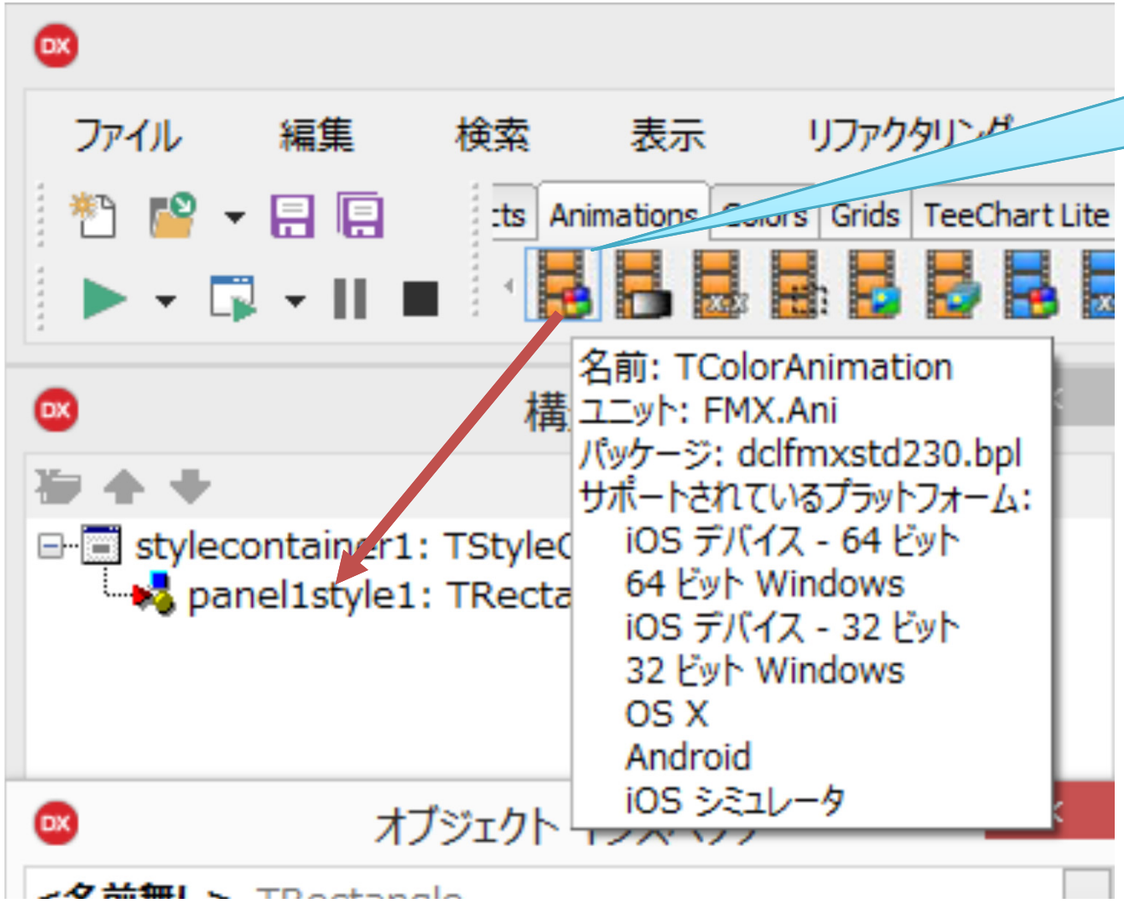
これもスタイルでやります！！

アニメーションを使う

ここでは、マウスオーバーで色が変わるパネルを作ります。

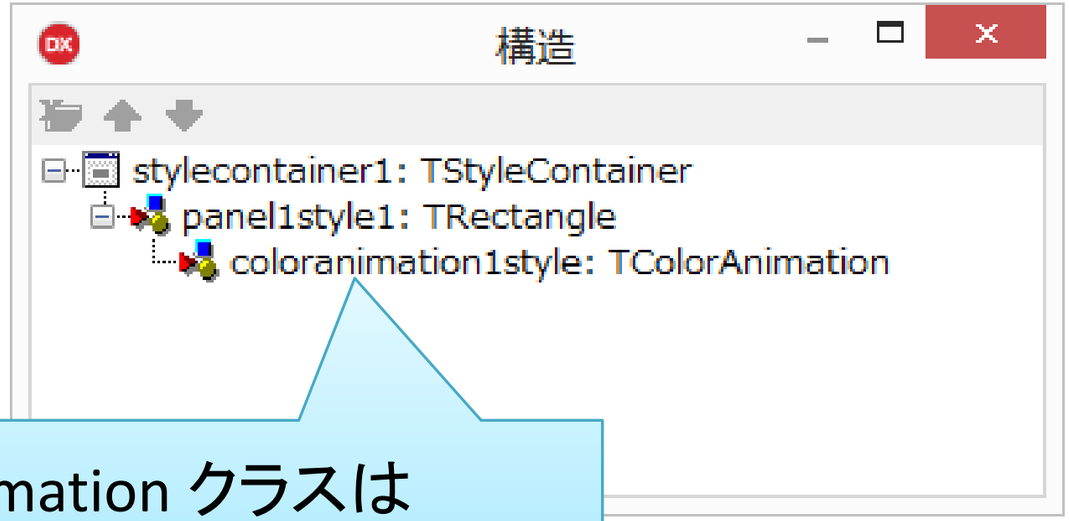
1. 新規にマルチデバイスアプリケーションを作ります
2. TPanel をます。
3. TPanel のカスタムスタイルを編集します

アニメーションを使う



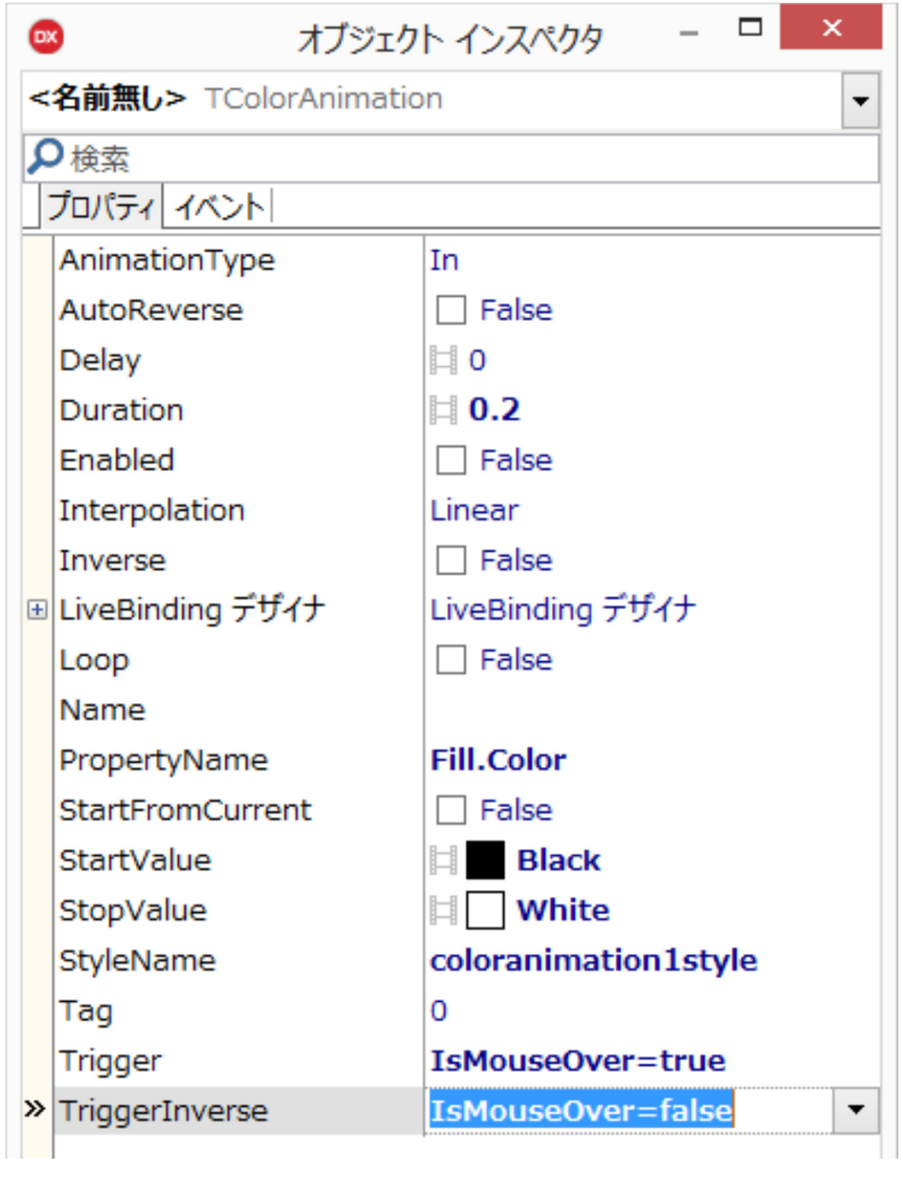
TColorAnimation を
panel1style の下にドロップします。

ドロップ後



Animation クラスは
全て親に対して働きます

アニメーションを使う



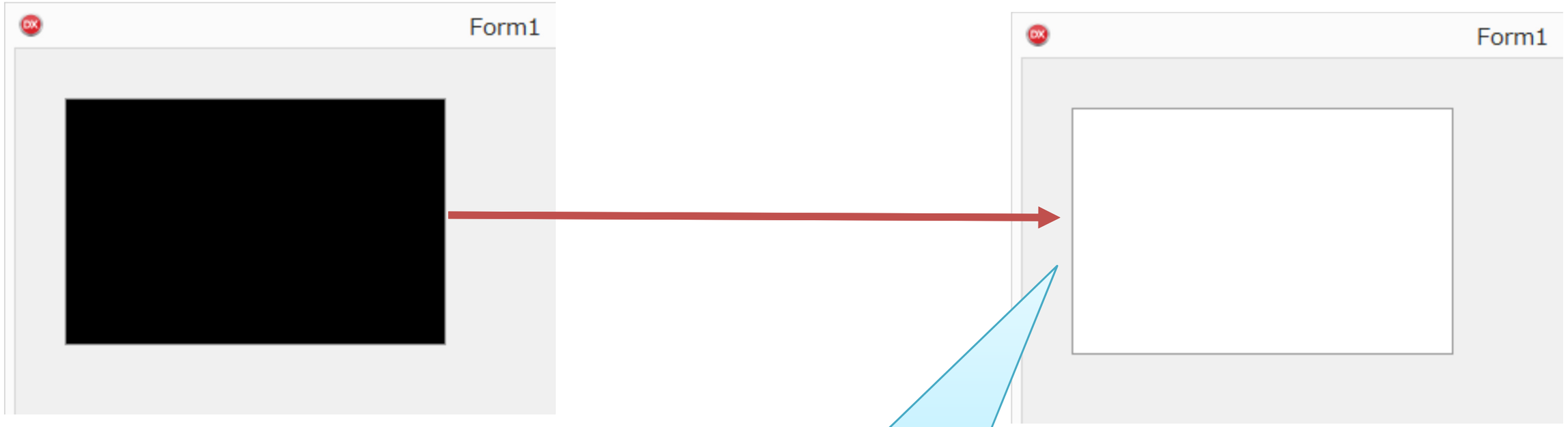
TColorAnimation のプロパティ値を

PropertyName	Fill.Color
StartValue	Black
Trigger	IsMouseOver = True
TriggerInverse	IsMouseOver = False

に設定します

たったこれだけで！
MouseOver 時に色を変更可能です！

アニメーションを使う



MouseOver で
色がふわっと変わる！

アニメーションを使う

- TAnimation
  - PropertyName
    - 親のプロパティ名が自動的にリストアップされて表示される
    - そのアニメーションクラスで適用可能なもの
    - TColorAnimation では、Fill.Color / Stroke.Color がリストアップされていた

アニメーションを使う

- TAnimation
  - StartValue / StopValue
    - アニメーション開始時の値と、終了時の値
    - StartValue はトリガーが掛かったときだけ指定されるので、親コンポーネントの対象を StartValue と同じ値にしておくと良い
      - 今回は TPanel.Fill.Color を Black にしておく

アニメーションを使う

- TAnimation
  - Trigger / TriggerInverse
    - アニメーションを開始するきっかけ
    - 親コントロールの Is～ プロパティが表示されている
    - こちらは親コントロールに適用可能かどうかは判断されない
    - TPanel には IsPress プロパティは存在しないが指定できる

はじめに

- FMX の考え方

  それぞれの OS でコントロールの見た目が違う

  …めんどくさいから全部自分で描いちゃおうZE！

  ~~**Style で見た目をコントロール**~~

  OSごとに DirectX / OpenGL / OpenGL ES で描画する

Style は見た目だけではない！
色々な機能を持たせられる！！

まとめ

まとめ

- VCL を使っていると勘違いしやすいポイント
  - VCL ではこうやる、を **FMX のコントロール**に求めがち
  - その多くは**スタイルで実現**する
  - FMX のスタイルは VCL スタイルと違い、見た目だけではない！
  - スタイル側に機能を持たせるため、FMX では**カスタムコントロールを作る事はほぼ無くなった**

まとめ

- FMX スタイルを上手に使うには
  - **Shape カテゴリのコントロールを使う**
    - Shape カテゴリのコントロールは Style を作るため（といってもいい）のコントロール群
  - **TLayout を使う**
    - TLayout ごとに Align を指定できる
    - TLayout を複数使うと、複雑な Align も実現できる
      - 今回の例でいえば、TLayout には Right で右端に寄るようにし、TEdit は VertCenter で垂直方向の真ん中に表示できた。

付録

- FMX でカスタムコンポーネントを作る場合とは？
  - コンポーネントのユーザーに選択権を与える場合
  - たとえば、画像付きエディットでは
    - 画像の選択
    - 画像の位置の選択

    など
  - ユーザーの意思が入る場合、スタイルで何とかするのは難しい
  - こういった場合にはカスタムコンポーネントを作成します

31ST EMBARCADERO DEVELOPER CAMP

第 31 回　エンバカデロ・デベロッパーキャンプ