畑中 侑

株式会社ミガロ.

システム事業部 システム2課

[Delphi/400]

FireMonkeyの活用 カメラコンポーネントを使ったアプリ

略歴
1983年7月6日生まれ
2006年3月 京都産業大学 法学部卒業
2006年4月 株式会社ミガロ. 入社
2006年4月 システム事業部配属

現在の仕事内容
システムの受託開発を担当しており、要件確認から納品・フォロー、保守作業に至るまで、システム開発全般に携わっている。

1. はじめに

Delphi/400は登場当初より、Windowsネイティブ開発環境としてVCLフレームワークを提供してきたが、XE3から新たにWindows、Mac対応が始まり、現在ではWindows、Mac、iOS、Androidの4つのプラットフォームに対応したFireMonkeyフレームワークが備わっている。

VCLフレームワークはWindows APIをラッピングしたフレームワークであるのに対して、FireMonkeyフレームワークは、OS固有のAPIに依存しないため､マルチデバイス化が可能である。

VCLとFireMonkeyとでフレームワークが異なるといっても、開発者がどちらを採用しても、コンポーネントを画面に配置し、必要なイベントを実装するといった開発手法は同じだ。業務システムにもマルチデバイスが多用される昨今、4つのプラットフォームに対応したマルチデバイス用のFireMonkeyフレームワークを使わない手はない。

今回は、近年どのデバイスにも搭載されているカメラ機能に注目し、FireMonkeyフレームワークを利用したマルチデバイス対応の写真撮影アプリの開発手法を説明することで、FireMonkeyフレームワークの活用方法の一端を紹介する。

2. TActionを利用した写真撮影

TActionを利用することで、簡単に写真が撮影できる。TActionを利用するためにTActionListを用いるが、それ以外には、カメラを起動するTButton、撮影した画像を表示するTImageと、合計3つのコンポーネントをフォームに配置するだけで写真撮影アプリが実現できる｡実装方法も至ってシンプルである。作成方法を以下に紹介する。

作成手順

～TActionを利用した写真撮影～

①新規プロジェクトの作成

まず開発画面のメニューバーより、「ファイル|新規作成|マルチデバイスアプリケーション」を選択し、「空のアプリケーション」を選択する。新規プロジェクトが作成されるので、メニューバーより、「ファイル|すべて保存」にて任意の場所に保管する。

②コンポーネントの配置

TActionを利用するため、TActionListをフォームに配置する。TActionListは非ビジュアルコンポーネントのため、配置位置はどこでも問題ない。

次にカメラを実行するためのTButtonを配置する。ここでは画面レイアウト上、下部に配置するため、Alignプロパティを Bottomに設定し、StyleLookupプロパティを cameratoolbuttonに指定する。

このFireMonkeyのTButtonが持つStyleLookupプロパティは、リスト形式で設定でき、アイコンや見た目、カーソルなど、いろいろなスタイルを自動で適用できる便利なプロパティである。

最後に撮影した画像を表示するため

図1　コンポーネントの配置

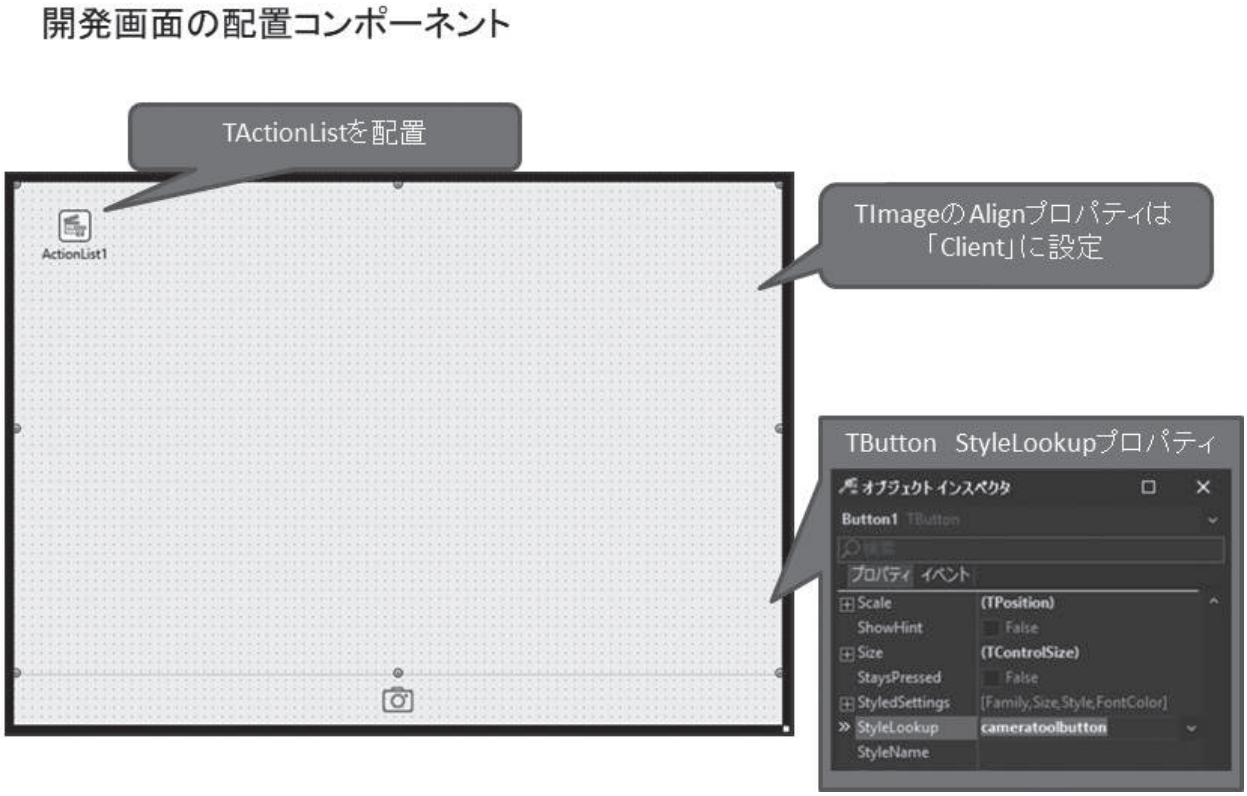


図2　Actionプロパティの設定

ソース1　作成手順～TActionを利用した写真撮影～① OnDidFinishTakingイベントの実装例

```
procedure TForm1.TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
begin
  // 撮影した写真を画面へ表示
  Image1.Bitmap.Assign(Image);
end;

end.
```

の TImage を配置する。画面レイアウトでは画面一面に表示するので、Align プロパティを Client に設定する。【図 1】

③ Action プロパティの設定

フォームに配置した TButton の Action プロパティのリストより「標準アクションの新規作成 | メディアライブラリ | TTakePhotoFromCameraAction」を選択する。これで画面のボタンをタッチすることでカメラが起動する。【図 2】

④イベントの設定

フォームに配置した TActionList をダブルクリック（もしくは「右クリック | アクションリストの設定」）し、カテゴリ＝メディアライブラリから TakePhotoFromCameraAction1 を選択する。オブジェクトインスペクタのイベントタブより、OnDidFinishTaking をダブルクリックし、実装部を作成してコーディングする。

OnDidFinishTaking イベントには、カメラで撮影された画像がその引数 Image（TBitmap 型）に格納されているため、配置した TImage コンポーネントにアサインするだけで実装完了である。【ソース 1】

⑤コンパイル

最後にコンパイルし、エラーがないことを確認し、「ファイル | すべて保存」よりプロジェクトを保管する。

以上の開発手順で、TAction を用いた写真撮影アプリが完成である。iOS や Android OS の端末に配布し実行すれば、開発したアプリで写真撮影ができることを確認できる。

ただし Windows OS 上で実行すると、カメラが起動されない。これについては残念ながら FireMonkey の Action がサポートされていないためである。しかし、Windows OS であっても ShellExecute メソッドを用いることでカメラは起動できる。先ほど作成したプロジェクトをもとに、以下に実装例を挙げる。

Windows OS でカメラを起動する実装例

条件付きコンパイル「{$IFDEF MSWINDOWS}」を用いて、Windows OS の場合は Windows API を uses 節に加えている。また、TButton の Action プロパティには TakePhotoFromCameraAction を紐づけず、On Click イベントにて Windows OS の場合は ShellExecute メソッドでカメラを起動し、それ以外の場合は TakePhotoFromCameraAction を実行するようにする。【ソース 2、ソース 3】

上記の実装例で、Windows OS 用の ShellExecute メソッドによりカメラを起動した場合は注意が必要である。これはあくまでカメラの起動であり、撮影したファイル名等の情報は取得できない。

TakePhotoFromCameraAction を利用した先の例では、OnDidFinishTaking イベントによって、撮影した画像が取得でき、それを利用して TImage に表示していた。しかし Windows OS 用に ShellExecute メソッドでカメラを起動した場合は、カメラロールを監視して、作成されるファイルのタイムスタンプで撮影された画像を判断するなどして画面に表示する必要があり、処理が煩雑になる。そもそも、実行する OS に合わせてコーディングすると手間も増えることになる。

その煩雑さや手間を解決するため、以下に FireMonkey 固有のコンポーネント「TCameraComponent」を紹介する。

3. TCamera Componentを利用した写真撮影

TCameraComponent は、FireMonkey 特有のカメラデバイスに対応するコンポーネントである。保持しているプロパティやイベントも少なく、わずかなコーディングでカメラデバイスを操作できる。

TCameraComponent を使う上でのポイントは、イベント OnSampleBufferReady である。まず TCameraComponent が保持する Active プロパティを True にすることで、カメラが起動する。このアクティブ状態（Active プロパティ＝True）の際に断続的に実行されるイベントが、OnSampleBufferReady である。

このイベントで、ビットマップに出力するために用意された SampleBufferToBitmap メソッドを利用して、アクティブ中のカメラの映像をイメージとして描画し、シャッターのタイミングで画像ファイルとして保存すれば、写真撮影アプリが作成できる。

それでは、実際に TCameraComponent を利用して写真撮影アプリを作成する。写真撮影アプリを作成するために用いる TCameraComponent 以外のコンポーネントは、カメラのシャッター用に TButton、カメラ画像を表示する TImage の 2 つである。

作成方法を以下に紹介する。なおこの作成手順は、先ほど紹介した「TAction を利用した写真撮影」の手順①のとおり、新規プロジェクトを作成した後のものとする。

作成手順

～ TCameraComponent を利用した写真撮影～

①コンポーネントの配置

TCameraComponent をフォームに配置する。TAction コンポーネントと同様、TCameraComponent は非ビジュアルコンポーネントなので配置位置はどこでも構わない。

次にカメラのシャッターとして、TButton を配置する。Align プロパティを Bottom に設定し、StyleLookup プロパティを cameratoolbutton に指定する。さらにカメラの映像を表示するための TImage を配置し、Align プロパティを Client に設定する。【図 3】

②イベントの設定

●フォーム生成時と破棄時

アプリの起動時にカメラ撮影を開始するため、フォームの OnCreate イベントにて TCameraComponent の Active プロパティを True にする。また、フォームの破棄時に生成時に起動したカメラ撮影を終了するため、OnCloseQuery イベントで Active プロパティを False にする。【ソース 4】

●ボタン押下時（シャッター）

カメラの映像からイメージへの描画をいったん停止し、映像を画像ファイルとして保存する。try-finally 構文で TCameraComponent の OnSampleBufferReady イベントをクリアし、再設定する間に、TImage の SaveToFile

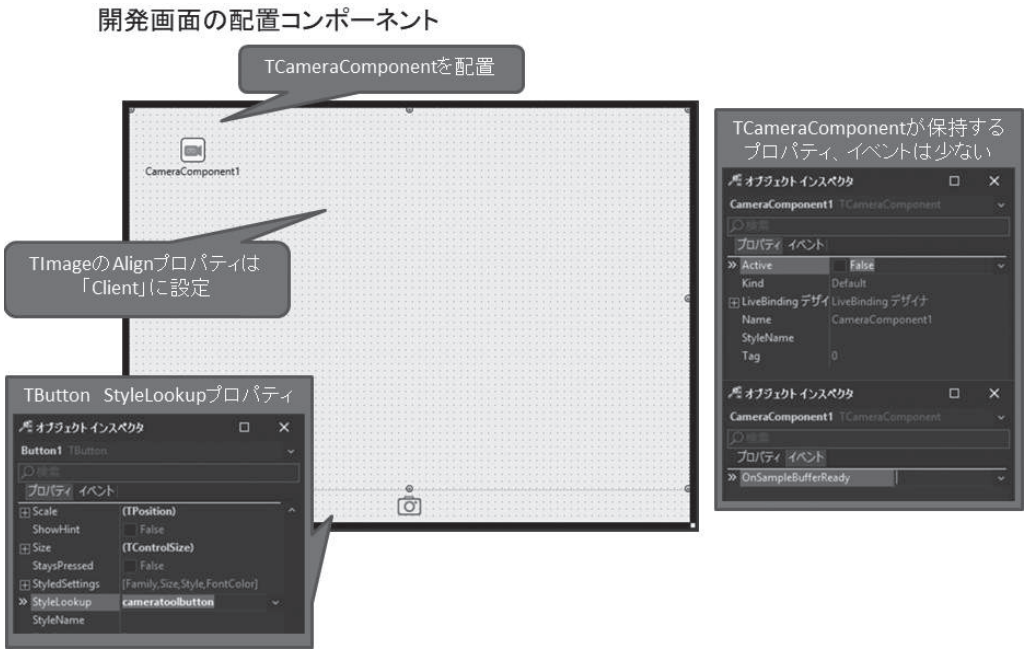
ソース2　作成手順～TActionを利用した写真撮影～② Windows OSでカメラを起動する実装例

```
unit Unit1;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Objects,
  FMX.Controls.Presentation, FMX.StdCtrls, System.Actions, FMX.ActnList,
  FMX.StdActns, FMX.MediaLibrary.Actions
{$IFDEF MSWINDOWS}
  , Winapi.Windows, Winapi.ShellAPI
{$ENDIF}
  ;

type
  TForm1 = class(TForm)
    Image1: TImage;
    ActionList1: TActionList;
    TakePhotoFromCameraAction1: TTakePhotoFromCameraAction;
    Button1: TButton;
    procedure TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
    procedure Button1Click(Sender: TObject);
  private
    { private 宣言 }
  public
    { public 宣言 }
  end;

var
  Form1: TForm1;

implementation        ソース３へ続く
```

ソース3　作成手順～TActionを利用した写真撮影～③ Windows OSでカメラを起動する実装例

```
{$R *.fmx}            ソース２の続き

procedure TForm1.Button1Click(Sender: TObject);
begin

{$IFDEF MSWINDOWS}
  // Windowsの場合、アクションがサポートされていない為、カメラアプリをShellExecute関数で起動
  // この場合、撮影したファイル名等が取得できないため、後続処理で使用したい場合に
  // カメラロールを参照してタイムスタンプ等で判断する必要がある
  ShellExecute(0, 'OPEN', PChar('microsoft.windows.camera:'), nil, nil, SW_SHOWMAXIMIZED);
{$ELSE}
  TakePhotoFromCameraAction1.Execute;
{$ENDIF}
end;

procedure TForm1.TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
begin
{$IF not Defined(MSWINDOWS)}
  // 撮影した写真を画面へ表示
  Image1.Bitmap.Assign(Image);
{$IFEND}
end;

end.
```

図3　コンポーネントの配置



メソッドを利用してファイルとして保存する。【ソース 5】

●カメラがアクティブ状態の時

カメラ映像の撮影はメインスレッドとは別のスレッドで行われるため、画面に配置した TImage へ画像ファイルを描画する際には、Synchronize メソッドを用いる。Synchronize は引数にメソッドが必要なので、「GetImage」というメソッドを別に作成した。

procedure として実装した GetImage メソッドでは、TCameraComponent の SampleBufferToBitmap メソッドを用いて、撮影中の映像を TImage へビットマップファイルとして読み込ませる。【ソース 6】

③コンパイル

最後にコンパイルし、エラーがないことを確認し、「ファイル | すべて保存」よりプロジェクトを保管する。

以上の開発手順で、TCameraComponent を用いた写真撮影アプリが完成である。iOS や Android OS の端末、カメラデバイスがある Windows OS でも、開発したアプリで写真を撮影できる。

4. デバイス搭載カメラの切り替え

前述の開発手順では、基本的な TCameraComponent の利用方法を紹介した。ここではさらに、プロパティについて補足する。

TCameraComponent 自体が持つプロパティは少ないが、開発画面のオブジェクトインスペクタでは「Kind」というプロパティを保持していることがわかる。これはデバイスが搭載しているどのカメラを使うかを指定するプロパティである。選択肢とカメラは、下記のようになる。

Default	：標準
FrontCamera	：前面カメラ
BackCamera	：背面カメラ

仕組みとして、TCameraComponent では使用するカメラを private 宣言の Device 変数で保持しており、Kind プロパティを FrontCamera や BackCamera に変更することで、そのデバイスの適切なカメラが自動的に選択され、Device 変数に格納される。

スマートフォンなどのモバイル端末ではこの選択肢で事足りるが、Windows 端末では USB 外付けカメラが利用できるため、その選択肢を持たない Kind プロパティでは対応できない。Device 変数自体に指定する必要がある。

前述のとおり、Device 変数は private 宣言のため、直接アクセスできないため、ヘルパークラスを作成し、搭載カメラの切り替えを実現する方法を以下に紹介する。これを CameraComponentHelper ユニットとする。

なお、そのデバイスが利用できる対象のカメラは、TCaptureDeviceManager.Current.GetDevicesByMediaType を参照することで取得できるので、合わせてカメラリストのクラスも管理することとする。【ソース 7 ～ 9】

CameraComponentHelper ユニットの概要

● TCameraComponentHelper クラス

TCameraComponent の helper として定義する。UseDevice プロパティを持ち、TCaptureDevice 型の Device プロパティに対して、参照ならびに設定を可能にする。

● TCameraList クラス

TCaptureDeviceList として、GetDevicesByMediaType より取得したカメラを TCaptureDevice 型の Item として、リスト形式で保持する。

CameraComponentHelper ユニットの使い方

具体的な CameraComponentHelper ユニットの使い方を説明する。先ほど紹介した「TCameraComponent を利用した写真撮影」のアプリへの追加実装を前提とし、カメラ切り替えボタンを押すことで、順次、利用可能なカメラに切り替わるものとする。

① uses 節への追加

CameraComponentHelper を uses 節に追加する。これで TCameraList、UseDevice が利用できるようになる。【ソース 10】

②コンポーネントの配置

カメラ切り替え用にボタンを配置する。画面の右下に配置することにする。また Text プロパティを「カメラ切り替え」とする。

③イベントの設定

●フォーム生成時と破棄時

フォームの生成時には、デバイスが利用可能なカメラを TCameraList を用いてリスト化する。コーディングは TCameraList 型のユニット内変数を private 宣言部で定義し、フォーム生成時に TCameraList を Create し、キャストする。

また、取得した利用可能なカメラの数を保持するためインデックス用の Integer 型を private 宣言部で合わせて定義する。フォームの破棄時には、生成した TCameraList 型のユニット内変数を破棄する。【ソース 11】

●ボタン押下時（カメラ切り替え）

カメラの切り替えは、CameraComponentHelper を uses 節に追加したことで利用できるようになった TCameraComponent の UseDevice に、TCameraList 型の Item を指定することで可能となる。またカメラを切り替える際は、TCameraComponent のアクティブ状態を停止してから行う。【ソース 12】

④コンパイル

最後にコンパイルし、エラーがないことを確認し、「ファイル | すべて保存」よりプロジェクトを保管する。

以上の実装で、デバイスが利用可能なカメラを切り替える機能が追加できた。実際に複数のカメラデバイスがあるモバイルや、USB 外付けカメラを接続した Windows 端末で動作を確認すると、カメラ切り替えボタンを押すたびに、撮影しているカメラが切り替わることが確認できる。

5. 画質の指定

TCameraComponent には、オブジェ

ソース4　作成手順～TCameraComponentを利用した写真撮影～①

フォーム生成時と破棄時

```
procedure TForm2.FormCreate(Sender: TObject);
begin
  // カメラの起動
  CameraComponent1.Active := True;
end;

procedure TForm2.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  // カメラ終了
  if CameraComponent1.Active then
  begin
    CameraComponent1.Active := False;
  end;
end;
```

ソース5　作成手順～TCameraComponentを利用した写真撮影～②

ボタン押下時（シャッター）

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  // カメラの起動状態を確認
  if not CameraComponent1.Active then Exit;

  // カメラからイメージへの描画をストップ
  CameraComponent1.OnSampleBufferReady := nil;
  try
    // 写真をJPG形式で保存
    Image1.Bitmap.SaveToFile('Picture.jpg');
  finally
    // カメラからイメージへの描画を再開
    CameraComponent1.OnSampleBufferReady := CameraComponent1SampleBufferReady;
  end;
end;
```

ソース6　作成手順～TCameraComponentを利用した写真撮影～③

カメラがアクティブ状態の時

```
procedure TForm2.CameraComponent1SampleBufferReady(Sender: TObject;
  const ATime: TMediaTime);
begin
  // メインスレッドでカメラの内容をイメージに描画
  TThread.Synchronize(TThread.CurrentThread, GetImage);
end;

procedure TForm2.GetImage;
begin
  if CameraComponent1.Active then
  begin
    // カメラの内容をイメージに描画
    CameraComponent1.SampleBufferToBitmap(Image1.Bitmap, True);
  end;
end;
```

クトインスペクタで指定できないPublicプロパティも保持している。その中でも写真を撮影したファイルの容量に関する画質の指定について、以下に紹介する。

TCameraComponentのQualityプロパティを用いると、デバイスに設定されている解像度を指定できる。その選択肢は解像度の高いものから順に、「PhotoQuality ＞ HighQuality ＞ MediumQuality ＞ LowQuality」となっている。

指定方法は至ってシンプルである。先ほどのカメラ切り替え用のソースを例として、フォームの生成時に中解像度に指定する方法を紹介する。

Qualityプロパティの指定方法

Qualityプロパティを指定する場合は、FMX.Media.クラスで定義されているTVideoCaptureQuality型からアクセスした各定数を利用する。中解像度は「MediumQuality」である。【ソース13】

これで保存した画像ファイルの用途に応じて、画質の指定が可能となる。

6. おわりに

本稿では、マルチデバイス用のFireMonkeyフレームワークの中からTCameraComponentの開発方法を説明し、少ないコーディングでデバイスのカメラ機能が利用できることを紹介した。

FireMonkeyにはその他にも、VCLフレームワークにない、特有の魅力的な機能やコンポーネントが多数存在する。本稿がきっかけとなり、FireMonkeyフレームワークを活用していただければ幸いである。

M

ソース7　CameraComponentHelperユニット①

TCameraComponentHelperユニット宣言部

```
unit CameraComponentHelper;

interface

uses
  System.Classes, FMX.Media;

type
  { TCameraComponentHelper }
  TCameraComponentHelper = class helper for TCameraComponent
  private
    procedure SetUseDevice(const Value: TCaptureDevice);
    function GetUseDevice: TCaptureDevice;
  public
    property UseDevice: TCaptureDevice read GetUseDevice write SetUseDevice;
  end;

  { TCameraList }
  TCameraList = class
  private
    FCameras: TStringList;
    function GetItems(Index: Integer): TCaptureDevice;
    function GetCount: Integer;
  public
    constructor Create;
    destructor Destroy; override;
    property Count: Integer read GetCount;
    property Items[Index: Integer]: TCaptureDevice read GetItems;
  end;

implementation
```

usesの節に「System.Classes」「FMX.Media」を追加する

TCameraComponentHelperクラス

TCameraListクラス

ソース8　CameraComponentHelperユニット②

TCameraComponentHelper実装部

```
{ TCameraComponentHelper }

function TCameraComponentHelper.GetUseDevice: TCaptureDevice;
begin
  // CameraComponentのデバイスを取得
  with Self do
  begin
    Result := FDevice;
  end;
end;

procedure TCameraComponentHelper.SetUseDevice(const Value: TCaptureDevice);
begin
  // CameraComponentへデバイスを設定
  with Self do
  begin
    FDevice := TVideoCaptureDevice(Value);
  end;
end;
```

TCameraList実装部

```
{ TCameraList }

destructor TCameraList.Destroy;
begin
  FCameras.Free;
  inherited;
end;
```

↓[ソース9]へ続く

ソース9　CameraComponentHelperユニット③

TCameraList実装部

↓[ソース8]の続き

```
{ TCameraList }

constructor TCameraList.Create;
var
  i: Integer;
  CaptureDeviceList: TCaptureDeviceList;
begin
  // カメラデバイスの一覧を保持
  FCameras := TStringList.Create;
  CaptureDeviceList := TCaptureDeviceManager.Current.GetDevicesByMediaType(TMediaType.Video);
  for i:=0 to CaptureDeviceList.Count-1 do
  begin
    FCameras.Add(CaptureDeviceList[i].Name);
  end;
end;

function TCameraList.GetCount: Integer;
begin
  Result := FCameras.Count;
end;

function TCameraList.GetItems(Index: Integer): TCaptureDevice;
begin
  // 指定したカメラデバイスを取得
  if (Index < 0) or (Index >= FCameras.Count) then
  begin
    Result := nil;
    Exit;
  end;
  Result := TCaptureDeviceManager.Current.GetDevicesByName(FCameras[Index]);
end;
```

ソース10　CameraComponentHelperユニットの使い方①

uses節への追加と変数定義

```
uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Media,
  FMX.Objects, FMX.Controls.Presentation, FMX.StdCtrls, Winapi.MMSystem,
  CameraComponentHelper;                                   uses節に「CameraComponentHelper」を追加する

type
  TForm3 = class(TForm)
    CameraComponent1: TCameraComponent;
    Image1: TImage;
    Rectangle1: TRectangle;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure CameraComponent1SampleBufferReady(Sender: TObject;
      const ATime: TMediaTime);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { private 宣言 }
    CameraList: TCameraList;
    FCurrentIndex: Integer;                    TCameraList型、Integer型の変数を定義する

    procedure GetImage;
  public
    { public 宣言 }
  end;
```

ソース11　CameraComponentHelperユニットの使い方②

フォーム生成時と破棄時

```
procedure TForm3.FormCreate(Sender: TObject);
begin
  // デバイス搭載カメラのリスト生成
  CameraList := TCameraList.Create;
  FCurrentIndex := CameraList.Count - 1;

  // カメラコンポーネントの設定
  CameraComponent1.Kind := FMX.Media.TCameraKind.Default;                  // 使用カメラ
  CameraComponent1.UseDevice := CameraList.Items[CameraList.Count - 1];
  CameraComponent1.Active := True;                                         // カメラ起動
end;

procedure TForm3.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  // カメラのリスト解放
  FreeAndNil(CameraList);

  // カメラ終了
  if CameraComponent1.Active then
  begin
    CameraComponent1.Active := False;
  end;
end;
```

変数CameraListを生成し
インデックスを初期設定する

UseDeviceの指定には
変数CameraListのItemを利用する

ソース12　CameraComponentHelperユニットの使い方③

ボタン押下時（カメラ切り替え）

```
procedure TForm3.Button2Click(Sender: TObject);
begin
  // カメラの切り替え
  CameraComponent1.Active := False; // 一旦カメラを停止

  Inc(FCurrentIndex);
  FCurrentIndex := FCurrentIndex mod CameraList.Count;
  CameraComponent1.UseDevice := CameraList.Items[FCurrentIndex]; // カメラ設定

  CameraComponent1.Active := True; // カメラ再開
end;
```

ソース13　Qualityプロパティの指定方法

フォーム生成時

```
procedure TForm3.FormCreate(Sender: TObject);
begin
  // デバイス搭載カメラのリスト生成
  CameraList := TCameraList.Create;
  FCurrentIndex := CameraList.Count - 1;

  // カメラコンポーネントの設定
  CameraComponent1.Kind := FMX.Media.TCameraKind.Default;                  // 使用カメラ
  CameraComponent1.UseDevice := CameraList.Items[CameraList.Count - 1];
  CameraComponent1.Quality := FMX.Media.TVideoCaptureQuality.MediumQuality; // 画質
  CameraComponent1.Active := True;                                         // カメラ起動
end;
```