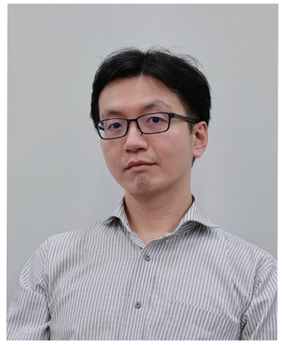
Delphi/400 11 Alexandriaによる最新モバイルアプリ開発術

株式会社ミガロ.
プロダクト事業部 技術支援課
佐田　雄一



略　歴

生年月日：1985年12月6日
最終学歴：2009年　甲南大学　経営学部卒業
入社年月：2009年04月　株式会社ミガロ. 入社
社内経歴：
2009年04月　システム事業部配属
2019年04月　RAD事業部（現プロダクト事業部）配属

現在の仕事内容：

Delphi/400を利用した
システム開発や保守作業の経験を経て、
現在はDelphi/400のサポート業務を担当している。

1.はじめに

Delphi/400がiOS・Androidを中心としたスマートデバイスに対応して以降、これまで弊社ではテクニカルセミナー、テクニカルレポート等で度々その時の最新トピックをご紹介してきた。

しかし、なにぶん技術の進歩はめざましく、iOS・Androidとも、1年前後といった非常に短いスパンで次期バージョンがリリースさている。そのたびに対応するDelphiのバージョンも変遷している。

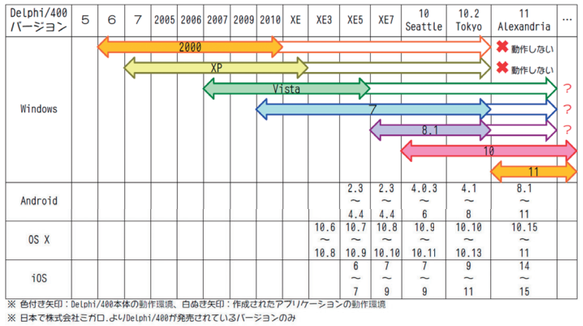
本稿では最新のDelphi/400 11 Alexandria（Update 2）を使用し、2022年9月時点での開発環境や開発手順、配布・運用のポイントを解説する。

2.Delphi/400 11 AlexandriaとFireMonkey対応OS

最新のDelphi/400 11 Alexandria（Update 2）では、以下のバージョンがサポートされている。**【図1】**

- Windows 11（Windows 7以上・Windows Server 2016以上）
- macOS 12 Monterey（OSX 10.15 Catalina以上）
- iOS 15（iOS 14以上）
- Android 12（Android 8.1以上）

図 1　サポートされているバージョン



※ 色付き矢印：Delphi/400本体の動作環境、白ぬき矢印：作成されたアプリケーションの動作環境
※ 日本で株式会社ミガロ.よりDelphi/400が発売されているバージョンのみ

日本国内におけるDelphi/400は、11 Alexandriaの前バージョンが10.2 Tokyoである。10.2 Tokyoにおいては iOSが11まで、Androidは8までのサポートとなっていた。新しいOSでも互換の範囲で動作できる部分はあったが、例えばAndroid 11以上では内部的なシステムパスの違いにより、10.2 Tokyoで作成・配置したアプリがエラーで起動できない。**【図2】**

図 2　10.2 Tokyo Android11のエラー

デバッグPC側ではこのエラーが表示されるが、
Android端末側では何も表示されずにアプリが落ちる

また、iOS・Androidとも、近年のバージョンでは32bitアプリのサポートが終了しており、最新のDelphi/400 11 Alexandriaにおいては64bitアプリとして作成するのが望ましい。（iOSでは32bitアプリの作成自体が対応終了している。）

各バージョンの対応状況はエンバカデロ社のDocwiki（公式Webヘルプ）
https://docwiki.embarcadero.com/PlatformStatus/ja
でも公表されているので、ご参照いただきたい。

なお、厳密にはiPad向けのOSは2019年にiOSから独立して「iPadOS」となっているが、Delphi・Delphi/400および本稿では同義として扱う。

3. Delphi開発環境におけるプラットフォームの選択

本章では、Delphi/400上で動作させる対象のデバイスOS（プラットフォーム）に合わせた開発を行うための準備について記載する。

＜Delphiインストール時の指定＞

Delphi 11 Alexandriaのインストール時に、「プラットフォーム選択」の画面でWindows・iOS・Androidの中から動作させる予定のOSに対応する項目にチェックを入れる。**【図3】**

Androidアプリの開発を行う場合は、同じ画面の「追加オプション」タブにて「Android SDK 25.2.5 - NDK r21」にもチェックを入れる必要がある。

図 3　プラットフォームの選択画面

なお、後から動作対象のプラットフォームを追加する必要が生じた場合でも、Delphiを再インストールする必要はない。Delphi開発画面のメニューから［ツール］プラットフォームの管理］を選択することで、**【図3】**のプラットフォームの選択画面が表示されるので、ここからチェックを切り替えて適用ボタンを押すことで対象項目のインストールが実行される。

＜開発画面でのプラットフォームを切り替え＞

開発画面で複数のプラットフォーム向けのプロジェクトを作成する場合、画面右側のプロジェクトマネージャにある「ターゲット　プラットフォーム」から対象のプラットフォームをダブルクリックすることで切り替えを行う。**【図4】**

プラットフォームを追加する場合は「ターゲット プラットフォーム」と記載の部分を右クリックすると「プラットフォームの追加」が選択できる。

図 4　プラットフォームの切り替え

4.FireMonkeyでのiOSアプリ開発

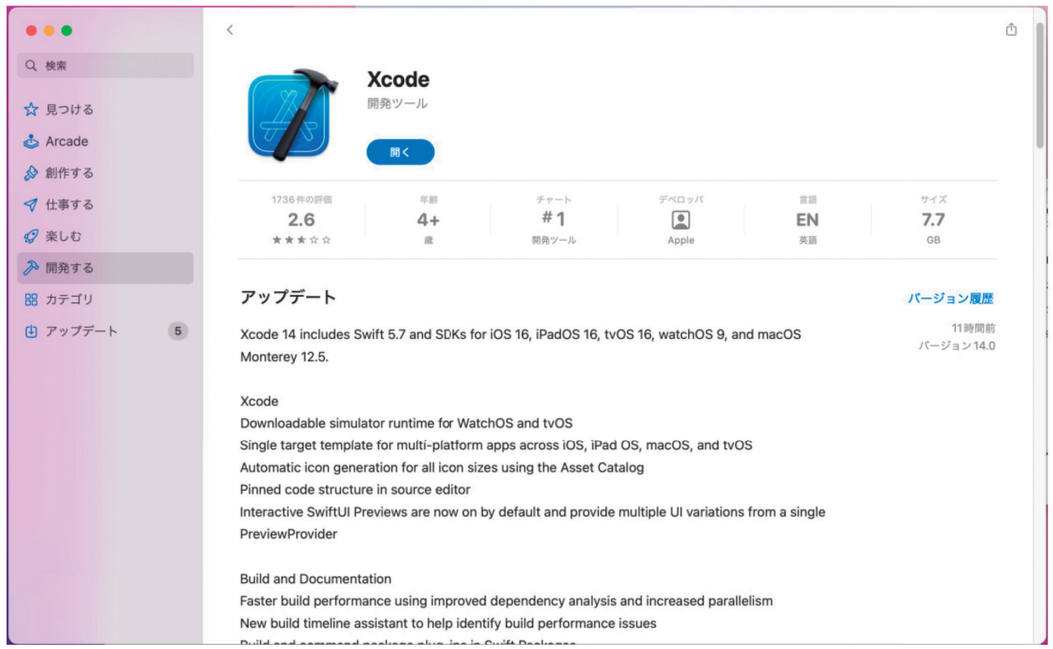
本章では、Delphi/400 11 Alexandriaを使用してiOSアプリケーションを開発する手順を紹介する。

＜必要となる環境＞
- Windows端末（64bitのWindows10以上、Delphi/400 11 Alexandria）
- Mac 端末（OSX 10.15 Catalina以上）
- iOS Developer Program（Xcode）
- iOS 実機（iPhone、iPadなど iOS 14～15）

＜Mac環境の構築＞
iOSの開発環境では、Delphi/400をインストールしているWindows端末とは別にMac端末が必要になる。Mac端末はOSX 10.15 Catalina以降をサポートしている。

＜Xcodeのインストール＞
Mac端末には最新のXcodeのインストールが必要になる。XcodeはMac App Storeからダウンロードしてインストールすることができる。**【図5】**

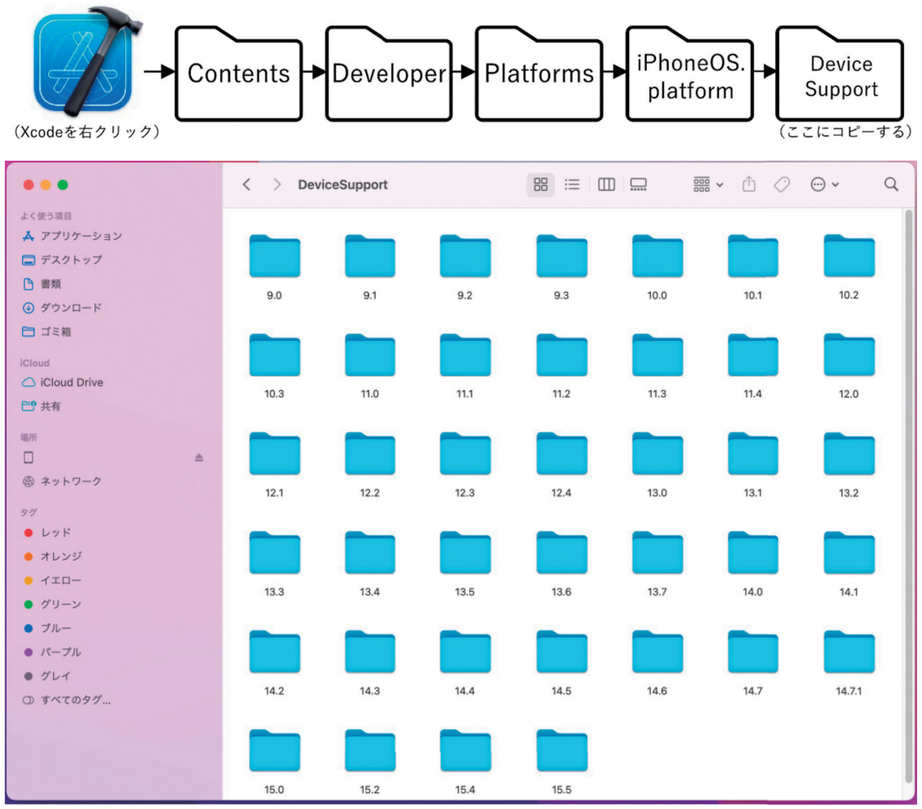
図 5　ストアでXcodeを入手



Mac端末とiPadなどの実機をUSB接続し、開発モードによる動作検証を行う場合、Xcodeのインストール後に対応するiOSバージョンのDevice Supportを導入する必要がある。
Mac端末にて、
https://github.com/iGhibli/iOS-DeviceSupport/tree/master/DeviceSupport
より実機のiOSと同じバージョンのzipファイルをダウンロードして解凍し、そのバージョン名のフォルダ（例：「15.5」）を得る。

次にアプリケーションの一覧にインストールされた「Xcode」を右クリックし、「パッケージの内容を表示」を選択する。
すると「Contents」フォルダが表示される。そこから「Developer」「Platforms」「iPhoneOS.platform」「DeviceSupport」の順にフォルダを遷移し、Device Supportの中にバージョン名のフォルダをコピーする。**【図6】**

図 6　DeviceSupportを適用する



<iOS Developer Program>
iOSアプリケーションはApple社の規約により、iOSへ配布するためには「iOS Developer Program」に加入する必要がある。
こちらの詳細については本稿では省略するため、最新情報はApple社サイトなどでご確認いただきたい。

<iOS実機の登録>
開発したiOSアプリケーションを動作検証する際にはMac上のiOSシミュレータも使用できるが、機種ごとに画面サイズの違いが存在するほか、端末にかかる負荷や処理速度はシミュレータでは計測しきれない。
そのため、実際の開発では実機でのテストが必須といえる。

iOS実機をテストで使用するにあたっては、実機をMacと直接USBケーブルで接続する（構成：開発）か、Mac上でキーチェーンアクセスを行って登録しておく（構成：アドホック）必要がある。
キーチェーンアクセスはMac上の「アプリケーション|ユーティリティ」メニューから作業できるので、Apple社のマニュアルを参考に登録作業を行う。

<接続プロファイルの作成>
Delphi/400の開発PCから、Macに接続する設定を作成する。Delphi開発画面の［ツール|オプション］からオプション画面を開き、配置メニューにある「接続プロファイルマネージャ」を選択する。【図7】

図 7　接続プロファイルマネージャ①

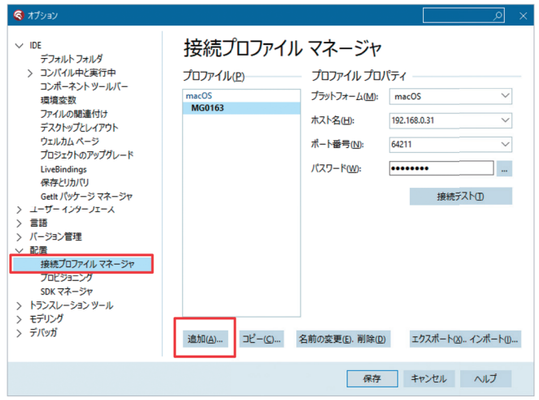
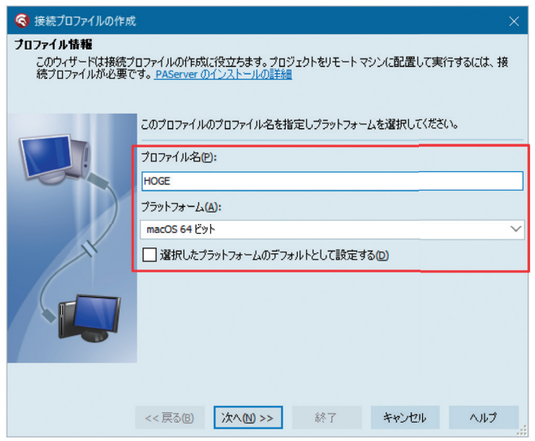
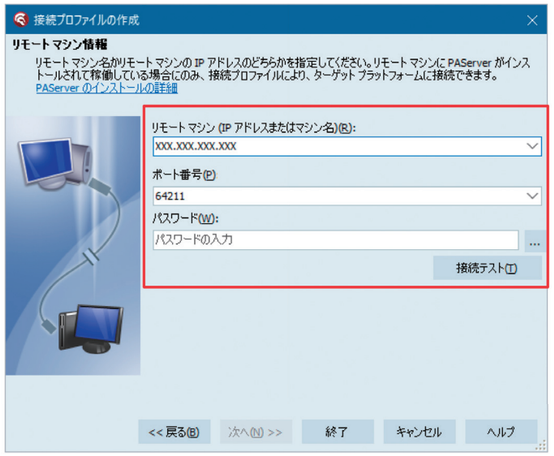


図 8　接続プロファイルマネージャ②



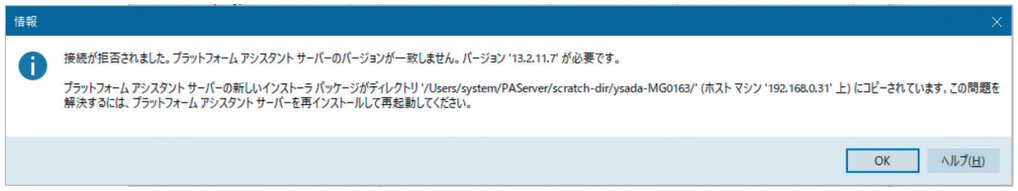
追加ボタンを押すと【図8】のようなダイアログが表示される。任意のプロファイル名とプラットフォームにはMacOSを設定して「次へ」を押す。プロファイル名は分かりやすい名前（iOS開発など）にしておくとよいだろう。
次に表示される設定画面【図9】でMac端末の接続情報を設定し、「接続テスト」ボタンを押して接続テストを行う。

図9　接続プロファイルマネージャ③



<PAServerのインストール>
接続テストで【図10】のようなダイアログが表示された場合、接続には成功したが、追加でMacに対応バージョンのPAServer（Platform Assistant Server）をインストールする必要がある。

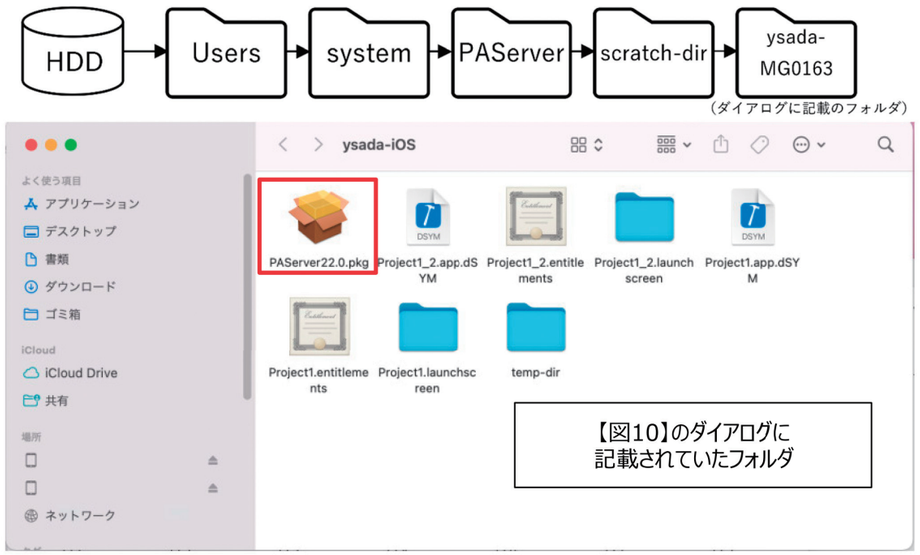
図10　接続プロファイルマネージャ④



PAServerとはDelphi/400のWindows開発PCからコンパイルしたアプリケーションを転送したり、デバッグを行ったりするソフトウェアで、Delphiのバージョンごとに PAServerのバージョンも異なっている。

【図10】のダイアログに記載されているディレクトリにインストーラーである「PAServer22.0.pkg」がコピーされているので、Mac端末でそのディレクトリを参照してインストーラーをダブルクリックし、PAServer 22.0をインストールする。**【図11】**

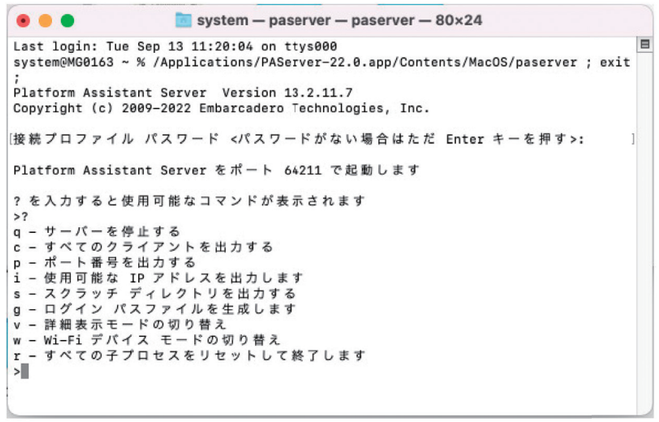
図11 PAServerのインストール



インストールが完了すると、メニューの「アプリケーション」に「PAServer-22.0」として登録されるので、これをダブルクリックして起動する。

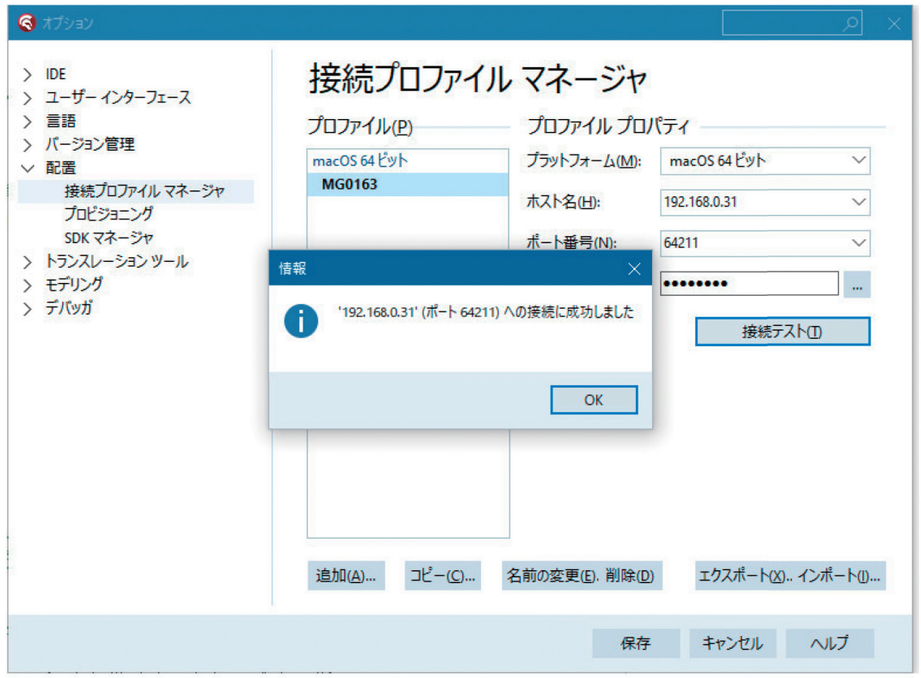
PAServerが起動するとコンソール画面で「Enterキーを押す」と表示されるので、[Enter]キーを押してサービスを開始する。**【図12】**

図12 PAServerの接続テスト（Mac）



サービスが開始した状態で、Delphi側で【図9】の「接続テスト」を押してみよう。**【図13】**のように接続成功のダイアログが表示されれば完了である。

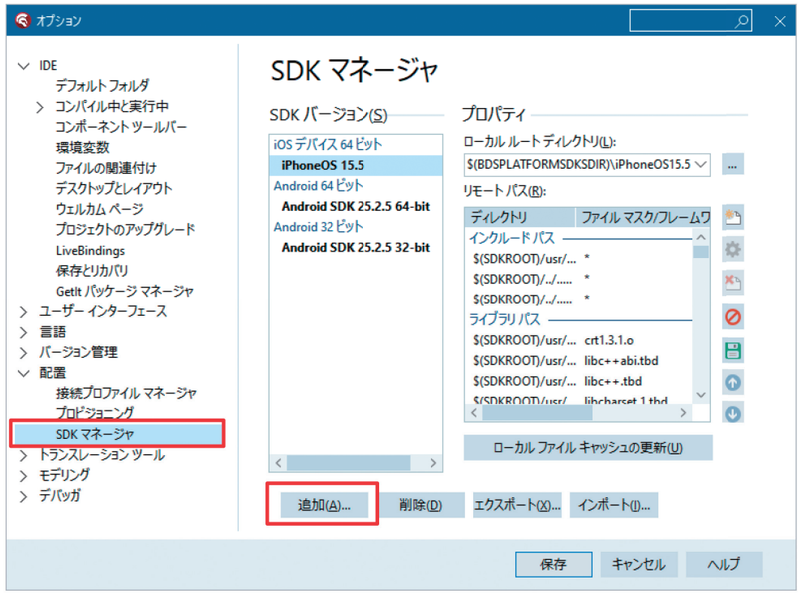
図13 PAServerの接続テスト（Delphi）



<SDKの取得>
Delphi/400上で対象のデバイスOSに合わせた開発を行うために、SDKの取り込みが必要になる。
Delphi本体のインストール時に、プラットフォームの選択で「Delphi iOS Enterprise」を含めている必要がある。

接続プロファイル同様にDelphi開発画面の[ツール|オプション]からオプション画面を開き、先程と同じ配置メニューにある「SDKマネージャ」を選択する。**【図14】**

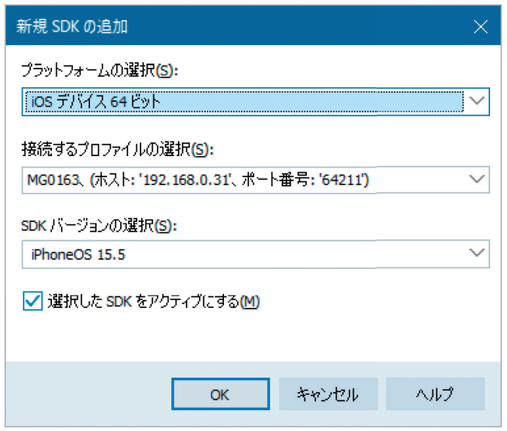
図 14　SDKマネージャ①



追加ボタンを押すと**【図15】**のダイアログ画面が表示される。
プラットフォームに「iOSデバイス」を選択して、接続するプロファイルには作成済のプロファイルを選択して設定する。

最後に接続先から対象のSDKバージョンが自動表示されるので、選択して「OK」ボタンを押下する。
これだけで、自動的にSDKがダウンロードされて組み込みが完了となる。

図 15　SDKマネージャ②



5.FireMonkeyでのAndroidアプリ開発

本章では、Delphi/400 11 Alexandriaを使用してAndroidアプリケーションを開発する手順を紹介する。
Delphi/400でAndroid向けのアプリケーションを開発する場合、開発環境はWindows内のみで構築することができる。
<必要となる環境>
●Windows端末（64bitのWindows10以上、Delphi/400 11 Alexandria）
●Android実機（Android 8.1以降）

<開発環境の構築>
Androidの開発環境は、iOSと異なり全てWindows端末上に構築できる。ただし、開発の対象となるAndroid実機のPC接続用ドライバは事前にインストールが必要となる。
Androidの機種によってインストール方法が異なるため、機種の製造元が提供する方法を確認してインストールを行う。
SDKのインストール方法については、第3章に記載の通りである。

6.クロスプラットフォームなアプリ開発例

Delphi/400では、ソースの一部を変更するだけで、冒頭で紹介した対応する各OSのモジュールをコンパイル可能である。
本章ではサンプルとして文字と画像をサーバーに送信するプログラム**【図16】**を作成しながら、各OSにおいてロジックを書き分ける必要がある箇所について解説する。

図 16-1　サンプルアプリの完成イメージ

図 16-2　サンプルアプリの完成イメージ

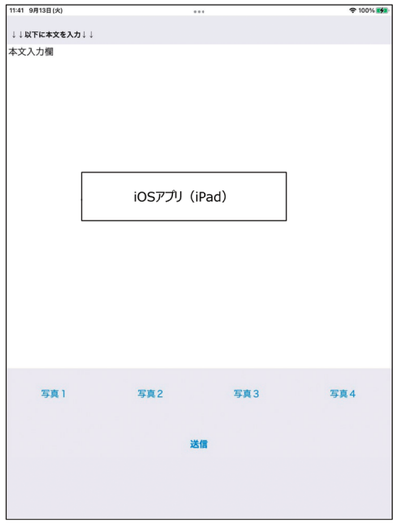
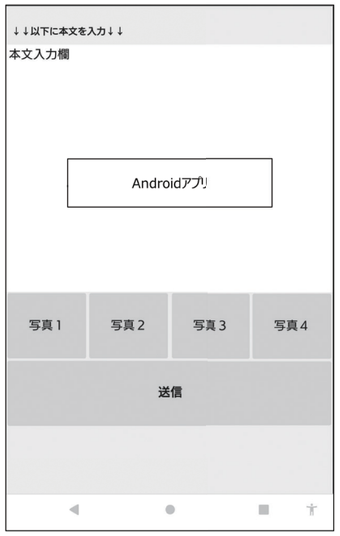


図 16-3　サンプルアプリの完成イメージ



<コンパイラ指令と条件付きコンパイルの活用>

Delphi/400では、条件シンボルを使用して、特定の条件下でコンパイルした場合にのみロジックを有効にできる方法が存在する。

例えば**【ソース1】**のように記述すれば、デバッグモード時、リリースモード時で処理を呼び分けることが可能である。

ソース 1

条件付きコンパイル①

```
{$IFDEF DEBUG}
ShowMessage('Debug is on.');   // デバッグでコンパイル時のみ動作する
{$ELSE}
ShowMessage('Debug is off.');  // デバッグ以外（リリース）でコンパイル時のみ動作する
{$ENDIF}
```

これを発展させて**【ソース2】**のように記述することで、iOS専用ロジック、Android専用ロジック、それ以外(Windows)専用ロジックを書き分けることが可能である。

ソース 2

条件付きコンパイル②

```
{$IFDEF IOS}
  // iOS向け処理をここに記述
{$ELSE}
{$IFDEF Android}
  // Android向け処理をここに記述
{$ELSE}
  // それ以外(Windows)向け処理をここに記述
{$ENDIF}
{$ENDIF}
```

この条件シンボルは通常のロジックのみならずconst宣言やuses節でも有効であるため、例えばconst値をターゲットプラットフォームごとに分岐させたり、ターゲットプラットフォームがiOS以外の場合ではコンパイルエラーになるような「iOSapi.～～～」という名前のユニットを、iOS向けにコンパイルした場合だけuses節で参照させたりすることも可能である。**【ソース3】**

条件シンボルは上記のデバッグとリリース、ターゲットプラットフォーム以外にもいくつかの種類が存在する。

条件シンボルの一覧については、Docwikiを参照いただきたい。

ソース 3

条件付きコンパイルを使用した条件分岐の例

```
// 通常ロジックの例
const
{$IFDEF IOS}
  cFMXAPP = 'iOSアプリから送信';                    // iOS向け処理
{$ELSE}
{$IFDEF Android}
  cFMXAPP = 'Androidアプリから送信';                // Android向け処理
{$ELSE}
  cFMXAPP = 'FireMonkeyアプリ版 Windowsから送信'; // それ以外(Windows)向け処理
{$ENDIF}
{$ENDIF}

// uses節の例
uses
  System.SysUtils, System.Types, System.UITypes, System.Classes,
  ・・・（中略）・・・,

{$IFDEF IOS} // iOSでのみ使用するユニット（他ではコンパイルエラー）
  iOSapi.Foundation, iOSapi.UIKit, MacAPI.Helpers, FMX.Platform.IOS,
{$ENDIF}

  // 以下権限設定用ユニット（後続の記述で使用）
  System.Permissions, FMX.DialogService;
```

<サンプルプログラムの作成：画面設計>

それでは、ここからサンプルプログラムを作成していく。

ファイルメニューから新規作成の「マルチデバイス　アプリケーション」を選択したら『空のアプリケーション』を選択する。

表示された新規フォームに、**【図17】**のようにTPanel・TMemo・TButton・TEdit・TLabel・TImageを配置し、Textプロパティを設定する。(VCLではTButtonやTLabelのキャプションはCaptionプロパティだが、FireMonkeyではTextプロパティとなっている)

図 17　設計画面のコンポーネント

FireMonkeyではコントロールのAlignがVCLよりも細かく設定可能である。
画面中段の「写真１～４」のボタンには、それぞれのAlignにMostLeft・Left・Right・MostRightを設定する。この設定を行ったうえで、フォームのOnShowイベント【ソース4】でボタンの幅が4等分されるように記述すると、実行する各プラットフォームとも4つのボタンが画面上【図17】のように4等分のサイズで配置される。

ソース4

サンプルの画面表示時処理

```
{*******************************************************************************
 目的: 画面表示時処理
*******************************************************************************}
procedure TForm1.FormShow(Sender: TObject);
var
  iWidth: Integer;
begin
  // 画像用ボタンの位置を調整
  iWidth := Trunc(Panel5.Width / 4);
  btnPH1A.Width := iWidth;
  btnPH2A.Width := iWidth;
  btnPH3A.Width := iWidth;
  btnPH4A.Width := iWidth;

  // 内部処理用ボタンを非表示（設計画面でVisibleをFalseにすると見えなくなる）
  btnPhotoWin.Visible := False;
end;
```

＜サンプルプログラムの作成：画像の読み込み＞
「写真１～４」のボタンは、押下時に端末内の画像を選択して読み込ませるために使用する。読み込ませた画像は対応するTImageに内部保持させるようにコーディングを行う。
その機能を実装するため、画面にTOpenDialogを配置する。また画面に非表示ボタンとして配置していた「btnPhotoWin」の押下時処理を【ソース5】のように記述する。TOpenDialogのコーディングや挙動についてはVCLと同様のため、本稿では割愛する。

ソース5

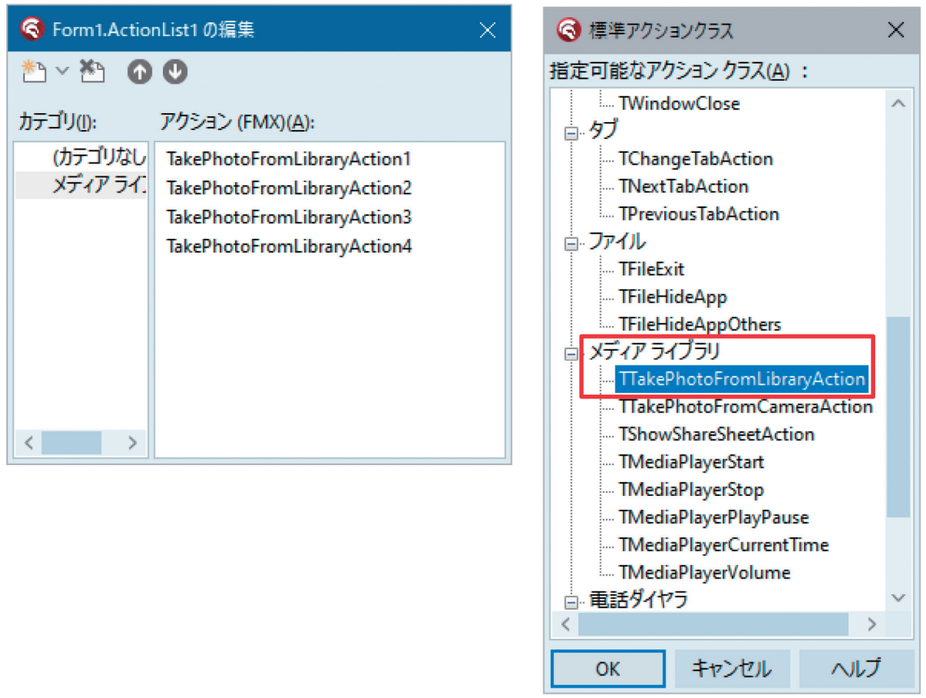
Windowsの画像読込処理

```
{*******************************************************************************
 目的: 写真１～４ボタン 押下時処理（Windows）
*******************************************************************************}
procedure TForm1.btnPhotoWinClick(Sender: TObject);
var
  img: TImage;
begin
  // 対象コンポーネントの指定
  if (Sender = btnPH1A) then img := Image1;
  if (Sender = btnPH2A) then img := Image2;
  if (Sender = btnPH3A) then img := Image3;
  if (Sender = btnPH4A) then img := Image4;

  // 画像を読み込む
  OpenDialog1.FileName := '';
  if OpenDialog1.Execute then
  begin
    img.MultiResBitmap.Items[0].Bitmap.LoadFromFile(OpenDialog1.FileName);
  end;
end;
```

さて、ボタンの名前で既にお気付きの方もいるかもしれないが、この「btnPhotoWin」およびTOpenDialogはWindowsで動作する際にしか使用できない。iOSやAndroidにおいては「OpenDialog1.Execute」といったロジックを記述しても、非対応のため何も動作しない。
そこでiOS・Android向けに同じ写真の取り込みを行うためには、Actionを使用する。画面にTActionListを配置し、その中のアクション一覧の右クリックメニューにある「標準アクションの新規作成」から「TTakePhotoFromLibraryAction」を選択して作成する。今回は写真ボタンを4つ使用するため、アクションも4つ作成する。【図18】

図 18　TTakePhotoFromLibraryAction追加

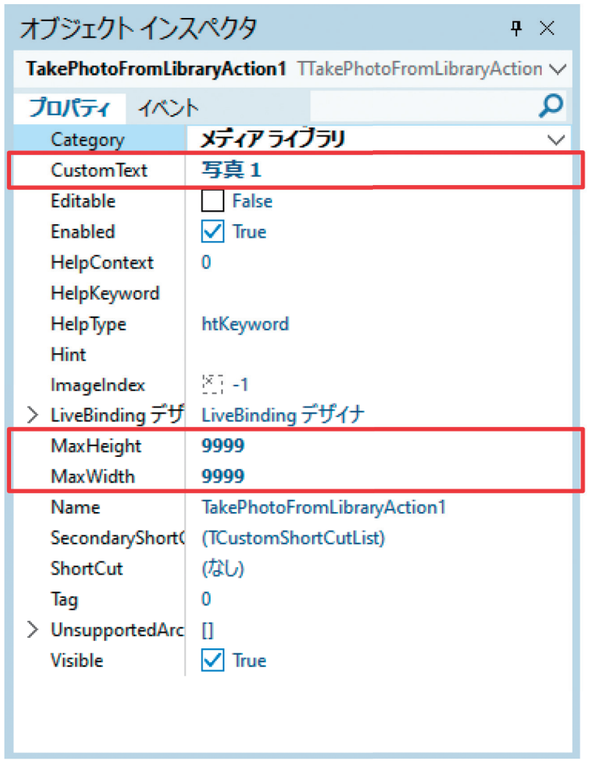


オブジェクトインスペクタにて、作成した各TTakePhoto FromLibraryActionのプロパティに

- CustomText＝ボタンのキャプション（「写真１」など）
- MaxHeight＝9999
- MaxWidth＝9999

を設定する。**【図19】**

図 19　TTakePhotoFromLibraryAction設定



MaxHeightとMaxWidthの値は、初期値のままだとライブラリから取り込んだ画像の縦横が縮小されてしまうため、その対策で変更している。実際には運用上の最大サイズを指定すればよい。

ここまで「写真１～４」のボタン押下時の処理（イベント）を直接設定してこなかったことにお気付きだろうか。
これは対象プラットフォーム（iOS・Android・Windows）ごとに処理を呼び分けるためである。
画面のOnCreateイベントに、本項の冒頭で紹介したコンパイラ指令を使って、ロジックでボタン押下処理の呼び分けを行う。ここではWindows向けにコンパイル時はTOpenDialogを表示し、iOSやAndroidではTTakePhotoFromLibraryActionを使うように指定している。**【ソース6】【ソース7】**

ソース 6

画面表示時処理

```
{*******************************************************************************
 目的: 画面生成時処理
*******************************************************************************}
procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF IOS}     // iOS向け処理
  // 写真ボタンのアクション指定
  btnPH1A.Action  := TakePhotoFromLibraryAction1;
  btnPH2A.Action  := TakePhotoFromLibraryAction2;
  btnPH3A.Action  := TakePhotoFromLibraryAction3;
  btnPH4A.Action  := TakePhotoFromLibraryAction4;
{$ELSE}
{$IFDEF Android} // Android向け処理
  // 写真ボタンのアクション指定
  btnPH1A.Action  := TakePhotoFromLibraryAction1;
  btnPH2A.Action  := TakePhotoFromLibraryAction2;
  btnPH3A.Action  := TakePhotoFromLibraryAction3;
  btnPH4A.Action  := TakePhotoFromLibraryAction4;
{$ELSE}          // それ以外(Windows)向け処理
  btnPH1A.OnClick := btnPhotoWinClick;
  btnPH2A.OnClick := btnPhotoWinClick;
  btnPH3A.OnClick := btnPhotoWinClick;
  btnPH4A.OnClick := btnPhotoWinClick;
{$ENDIF}
{$ENDIF}
end;
```

ソース7

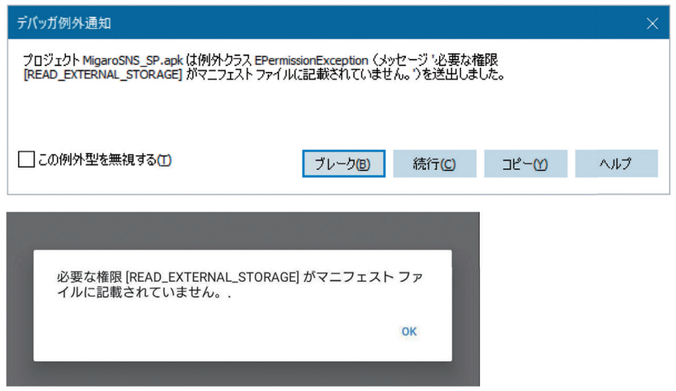
iOS/Androidの写真ボタンアクション押下時処理

```
宣言部に以下を宣言（詳細は【ソース8】）
    procedure SelectPhoto(img: TImage; bmp: TBitmap);

実装部に以下を記述
{*******************************************************************************
 目的: 写真１～４ボタン 押下時処理（iOS/Android）
*******************************************************************************}
procedure TForm1.TakePhotoFromLibraryAction1DidFinishTaking(Image: TBitmap);
begin
  SelectPhoto(Image1, Image); // 権限付与と写真の読み込み
end;
procedure TForm1.TakePhotoFromLibraryAction2DidFinishTaking(Image: TBitmap);
begin
  SelectPhoto(Image2, Image); // 権限付与と写真の読み込み
end;
procedure TForm1.TakePhotoFromLibraryAction3DidFinishTaking(Image: TBitmap);
begin
  SelectPhoto(Image3, Image); // 権限付与と写真の読み込み
end;
procedure TForm1.TakePhotoFromLibraryAction4DidFinishTaking(Image: TBitmap);
begin
  SelectPhoto(Image4, Image); // 権限付与と写真の読み込み
end;
```

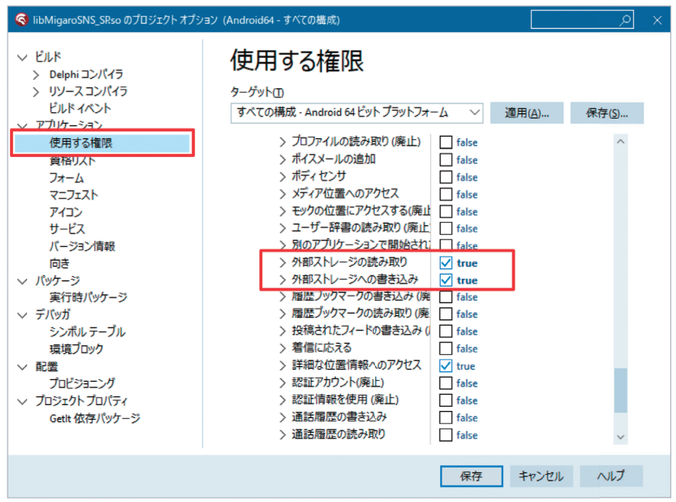
ここまででプログラムをコンパイルして実行すると、Windowsでは正常に動作するが、Android端末ではアクセス権限が不足している旨のエラーになる。**【図20】**

図 20　Android権限不足時のエラー



そこで**【図21】**の権限設定を行って外部ストレージへの読み取りと書き込みを許可するとともに、**【ソース8】**のように権限を付与するロジックを記載する。この設定およびロジックによって、アプリ起動時に権限を付与しても良いかスマートデバイス側で確認が表示されるようになり、デバイス側で承認されればその中の画像にアクセスできるようになる。

図 21　ファイルの読み書き権限設定



ソース8

スマートデバイスに権限を許可させる

```
※一部引用元：https://blogs.embarcadero.com/ja/?p=75042

宣言部のconstに以下を宣言（別処理でも使用するため宣言部に記述）
const
  perm_tbl: array[1..2] of string = (
      'android.permission.WRITE_EXTERNAL_STORAGE',
      'android.permission.READ_EXTERNAL_STORAGE'
  );

実装部に以下を記述
{*******************************************************************************
 目的: 権限付与と写真の読み込み
*******************************************************************************}
procedure TForm1.SelectPhoto(img: TImage; bmp: TBitmap);
begin
  //外部ストレージへの読み書きのPermissionを得る
  // (usesに'System.Permissions'が必要)
  PermissionsService.RequestPermissions(
    [ perm_tbl[1], perm_tbl[2] ],
    procedure(const APermissions: TClassicStringDynArray;
              const AGrantResults: TClassicPermissionStatusDynArray)
    begin
      if PermissionsService.IsPermissionGranted(perm_tbl[1]) and
         PermissionsService.IsPermissionGranted(perm_tbl[2]) then
      begin
        // TImageに画像を読み込ませる
        img.MultiResBitmap.Items[0].Bitmap.Assign(bmp);
      end
      else
      begin
        ShowMessage('外部ストレージへの読み書きの権限がありません。');
      end;
    end
  );
end;
```

権限が正しく設定され、画像を読み込むことができれば、読み込んだ画像を画面に表示できるだろう。

<サンプルプログラムの作成：サーバーへの更新>

読み込み完了後は、読み込んだ内容をサーバーに更新する処理を記述していく。

今回はIndyを使ってFTPサーバーへ転送する処理とする。接続先をIBM iに指定すれば、IBM iのIFS領域に転送することも可能である。なお本稿の主旨から外れるため、FTP送信されたファイルをサーバー側のアプリケーションで取り扱う箇所については省略する。

画面にTIdFTP(IdFTP)を配置し、送信ボタンのOnClickイベントおよび更新メソッドに**【ソース9】【ソース10】**のように記述する。

ソース9

送信ボタンのクリック時処理

```
宣言部に以下を宣言（詳細は【ソース10】）
    procedure FileWriteProc;

実装部に以下を記述
{*******************************************************************************
 目的: 送信ボタン 押下時処理
*******************************************************************************}
procedure TForm1.btnPostClick(Sender: TObject);
begin
  // 氏名のチェック
  if (cmbUSER.ItemIndex <= 0) then
  begin
    ShowMessage('名前が選択されていません。');
    Abort;
  end;
  edtUSER.Text := cmbUSER.Items[cmbUSER.ItemIndex];

  // 本文のチェック
  if (Length(AnsiString(Memo1.Lines.Text)) > 450) then
  begin
    ShowMessage('本文が長すぎます。450バイト以内で入力して下さい。');
    Abort;
  end;

  //外部ストレージへの読み書きのPermissionを得る
  // (usesに'System.Permissions'が必要)
  PermissionsService.RequestPermissions(
    [ perm_tbl[1], perm_tbl[2] ],
    procedure(const APermissions: TClassicStringDynArray;
              const AGrantResults: TClassicPermissionStatusDynArray)
    begin
      if PermissionsService.IsPermissionGranted(perm_tbl[1]) and
         PermissionsService.IsPermissionGranted(perm_tbl[2]) then
      begin
        FileWriteProc; // ファイル作成処理
      end
      else
      begin
        ShowMessage('外部ストレージへの読み書きの権限がありません。');
      end;
    end
  );
end;
```

ソース10

ファイル送信時処理

```
※一部引用元：http://oms.la.coocan.jp/coding/sample_d18.html
{*******************************************************************************
 目的: ファイルを送信処理
   (usesに'System.IOUtils'が必要)
*******************************************************************************}
procedure TForm1.FileWriteProc;
const
  cFTPLib = '/TECHREPORT2022/';  // FTPの転送先フォルダ階層
var
  fPath: string;
  fname: array [0..4] of string;
  fImg: array [0..4] of TImage;
  msg: string;
begin
  fTime := FormatDateTime('HHNNSS', Now);

{$IFDEF IOS}       // iOS向け処理
  fPath := System.IOUtils.TPath.GetTempPath;
{$ELSE}            // Android/Windows向け処理
  fPath := System.IOUtils.TPath.GetSharedDocumentsPath;
{$ENDIF}

  // 保存先フォルダがまだ無い場合は生成
  ForceDirectories(fPath);

  fname[0] := System.IOUtils.TPath.Combine(fPath, fTime + '_' + edtUSER.Text + '_0.TXT');
  fname[1] := System.IOUtils.TPath.Combine(fPath, fTime + '_' + edtUSER.Text + '_1.JPG');
  fname[2] := System.IOUtils.TPath.Combine(fPath, fTime + '_' + edtUSER.Text + '_2.JPG');
  fname[3] := System.IOUtils.TPath.Combine(fPath, fTime + '_' + edtUSER.Text + '_3.JPG');
  fname[4] := System.IOUtils.TPath.Combine(fPath, fTime + '_' + edtUSER.Text + '_4.JPG');

  fImg[0]  := Image1;
  fImg[1]  := Image2;
  fImg[2]  := Image3;
  fImg[3]  := Image4;

  msg :='送信します。よろしいですか？';
  TDialogService.MessageDialog(
    msg,  // ダイアログ本文
    TMsgDlgType.mtConfirmation, // ダイアログのタイプ
    mbYesNo,                    // 表示するボタンの集合
    TMsgDlgBtn.mbYes,           // デフォルトフォーカスボタン
    0,                          // HelpContext
    procedure(const AResult: TModalResult)
    var
      i: Integer;
    begin
      if AResult=mrYes then
      begin
        // FTP切断
        if IdFTP.Connected then
        begin
          IdFTP.DisConnect;
        end;

        // FTP接続先設定
        IdFTP.Host     :='XXXXXXXX'; // FTP接続サーバー名またはIPアドレス
        IdFTP.Username :='XXXXXXXX'; // FTP接続ユーザー
        IdFTP.Password :='XXXXXXXX'; // FTP接続パスワード
        IdFTP.ListenTimeout := 10000; // FTP接続時の待機時間
        IdFTP.Passive  := False;      // FTP接続時のPassive設定（接続先に合わせる）
        try
          try
            IdFTP.Connect;
          except
            on E: Exception do
            begin
              // 接続エラー時のメッセージ
              msg :='接続に失敗しました。' + sLineBreak + E.ClassName + ' ' + E.Message;
              ShowMessage(msg);
              Abort;
            end;
          end;

          // テキストファイルを保存
          Memo1.Lines.SaveToFile(fname[0], TEncoding.UTF8);

          // 画像ファイルがあれば保存（画像の有無はImageのHeightで判断する）
          for i := 0 to 3 do
          begin
            if (fImg[i].MultiResBitmap.Items[0].Bitmap.Height > 0) then
            begin
              fImg[i].MultiResBitmap.Items[0].Bitmap.SaveToFile(fname[i+1]);
            end;
          end;

          // FTPアップロード先の設定
          try
            IdFTP.ChangeDir(cFTPLib);   // カレントディレクトリ変更
          except
            IdFTP.MakeDir(cFTPLib);     // 転送先フォルダが存在しない場合は作成
            IdFTP.ChangeDir(cFTPLib);   // もう一度カレントディレクトリを変更
          end;

          // 上書きアップロード
          try
            IdFTP.TransferType := ftBinary; // テキスト/バイナリとも送受信可能にする
            IdFTP.Put(fname[0], ExtractFileName(fname[0]));   // 本文のテキストを送信
            if (FileExists(fname[1])) then IdFTP.Put(fname[1], ExtractFileName(fname[1]));
            if (FileExists(fname[2])) then IdFTP.Put(fname[2], ExtractFileName(fname[2]));
            if (FileExists(fname[3])) then IdFTP.Put(fname[3], ExtractFileName(fname[3]));
            if (FileExists(fname[4])) then IdFTP.Put(fname[4], ExtractFileName(fname[4]));
          except
            on E: Exception do
            begin
              // 送信エラー時のメッセージ
              msg :='送信に失敗しました。' + sLineBreak + E.ClassName + ' ' + E.Message;
              ShowMessage(msg);
              Abort;
            end;
          end;

        finally
          IdFTP.DisConnect;
        end;

        ShowMessage('送信しました。反映までしばらくお待ちください。');
      end;
    end
  );
end;
```

今回のサンプルではMemo1に記入されたテキストと画像を4枚まで添付できるように作成しており、テキストと画像の計最大5ファイルを一時フォルダに保管した後、それをFTP送信する。FTP送信ロジック自体は**【ソース10】**を見てもVCLと比較してもほとんど変わらない。これこそがDelphiの強みである。

ここまでコーディングを進めた上で、Windows・iOS・Androidの各端末でコンパイル～配置～実行を行うと、本項の最初に示した**【図16】**のようなアプリが完成する。
実際に起動して送信が完了すると**【図22】**のように完了メッセージが表示され、FTPサーバー側には**【図23】**のようにファイルが送信されている。

ここまでコーディングを進めた上で、Windows・iOS・Androidの各端末でコンパイル～配置～実行を行うと、本項の最初に示した**【図16】**のようなアプリが完成する。
実際に起動して送信が完了すると**【図22】**のように完了メッセージが表示され、FTPサーバー側には**【図23】**のようにファイルが送信されている。

図 22-1　サンプルアプリの実行イメージ

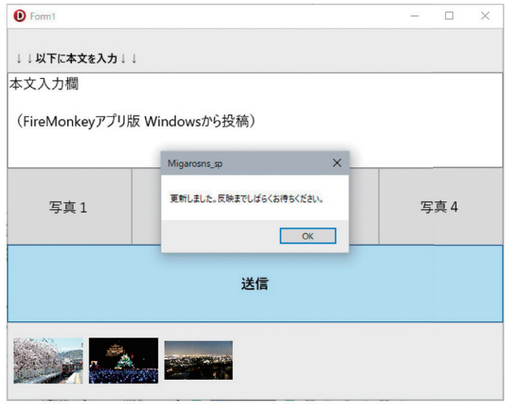
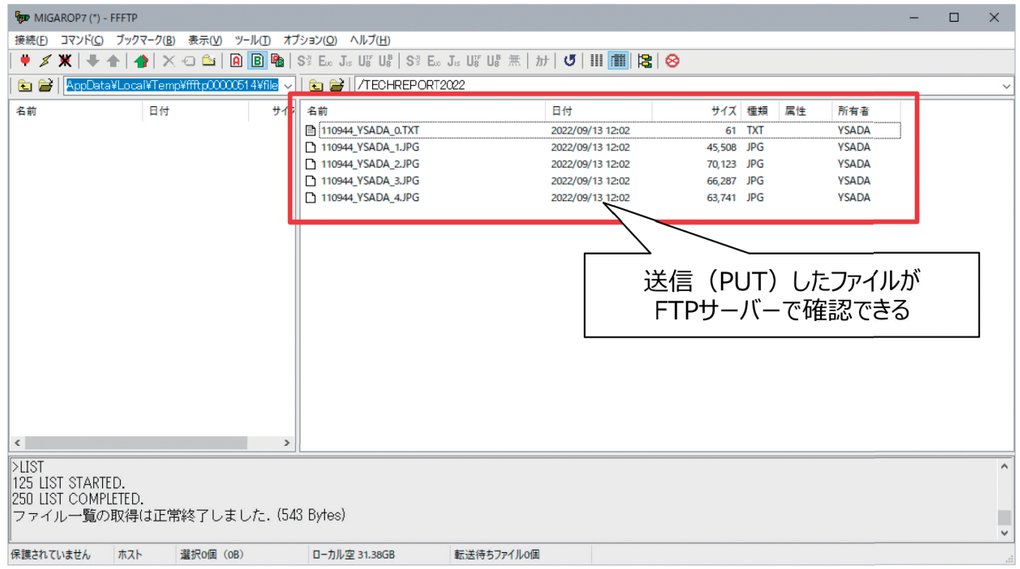


図 22-2　サンプルアプリの実行イメージ

図 22-3　サンプルアプリの実行イメージ

図 23　FTPサーバー側の送信結果



7.まとめ

iOS・Androidとも技術の進歩がめざましく、バージョン番号もどんどん上がっている。しかしDelphi/400側もそれに追従しており、最新のOSへの対応が進んでいる。
本稿をもとに、Delphi/400が得意とするWindows・iOS・Androidのクロスプラットフォーム開発をご検討いただければ幸いである。