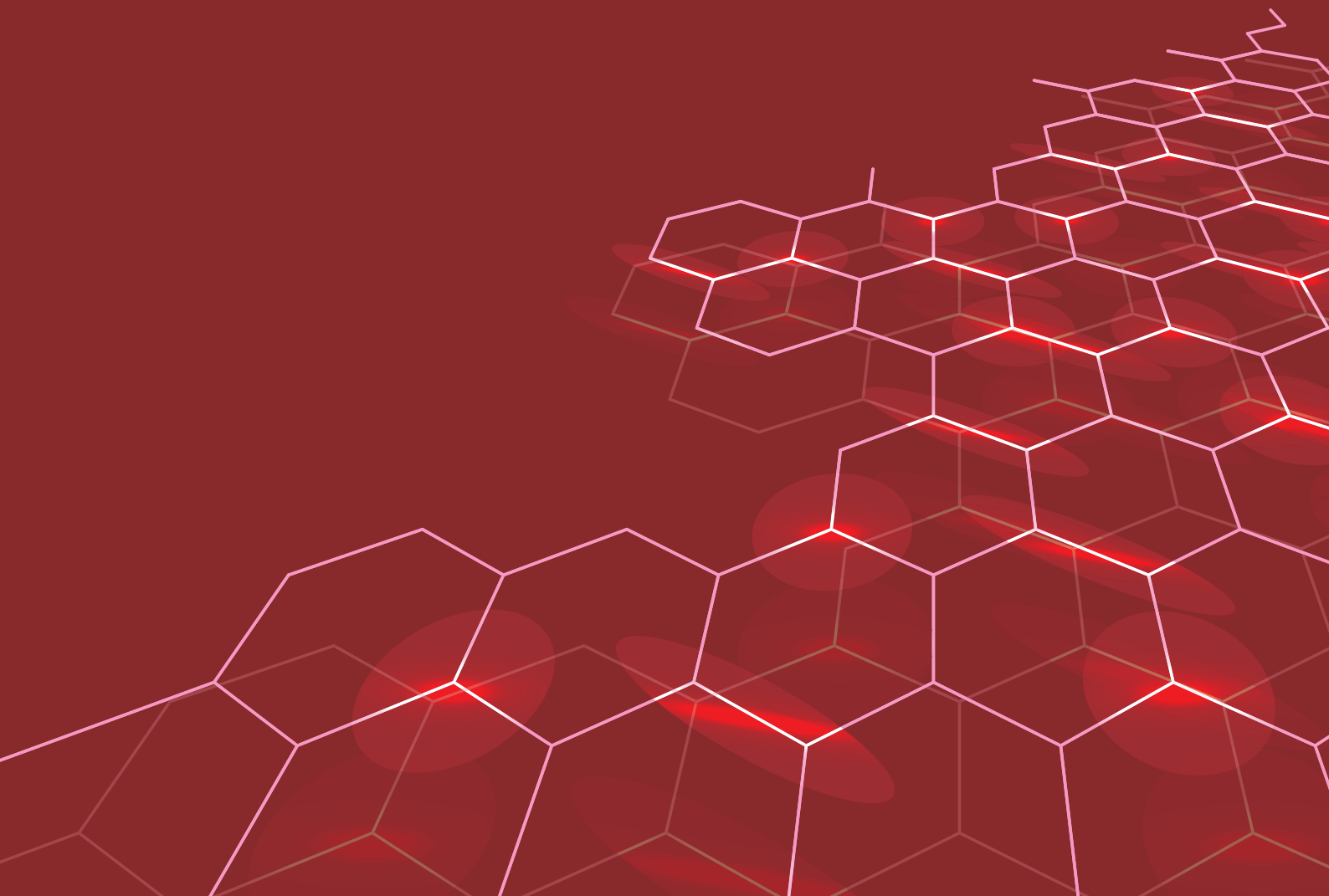
MIGARO. TECHNICAL REPORT 2022

ミガロ. テクニカルレポート 2022年

No.15

株式会社ミガロ.



MIGARO.
Technical Report 2022

ミガロ. テクニカルレポート 2022年

No.15

CONTENTS

目次

ごあいさつ

いつもミガロ. 製品をご愛用いただき誠にありがとうございます。

「ミガロ.テクニカルレポート」は、ミガロ.製品を使った開発に関する技術情報をお届けする論文集で、このたび第15号を発刊する運びとなりました。15年の長きにわたり継続できたことは、お客様からの論文ご寄稿を始めとする温かいご支援の賜物と心より感謝しております。今年はコロナ禍が継続する中、お客様論文の募集は行わず、弊社SE論文のみといたしました。なにとぞご了承を賜りますようお願い申し上げます。

さて、ミガロ.テクニカルレポートを発刊した2008年は、Power Systems（ハードウェア）、IBM i（OS）が命名され発表された記念すべき年になります。そして現在も、安定性と堅牢性にすぐれた IBM i は多くのユーザー企業に信頼され、愛用されています。IBM i が「レガシー」という評価を受けがちということを時々耳にしますが、ハード+OSの性能は常に向上しており、入力画面（UI）の印象によるところが大きいと感じております。

このUIの課題に対して、弊社では、Delphi/400、Valence、SP4iの3種類の開発ツールをとおして、お客様ご自身で効率よくIBM i のGUI、Web、モバイルアプリを開発いただくご提案をしております。そして、その実現のために、テクニカルサポートや技術情報の提供に引き続き努力をしてまいります。

今回のSE論文はDelphi/400、Valence、SP4iのすべての製品をテーマとして含んでいます。さまざまな電子機器が身の回りに溢れる中で、今回は特にモバイル端末などPC以外のデバイスをテーマとした論文が中心となりました。Delphi/400、Valenceそれぞれで、モバイル開発の基本的な手法をご紹介しています。また、SP4i では、タブレット等で利用されることの多い電子サインを採りあげました。別のDelphi/400論文では、モバイルカメラの利用、デジタルサイネージ用アプリ開発についてご紹介しています。さまざまなテクニックを開発に活用いただくために、各論文は詳細な実装方法までご紹介しています。本レポートが少しでも皆様の開発・保守のお役に立てば幸いです。

最後に、ミガロ.テクニカルレポート第15号を発刊するにあたり、多くのお客様・パートナー様にご支援、ご協力をいただきましたことを、この場をお借りして厚く御礼申し上げます。

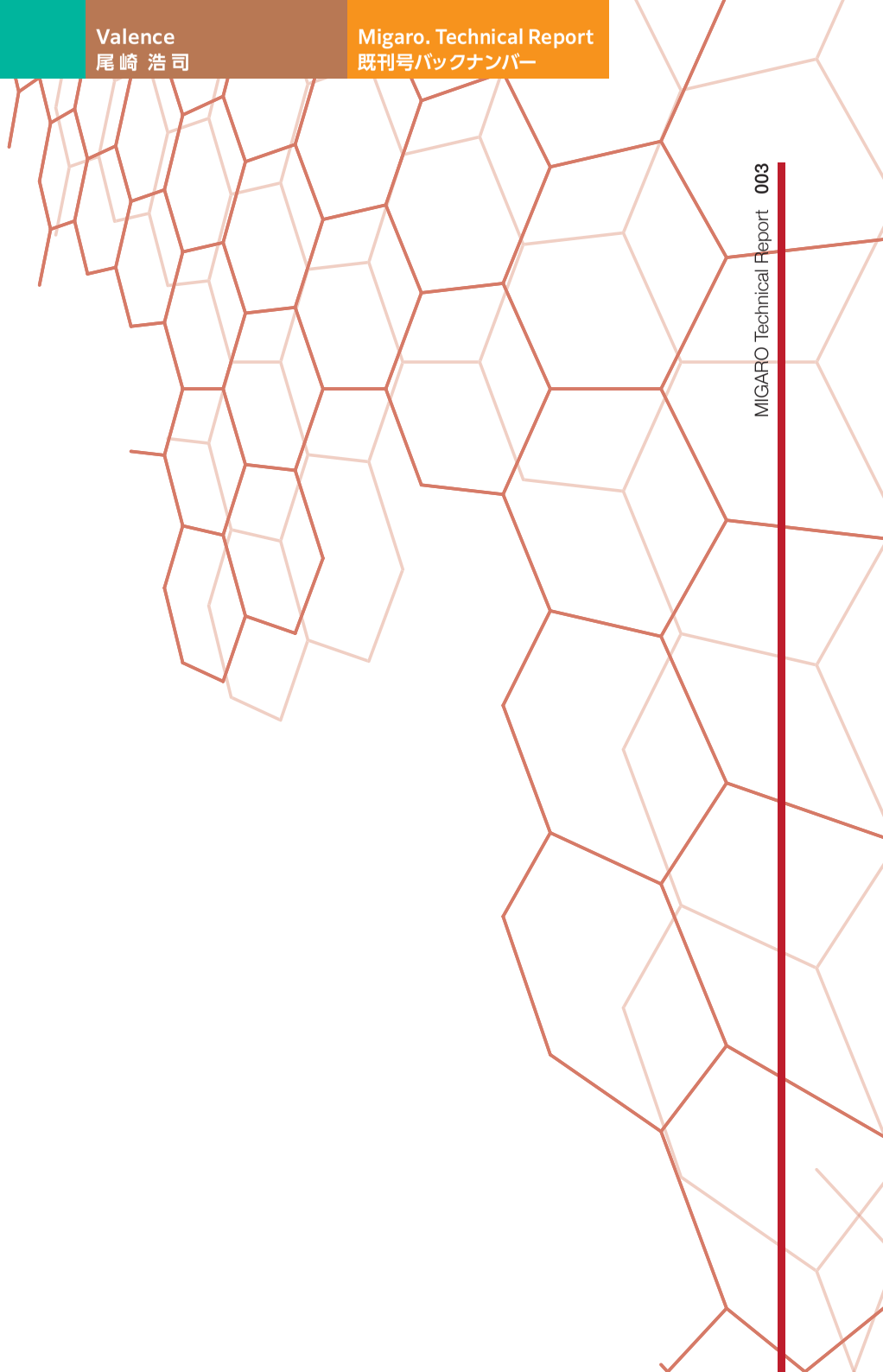
2022年12月

株式会社ミガロ.
代表取締役社長
上甲 將隆

MIGARO.
Technical Report 2022

ミガロ. テクニカルレポート 2022年

No.15



Delphi/400

Delphi/400による デジタルサイネージアプリの開発

株式会社ミガロ.
システム事業部 1課
宮坂　優大

略　歴

生年月日:1982年11月19日
最終学歴:2006年　近畿大学 理工学部卒業
入社年月:2006年4月　株式会社ミガロ. 入社
社内経歴:2006年4月　システム事業部配属

現在の仕事内容:

主にDelphi/400を利用したシステムの受託開発をメインに担当。Delphi/400のスペシャリストを目指して精進する日々である。

株式会社ミガロ.
システム事業部 1課
石山　智也

略　歴

生年月日:1988年4月5日
最終学歴:2012年　近畿大学 経済学部卒業
入社年月:2019年6月　株式会社ミガロ.入社
社内経歴:2019年6月　システム事業部配属

現在の仕事内容:

主にDelphi/400を利用したシステムの受託開発をメインに担当。開発スキルの向上を目指し、日々精進している。

1.はじめに

デジタルサイネージとは、屋外・店頭・公共空間・交通機関など、あらゆる場所で、ディスプレイなどの電子的な表示機器を使って情報を発信するメディアの事を指す。
私たちの身の回りには、すでに多くの場所で様々な情報がデジタルサイネージにより提供されている。**【図1】**

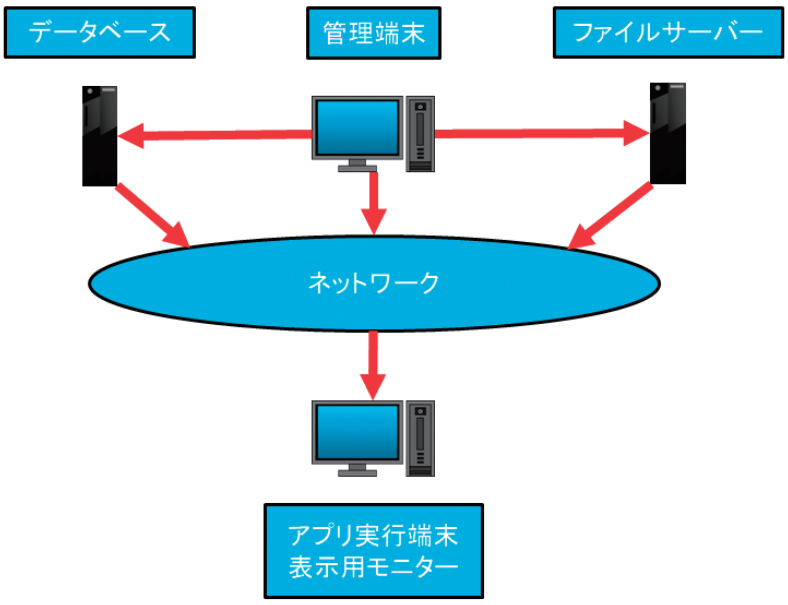
図 1　街中で見られるデジタルサイネージ

街頭の大型ビジョンや駅や空港、ショッピングモールはもちろん、小型店舗や学校、ホテル、病院、工場などにもデジタルサイネージは急速に広まっている。
様々な場所に設置できるため、特定の目的を持った人に合わせて効果的な情報を見せたり、時間帯によって表示内容を変えたりすることができるのは大きなメリットである。
また、デジタルデータのため、掲示物の張り替えなどの作業が不要で簡単に内容を入れ替えることができるのもデジタルサイネージのメリットのひとつである。
デジタルサイネージと聞くと広告メディアと思われがちだが、広告メディアに留まらず様々なシーンで明確な目的と効果を伴って情報を送り続ける手段として注目されている。
例えば、製造工場等では人材の確保が難しくなっていると言われているが、円滑に情報を共有し、生産性を向上させる手段として、デジタルサイネージを採用する企業が増えている。
IBMiに保存されている情報を取得し、リアルタイムな情報を表示することで、管理者・作業者がPCを操作することなく簡単に最新の情報を得る事ができる。
本稿ではこのようなDelphi/400を利用したデジタルサイネージアプリの開発方法を紹介する。

2. ハード・インフラ構成などの全体構成

まず、ハード・インフラ構成などの全体構成について説明を行う。

図 2　ハード・インフラ構成などの全体構成



一般的には【図2】のような構成で構築されていることが多い。この場合の各ハード役割は以下の通りである。

●データベース
→デジタルサイネージアプリで表示する基幹データと表示内容や表示スケジュールの設定を保持

●管理端末
→デジタルサイネージアプリの設定をデータベースに登録する端末

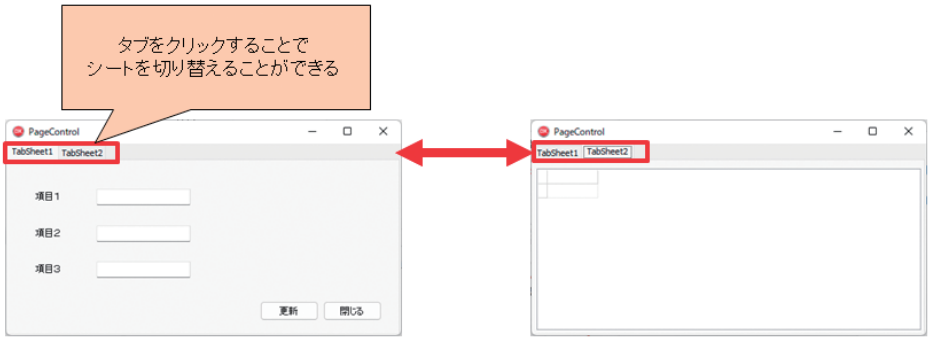
●ファイルサーバー
→デジタルサイネージアプリで表示する画像・動画ファイルを保存しておくサーバー

●アプリ実行端末及び表示用モニター
→デジタルサイネージを表示するモニターとアプリ実行端末

3.Delphi/400アプリにおけるデジタルサイネージの仕組み

次にモニターに表示させる仕組みの説明を行う。
表示させるコンテンツは一定間隔で表示を切り替えながらモニターに表示する。
この「一定間隔で表示を切り替える」動作を実現するために、TTimerとTTabSheetを使用する。
TTimerコンポーネントは一定間隔毎にイベントを実行することができるコンポーネントである。
イベントはOnTimerに記述し、イベントを発生させる頻度をIntervalプロパティで設定する。
TTabSheetはTPageControlより追加できるオブジェクトであり、1つの画面（フォーム）内でExcelのシートのようにタブを複数切り替えることができる。**【図3】**

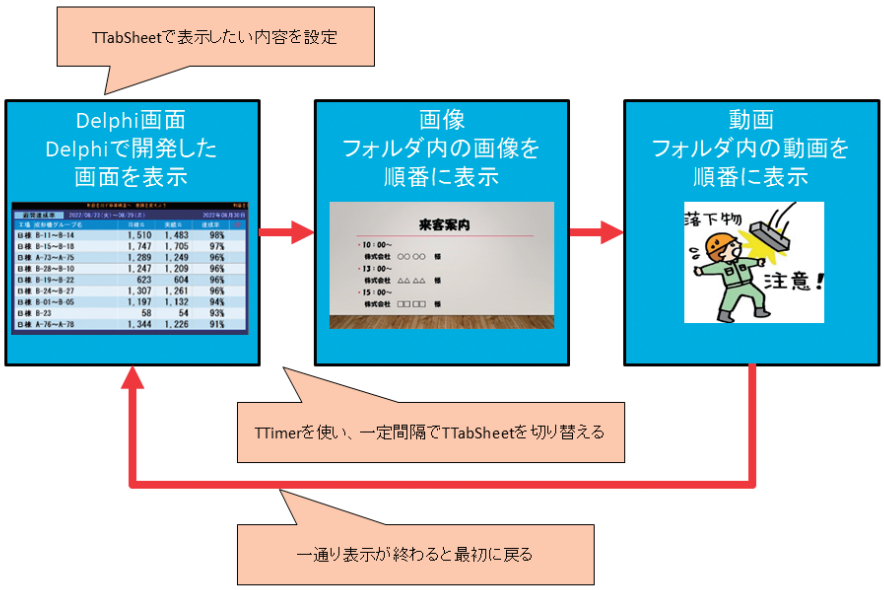
図 3　TpageControlとタブイメージ



本稿ではこのTTimerコンポーネントとTTabSheetコンポーネントの切り替え機能を使用し、アプリケーションを作成する。
それぞれのコンポーネントの具体的な設定方法は5-2.コンポーネントの配置にて説明する。

今回作成するアプリは、Delphiで開発した画面の他に、特定のフォルダに存在する画像や動画ファイルを表示するタブを用意しており、様々な情報を表示することができる。
Delphi画面→画像→動画と遷移するように設定したアプリを実行したときのイメージは**【図4】**の通りである。

図 4　アプリの実行イメージ



4.開発準備

この章では、開発に必要な設定ファイルの解説とコンポーネントのインストール方法について説明する。

4-1. 設定ファイル

表示する内容を自由に切り替える仕組みとして、IBMiにファイルを用意し使用する。
ループさせるコンテンツの順番、表示時間、表示させる画像や動画を配置している保存場所といった設定をIBMiのファイルに登録し、アプリでファイルの設定値を読み込んでコンテンツの表示を行う。
登録するファイルのレイアウトを**【図5】**に示す。

図 5　設定ファイル

パターン登録ファイル(PATRNF)

項目名	フィールド名	キー	属性	桁数
パターン	PAPTRN	1	S	2 0
表示内容	PADSPI	2	A	20
表示順	PAORDR		S	3 0
終了時間	PAENDT		A	5

表示内容設定ファイル(DSCTSF)

項目名	フィールド名	キー	属性	桁数
表示内容	DSDSPI	1	A	20
表示時間	DSDTIM		S	8 0
表示区分	DSDKBN		A	1
保存場所	DSPASS		O	255

ファイルの設定内容については以下の通りである。

【パターン登録ファイル(PATRNF)】

●パターン
→ループさせる表示内容ごとに同じ値を設定する(同値内でループが行われる)
●表示内容
→表示させる内容
表示内容設定ファイルの同項目と紐づく
●表示順
→パターンごとに表示させる順番
●終了時間
→表示を終了させる時間、同じパターンは同じ値を設定する

【表示内容設定ファイル(DSCTSF)】

●表示内容
→パターン登録ファイルの同項目と紐づく
●表示時間
→表示時間を秒で登録(動画は動画ファイルの再生時間に依存するため、0で設定する)
●表示区分
→どの画面を表示するかを設定
("1"がDelphi画面、"2"が画像、"3"が動画を表す)
●保存場所
→画像、動画を保管しているフォルダのパスを設定(ファイル名は指定しない)

登録ファイル例については**【図6】【図7】**のように設定した。
パターン登録ファイル(PATRNF)にはどのようなパターンで画面・画像・動画を表示するか、またその表示順、表示終了時間を登録する。
表示内容設定ファイル(DSCTSF)の項目「表示内容」で設定された表示時間、画像と動画の場合は「保存場所」に設定されたフォルダにある画像、動画ファイルを表示できるように設定している。

図 6　パターン登録ファイル登録例

パターン登録ファイル(PATRNF)

パターン	表示内容	表示順	終了時間
1	DISP01	1	12:00
1	IMAGE01	2	12:00
1	VIDEO01	3	12:00
1	VIDEO04	4	12:00
2	IMAGE02	1	13:00
2	VIDEO02	2	13:00
2	VIDEO03	3	13:00
3	DISP01	1	18:00
3	DISP02	2	18:00
3	IMAGE01	3	18:00
3	VIDEO01	4	18:00

パターン1
パターン2
パターン3

図 7　表示内容設定ファイル登録例

表示内容設定ファイル(DSCTSF)

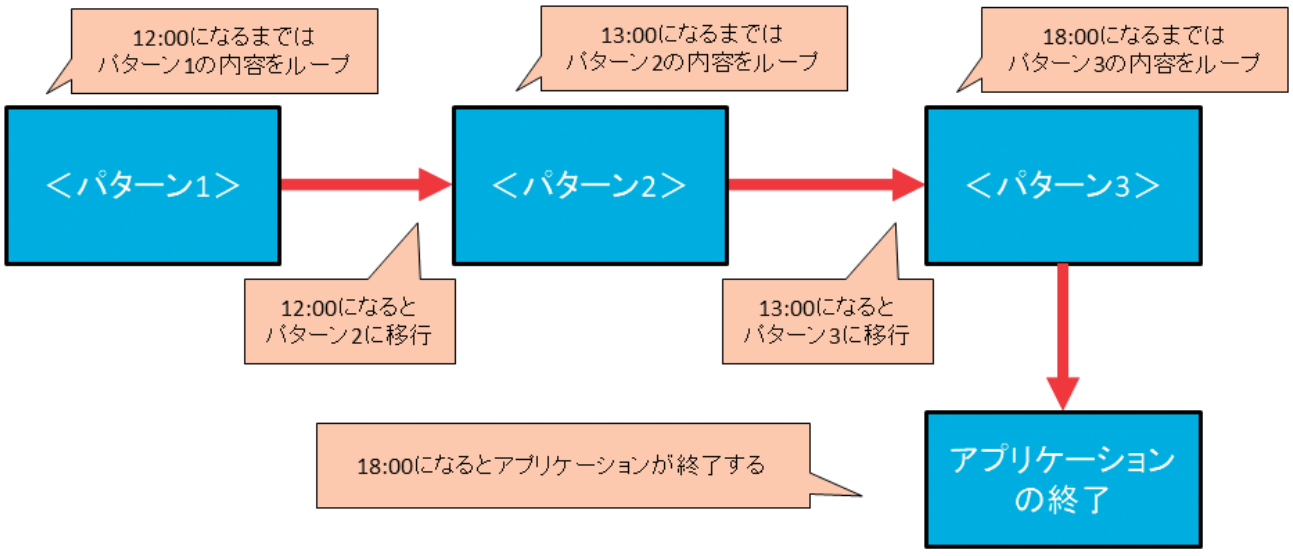
表示内容	表示時間	表示区分	保存場所
DISP01	30	1	ブランク
DISP02	30	1	ブランク
IMAGE01	20	2	C:¥.....
IMAGE02	20	2	C:¥.....
VIDEO01	0	3	C:¥.....
VIDEO02	0	3	C:¥.....
VIDEO03	0	3	C:¥.....
VIDEO04	0	3	C:¥.....

Delphi画面
画像
動画

表示内容設定ファイル(DSCTSF)の表示内容について、少し補足する。
Delphi画面である「DISP01」「DISP02」については、表示区分にDelphi画面を表す”1”を設定し、表示時間に画面を何秒間表示するかを設定する。(**【図7】**では30秒)
画像を表す「IMAGE01」「IMAGE02」については、表示区分に画像を表す”2”を設定し、表示時間はDelphi画面同様何秒間表示するかを設定する。(**【図7】**では20秒)
保存場所には画像の保存先を指定する。
動画を表す「VIDEO01」～「VIDEO04」については、表示区分に動画を表す”3”を設定し、表示時間には0を設定する。
動画はファイルに表示時間を設定するのではなく、再生時間で表示するため、表示時間の指定が不要となる。
保存場所については、動画の保存先を指定する。

パターン登録ファイル(PATRNF)を**【図6】**のように設定したときの動作イメージは**【図8】**の通りである。

図 8　動作イメージ



4-2. コンポーネントのインストール

Delphiアプリで動画を表示させるためにTWindows MediaPlayerコンポーネントのインストールを行う。TWindowsMediaPlayerコンポーネントのインストールは「コンポーネント」→「コンポーネントのインポート」→「ＡｃｔｉｖｅＸコントロールの取り込み」を選択→「WindowsMediaPlayer」を選択→「検索パス」を入力→「新規パッケージへインストール」を選択→「パッケージ名」を入力する。**【図9】～【図10】**

図 9　TWindowsMediaPlayerのインストール1

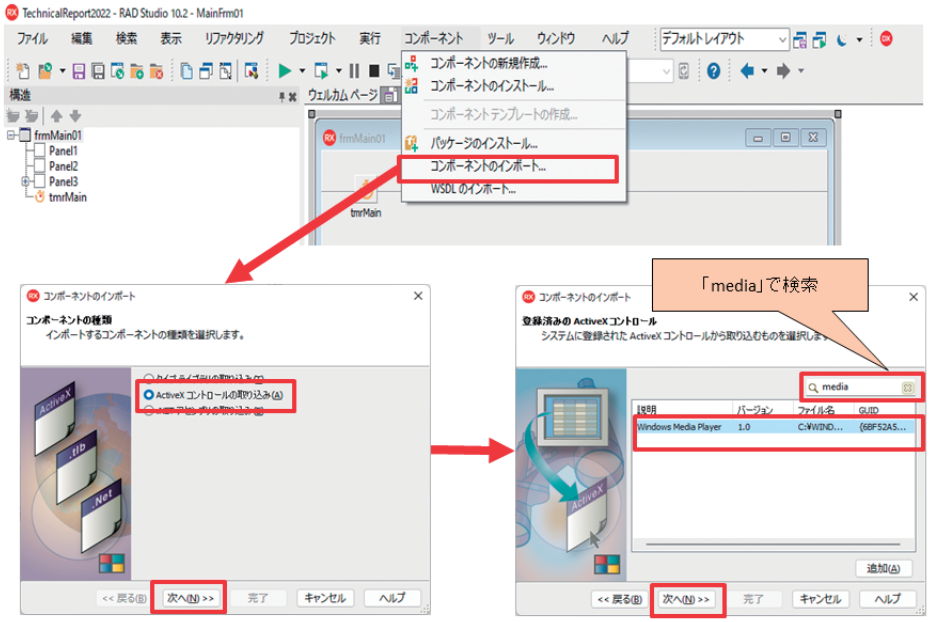
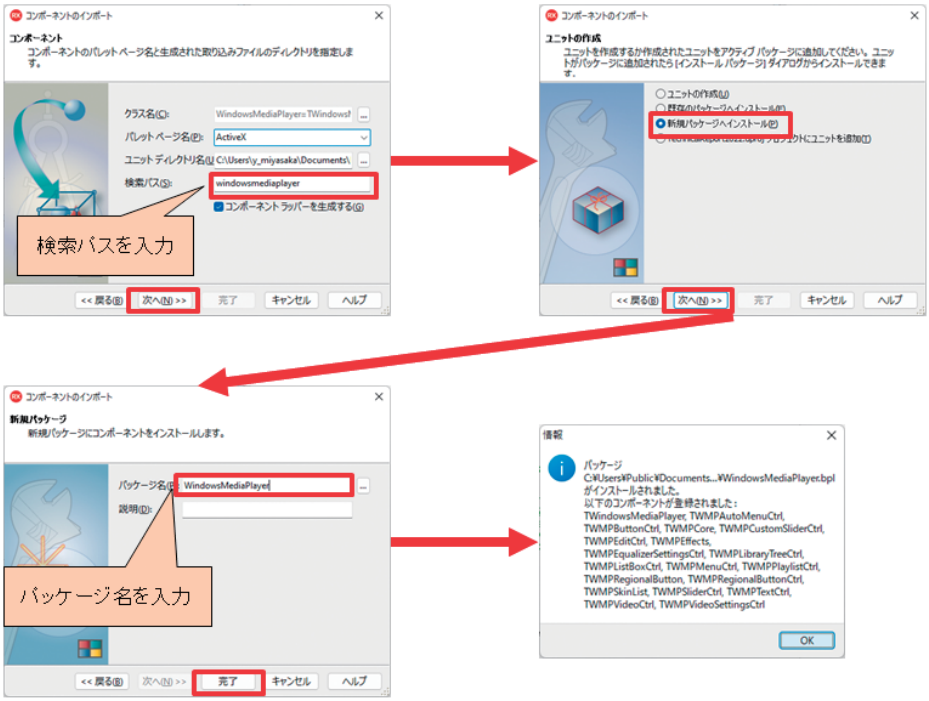


図 10　TWindowsMediaPlayerのインストール2



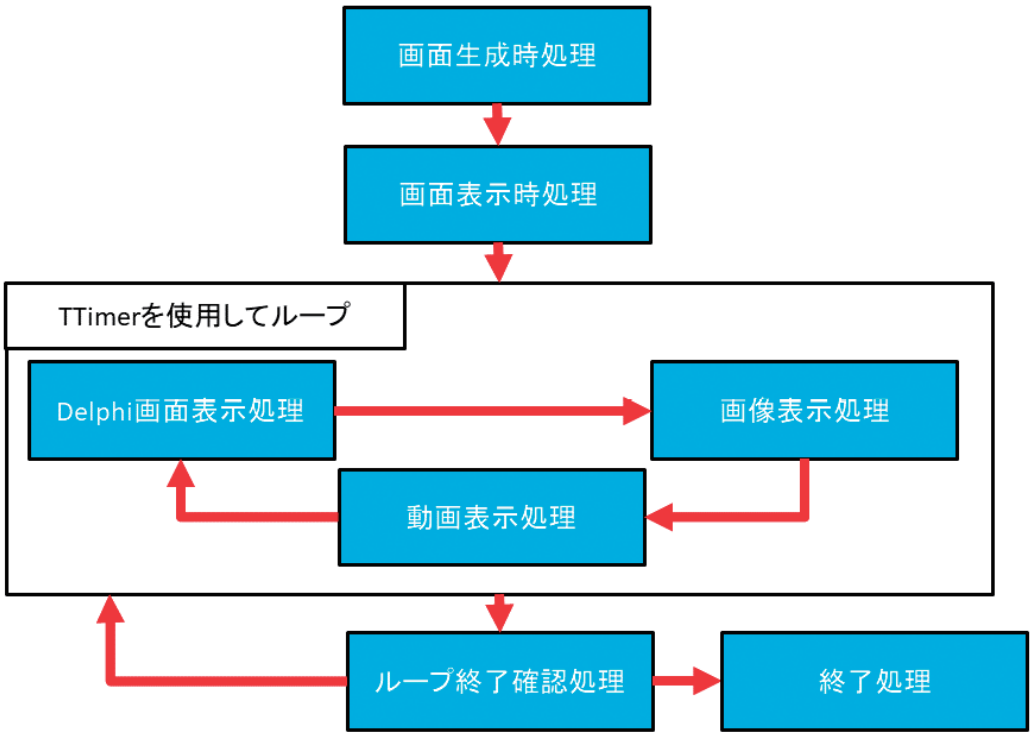
5.開発方法

この章ではデジタルサイネージの仕組みを実現するための処理の流れと、開発画面で配置するコンポーネントの設定について説明する。

5-1. 処理の流れについて

まず初めに処理の流れについて説明を行う。
処理の流れは**【図11】**のようになっており、画面生成時処理でIBMiとの接続を行い、画面表示時処理で「4-1.」で説明した設定ファイル(「パターン登録ファイル(PATRNF)」、「表示内容設定ファイル」(DSCTSF))よりデータを取得して内部保持を行う。
内部保持した設定ファイルの内容に応じてTTimerを使用しTTabSheetを切り替えることで画面表示を切り替えている。
【図11】のループ部分は「パターン登録ファイル(PATRNF)」の項目「パターン」が同一レコードの表示内容を表示順でループさせ、ループ終了確認処理で「パターン登録ファイル(PATRNF)」の項目「終了時間」が過ぎていれば次のパターンのループへ、次のパターンが無ければ処理を終了させるといった順番で処理を行っている。

図 11　処理の流れについて

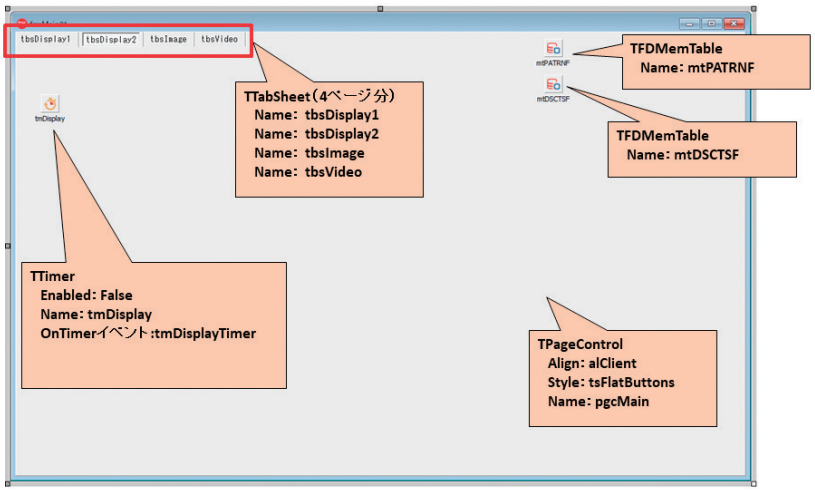


5-2. コンポーネントの配置

次に開発画面で配置するコンポーネントについて説明を行う。
デジタルサイネージアプリとして表示するフォームにコンポーネントを配置し各プロパティとイベントを設定する。
【図12】

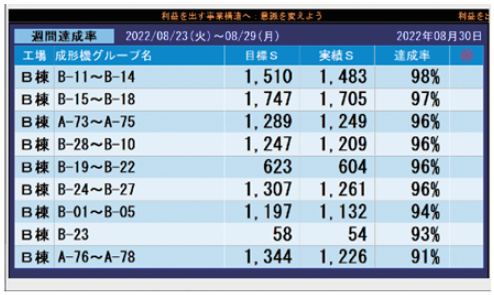
イベント記述内容については次章にて説明を行う。
TTabSheetの配置はTPageControl上で「右クリック」→「ページの新規作成」を選択するとタブを追加することができる。
【図12】で追加した各TTabSheetの役割は以下の通りである。

図 12　コンポーネントの配置（フォーム）



● 「tbsDisplay1」「tbsDisplay2」
→Delphi画面を表示するタブ
通常のDelphi開発と同じように画面を作成する。
Delphi画面の詳細な開発手順については割愛させて頂くが、実行イメージを分かりやすくするため実際に使用されている画面をDISP01、DISP02として使用している。
【図13】

図 13　Delphi画面例



DISP01画面
（tbsDisplay1シートに表示するDelphi画面）

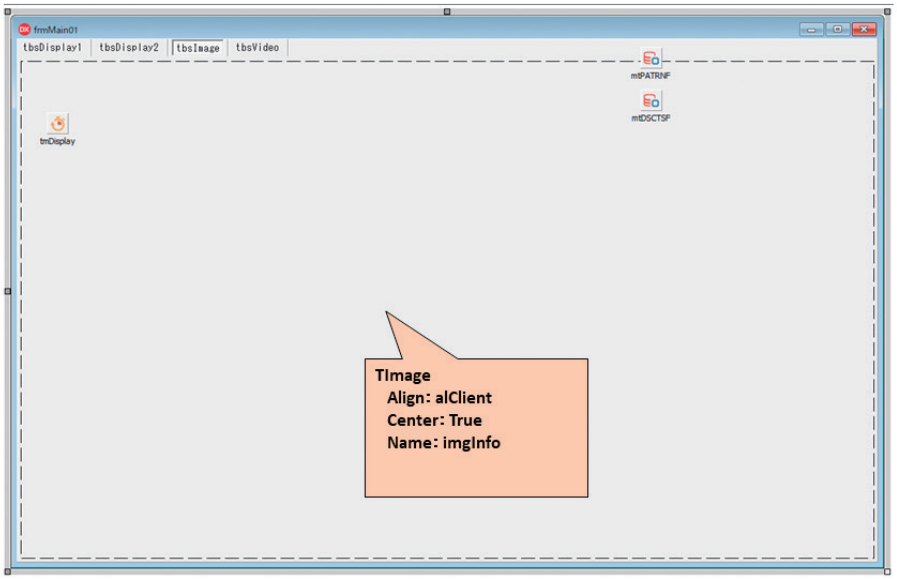
DISP02画面
（tbsDisplay2シートに表示するDelphi画面）

●「tbsImage」

→画像を表示するタブ

【図14】に従ってコンポーネントを配置し各プロパティを設定する

図 14　コンポーネントの配置（画像用タブ）

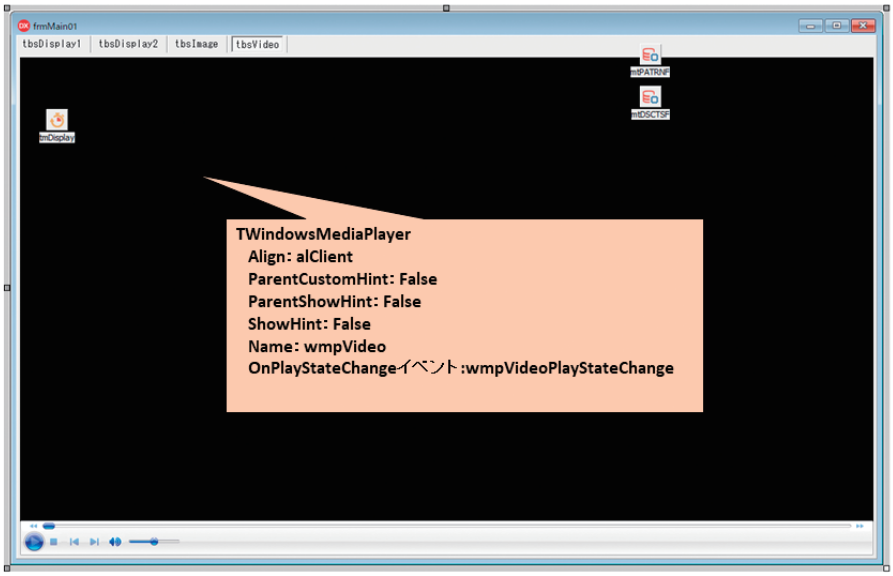


●「tbsVideo」

→動画を表示するタブ

【図15】に従ってコンポーネントを配置し各プロパティを設定する

図 15　コンポーネントの配置（動画用タブ）



6. 各処理内容について

この章では前章にて説明を行った各処理について、流れに沿って説明を行う。

6-1. 画面生成時の処理

画面生成時の処理は【ソース1】のように記述する。
IBMiとの接続はデータモジュールで行っているが本稿ではIBMiとの接続方法については割愛させていただく。
画面生成時に画面の切り替えに使用するTStringListを生成し(1-①)、TPageControlに設定したタブを全て非表示に設定する。(1-②)
タブを非表示に設定する理由は、ディスプレイに表示したときの見栄えを考慮して設定している。

ソース１

FormCreateイベント(画面生成時処理)

```
procedure TfrmMain01.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  // ※DB接続はデータモジュールのCreateで接続済

  // リスト用TStringListの生成
  slLoopList    := TStringList.Create;
  slBoardList   := TStringList.Create;
  slFileList    := TStringList.Create;
  slVideoList   := TStringList.Create;

  // タブを非表示
  for i := 0 to pgcMain.PageCount - 1 do
  begin
    pgcMain.Pages[i].TabVisible := False;
  end;

  // タイマー初期は停止状態
  tmDisplay.Enabled      := False;
end;
```

1-①

1-②

6-2. 画面表示時の処理

画面表示時の処理は**【ソース2】**のように記述する。
画面表示時では後続処理で使用する変数（FLoopNo：パターン登録ファイルの項目「パターン」の保持に使用、FDispCnt：Delphi画面タブの表示回数保持に使用）の初期化（2-①）→DBより「パターン登録ファイル（PATRNF）」、「表示内容設定ファイル（DSCTSF）」をSQLで取得し、それぞれのデータをTFDMemTable（mtPATRNF、mtDSCTSF）に保持（2-②、2-③）→ループ処理（2-④）へという流れで処理を行う。

TFDMemTableとはインメモリにデータを保管する非ビジュアルコンポーネントである。
FireDAC用のTClientDataSetとイメージして頂けると分かりやすい。
起動時に設定値をIBMiより取得するが、設定を毎回取得してはレスポンスに影響がでる可能性があるため、初回に取得した情報を内部で保持させている。

ソース 2

FormShowイベント（画面表示時処理）

```
procedure TfrmMain01.FormShow(Sender: TObject);
var
  sBFPTN: String;
begin
  // 初期化処理                                          ┐
  FLoopNo     := 0;                                      ├ 2-①
  FDispCnt    := 0;                                      ┘

  // パターン登録ファイルを参照し、パターンを設定する      ┐
  with dmMain.qryPATRNF do                               │
  begin                                                  │
    Close;                                               │
    SQL.Clear;                                           │
    SQL.Text := 'SELECT * FROM PATRNF ' +                │
                ' ORDER BY PAPTRN, PAORDR ';             │
    Open;                                                │
                                                         │
    // 取得データをデータセットに渡す                    │
    mtPATRNF.Close;                                      │
    mtPATRNF.AppendData(dmMain.qryPATRNF.Data, False);   │
  end;                                                   │
                                                         ├ 2-②
  // パターンをLoopListに保管                           │
  mtPATRNF.First;                                        │
  while not mtPATRNF.Eof do                              │
  begin                                                  │
    if (sBFPTN <> mtPATRNF.FieldByName('PAPTRN').AsString) then
    begin                                                │
      slLoopList.Add(mtPATRNF.FieldByName('PAPTRN').AsString);
    end;                                                 │
                                                         │
    // 前回値の保存                                      │
    sBFPTN := mtPATRNF.FieldByName('PAPTRN').AsString;   │
    mtPATRNF.Next;                                       │
  end;                                                   │
  FLoopCnt := slLoopList.Count;       // ループリストの件数┘

  // 表示内容設定ファイルを参照し、パターン毎の設定を保存する ┐
  with dmMain.qryDSCTSF do                                  │
  begin                                                     │
    Close;                                                  │
    SQL.Clear;                                              │
    SQL.Text := 'SELECT * FROM DSCTSF ';                    ├ 2-③
    Open;                                                   │
                                                            │
    // 取得データをデータセットに渡す                       │
    mtDSCTSF.Close;                                         │
    mtDSCTSF.AppendData(dmMain.qryDSCTSF.Data, False);      │
  end;                                                      ┘

  // ループリストの処理                                 ┐
  Proc_LoopList;                                        ├ 2-④
                                                        ┘
  // ループ終了時間チェック
  // （最初のループ終了時間以降に起動した場合を考慮）
  Check_LoopEndTime;

  // ボードリストの処理
  Proc_BoardList;
end;
```

6-3. ループ処理

次に【図11】にあるTTimerを使用したループ処理について説明を行う。

まず、画面表示時にTFDMemTable(mtPATRNF)へ保持した「パターン登録ファイル(PATRNF)」の設定情報を変数へセットする。具体的には【ソース3】のように記述する。

変数へセットする内容は以下の通りである。

●ループさせるパターンの終了時間(FLoopEnd)

→mtPATRNFをループさせるパターンでLocateし、終了時間をTTime型の変数へセットする。(3-①)

●表示内容(slBoardList、FBoardNo)

→mtPATRNFをループさせるパターンでFilterし、表示内容をTStringList型の変数へセット、この時mtPATRNFの内容はパターン、表示順の順番で並んでいる(SQLのORDER　BY句で指定した順番)

StringListより表示内容のStringを取得するためにInteger型の変数へ0をセットする。(3-②)

●表示内容の件数(FBoardCnt)

→mtPATRNFを前述の条件でFilterした件数をセット(3-②)

ソース 3

Proc_LoopList(ループリストの処理)

```
procedure TfrmMain01.Proc_LoopList;
var
  sLoop, sWork: string;
begin
  // ループリストよりパターンを取得
  sLoop := slLoopList.Strings[FLoopNo];

  // ループ終了時間をデータセットから取得
  mtPATRNF.Filtered := False;
  mtPATRNF.First;
  if (mtPATRNF.Locate('PAPTRN', VarArrayOf([sLoop]), [])) then
  begin
    sWork := mtPATRNF.FieldByName('PAENDT').AsString + ':00';
  end
  else
  begin
    sWork := '23:59:00';
  end;
  FLoopEnd := StrToTimeDef(sWork, StrToTime('00:00:00'));

  // パターン内容をボードリストに保持
  mtPATRNF.First;
  mtPATRNF.Filter := 'PAPTRN = ' + sLoop;
  mtPATRNF.Filtered := True;

  slBoardList.Clear;
  while not mtPATRNF.Eof do
  begin
    slBoardList.Add(mtPATRNF.FieldByName('PADSPI').AsString);
    mtPATRNF.Next;
  end;
  FBoardCnt := slBoardList.Count;   // ボードリストの件数
  FBoardNo  := 0;

  Check_LoopEndTime;
end;
```

3-①

3-②

次に、ボードリストの処理について説明する。
ボードリストとは、画面表示時にTFDMemTable（mtDSCTSF）へ保持した「表示内容設定ファイル（DSCTSF）」を元に表示内容に応じて画面の表示時間設定、表示させるタブの処理、TTimerで表示する画面のループまたは次のタブへ切り替える処理を行っている。

ボードリストの処理【ソース4】では、【ソース3】の処理でセットした表示内容の変数（slBoardList、FBoardNo）を使ってTFDMemTable（mtDSCTSF）へ保持した「表示内容設定ファイル（DSCTSF）」より表示する画面内容、表示時間、画像と動画の保存場所を取得してセットする（4-①）→取得した画面内容に対応したタブの処理を行う（4-②）→表示時間の設定をTTimerのIntervalプロパティにセットする（4-③）といった流れで処理を行う。

ソース 4

Proc_BoardList（ボードリストの処理）

```
procedure TfrmMain01.Proc_BoardList;
var
  sBoard: string;
  iDisTime: Integer;
begin
  // ボードリストよりボード名を取得
  sBoard := slBoardList.Strings[FBoardNo];

  //<ボード名>
  FBoardName := sBoard;

  if (mtDSCTSF.Locate('DSDSPI', VarArrayOf([sBoard]), [])) then
  begin
    //<表示時間>
    iDisTime := mtDSCTSF.FieldByName('DSDTIM').AsInteger;

    //<フォルダパス>
    FFolderPath := mtDSCTSF.FieldByName('DSPASS').AsString;
  end
  else
  begin
    // 存在しなかった場合、仮に5秒を設定
    iDisTime := 5;

    FFolderPath := '';
  end;

  // タブ切替
  pgcMain.Visible := False;
  try
    // Delphi画面１
    if (FBoardName = 'DISP01') then
    begin
      pgcMain.ActivePage := tbsDisplay1;
      Proc_DISP;
    end
    // Delphi画面２
    else if (FBoardName = 'DISP02') then
    begin
      pgcMain.ActivePage := tbsDisplay2;
      Proc_DISP;
    end
    // 画像パターン
    else if (COPY(FBoardName, 1, 5) = 'IMAGE') then
    begin
      pgcMain.ActivePage := tbsImage;
      Proc_IMAGE;
    end
    // 動画パターン
    else if (COPY(FBoardName, 1, 5) = 'VIDEO') then
    begin
      pgcMain.ActivePage := tbsVideo;
      Proc_VIDEO;
    end;

  finally
    pgcMain.Visible := True;
  end;

  // タイマーに時間セット／開始
  // 表示タイマー
  tmDisplay.Interval  := iDisTime * 1000;  // Intervalはミリ秒で指定（ファイルは秒で登録）
  if (iDisTime > 0) then
  begin
    tmDisplay.Enabled := True;
    Inc(FDispCnt);
  end;
end;
```

4-①
4-②
4-③

前述のタブの処理【ソース4】（4-②）について詳しく解説を行う。今回作成するデジタルサイネージアプリではDelphi画面、画像、動画の3種類を扱っているが、表示する画面によって切り替えの条件が異なる。
Delphi画面は一度表示したら次の表示内容へ、画像と動画はフォルダ内のファイルをすべて表示したら次の表示内容へ、さらに画像については1枚の画像を「表示内容設定ファイル（DSCTSF）」の表示時間に設定された時間分表示、動画の場合は1つの動画の再生が終了したら次の動画へ、といったように処理を分岐させる工夫が必要となる。

Delphi画面の処理(【ソース5】)

Delphi画面の処理は**【ソース5】**のように記述する。Delphi画面については1度表示すれば次の表示内容へ切り替えればよいので、表示回数保持変数(FDispCnt)で制御している。表示回数保持変数(FDispCnt)は**【ソース4】**(4-③)でカウントアップ、後述するTTimerのtmDisplayTimerイベント(表示内容の切り替え処理)で初期化する。

ソース 5

Proc_DISP(Delphi画面の処理)

```
procedure TfrmMain01.Proc_DISP;
begin
  if (FDispCnt > 0) then
  begin
    // 1度でも表示したら次のボードへ
    tmDisplayTimer(tmDisplay);
  end;
end;
```

画像の処理(【ソース6】～【ソース7】)

画像の処理は**【ソース6】**～**【ソース7】**のように記述する。まず**【ソース4】**(4-①)で取得した「表示内容設定ファイル(DSCTSF)」の画像の保存場所を参照し、TDirectory.GetFilesメソッドを使って画像ファイル名(今回は拡張子tifを指定)の一覧を取得、TStringList型変数へ保持する。(6-①)
画像ファイルが存在する場合(6-②)は**【ソース7】**で画像タブに配置したTImageコンポーネントのLoadFromFileメソッドで画像を表示している。(7-①)
画像については前述の通り、保存場所の画像全てを設定された表示時間分表示し、次の画像へ遷移させるようにしている。画像を表示するたびに画像リストカウント用変数(FImageCnt)をカウントアップさせて取得した画像表示回数を制御し、後述するTTimerのtmDisplayTimerイベント(表示内容の切り替え処理)では再度**【ソース7】**のメソッドを呼び出して次の画像を表示するまたは次の表示内容を表示するかの制御を行っている。

ソース6

Proc_IMAGE(画像の処理)

```
procedure TfrmMain01.Proc_IMAGE;
var
  sPtrn, sFileName: String;
  soOption: TSearchOption;
  sdaFiles: TStringDynArray;
begin
  // 画像をクリア
  imgInfo.Picture := nil;

  // tiffファイルを対象に検索
  sPtrn := '*.tif' ;

  // 指定フォルダのファイル一覧を取得
  FFolderPath := StringReplace(FFolderPath, '$', '¥', [rfReplaceAll]);
  sdaFiles := TDirectory.GetFiles(FFolderPath, sPtrn, soOption);
  slFileList.Clear;
  for sFileName in sdaFiles do
  begin
    slFileList.Add(sFileName);
  end;

  // 設定パスにファイルが存在する場合
  if (slFileList.Count >= 1) then
  begin
    // 画像の表示
    DspPic_IMAGE;
  end
  else
  begin
    // ファイルが存在しない場合は、次のボード処理に移る
    tmDisplayTimer(tmDisplay);
    Abort;
    Exit;
  end;
end;
```

6-①
6-②

ソース7

DspPic_IMAGE（画像表示の処理）

```
procedure TfrmMain01.DspPic_IMAGE;
begin
  // 画像をクリア
  imgInfo.Picture := nil;

  // 画像リストカウンター
  Inc(FImageCnt);

  // 最後のファイルを表示した後は、次のボード処理に移る
  if (FImageCnt > slFileList.Count) then
  begin
    FBoardName := '';
    FImageCnt  := 0;
    tmDisplayTimer(tmDisplay);
    Abort;
    Exit;
  end
  else
  begin
    imgInfo.Picture.LoadFromFile(slFileList.Strings[FImageCnt-1]);
  end;
end;
```

7-①

動画の処理（【ソース8】～【ソース10】）

動画の処理は**【ソース8】～【ソース10】**のように記述する。基本的な流れは画像の処理と同様に保存場所の動画ファイルの一覧を取得する。(8-①)
動画ファイルがあれば（8-②）動画リストカウント用変数（FVideoCnt）で動画表示回数をカウント（9-①）し、保存場所全ての動画を順番に再生する、といった流れとなっている。

動画については、画像と違い再生時間が存在するので、表示時間で動画ファイルを切り替えるのではなく、単純に動画を最後まで表示させるだけでよい。
保存場所全ての動画の再生が終わると後述するTTimerのtmDisplayTimerイベント（表示内容の切り替え処理）を呼び出すことで次の表示内容へ切り替える。

ここでは主に、どのようにして動画の再生が終了したときに次の動画を再生するのかと、動画再生用のコンポーネントTWindowsMediaPlayerついて説明を行う。
まず、**【ソース8】**より呼び出される**【ソース9】**（9-①）の処理でTWindowsMediaPlayerコンポーネント（wmpVideo）に以下の設定を行うことで動画を再生している。

●「wmpVideo.URL」
→再生する動画の場所をフルパスで指定
●「wmpVideo.stretchToFit」
→Trueを設定することでコンポーネントの表示エリアの大きさに引き伸ばして再生
●「wmpVideo.uiMode」
→noneを設定することでシークバー部分を非表示にする
●「wmpVideo.controls.play」
→動画を再生する命令

動画の再生が始まると動画再生状態が変更され**【ソース10】**のイベントが処理される。
【ソース10】のイベントは動画再生状態が変更されるたびに処理されるイベントで、動画がどういった状態なのかを判定することができる。
動画の再生準備が完了したときはwmppsReadyが、動画の再生が終了したときにwmppsMediaEndedが返されるようになっており、この値を使用して動画が終了した場合は**【ソース9】**のメソッドを呼び出して次の動画を再生するか次の表示内容を表示するか判断を行っている。

ソース 8

Proc_VIDEO（動画の処理）

```
procedure TfrmMain01.Proc_VIDEO;
var
  sPtrn, sFileName: string;
  soOption: TSearchOption;
  sdaFiles: TStringDynArray;
begin
  // 初期化
  slVideoList.Clear;
  FVideoCnt := 0;

  // mp4ファイルを対象に検索
  sPtrn := '*.mp4';

  // ディレクトリの列挙モード
  soOption := TSearchOption.soTopDirectoryOnly; // トップレベル列挙モード

  // 指定のディレクトリ内のファイルのリスト
  FFolderPath := StringReplace(FFolderPath, '$', '¥', [rfReplaceAll]);
  sdaFiles := TDirectory.GetFiles(FFolderPath, sPtrn, soOption);
  for sFileName in sdaFiles do
  begin
    slVideoList.Add(sFileName);
  end;


  // ファイルが存在する場合
  if (slVideoList.Count >= 1) then
  begin
    // 動画の有無チェック／再生
    Check_Video;
  end
  else
  begin
    // ファイルが存在しない場合は、次のボード処理に移る
    tmDisplayTimer(tmDisplay);
    Abort;
    Exit;
  end;
end;
```

8-①

8-②

ソース 9

Check_Video（動画の有無チェック／再生の処理）

```
procedure TfrmMain01.Check_Video;
begin
  if (FVideoCnt < slVideoList.Count) then
  begin
    wmpVideo.URL := slVideoList.Strings[FVideoCnt];
    wmpVideo.stretchToFit := True;
    wmpVideo.uiMode := 'none';
    wmpVideo.controls.play;
    Inc(FVideoCnt);
  end
  else
  begin
    //保存場所の動画を全て再生した場合は、次のボード処理に移る
    tmDisplayTimer(tmDisplay);
    Abort;
    Exit;
  end;
end;
```

9-①

ソース10

wmpVideoPlayStateChangeイベント（動画再生状態変更時の処理）

```
procedure TfrmMain01.wmpVideoPlayStateChange(ASender: TObject;
  NewState: Integer);
begin
  case NewState of
    wmppsMediaEnded:
    begin
      Check_Video;
    end;
    wmppsReady:
    begin
      if (Assigned(wmpVideo.currentMedia)) then
      begin
        wmpVideo.controls.play;
      end;
    end;
  end;
end;
```

表示内容の切り替え処理(【ソース11】)

表示内容の切り替え処理は**【ソース11】**のように記述する。前述の通り、画像の処理から呼び出された場合は再度画像の処理へ戻り(11-①)、画像の処理以外から呼び出された場合は後述のループ終了確認処理と終了処理を呼び出してループを終了するかの確認(11-②)を行い、終了時間でない場合は**【ソース4】**の処理(11-③)を呼び出してループするようになっている。

ソース11

tmDisplayTimerイベント(表示内容の切り替え処理)

```
procedure TfrmMain01.tmDisplayTimer(Sender: TObject);
begin
  // 画像の場合、フォルダ内のファイル分処理するまで、ループ
  if (COPY(FBoardName, 1, 5) = 'IMAGE') then
  begin
    // 画像表示
    DspPic_IMAGE;
  end
  else
  begin
    // タイマーを停止する
    tmDisplay.Enabled := False;

    // 次のボードの処理へ
    FDispCnt := 0;
    Inc(FBoardNo);

    // ボードが一周したら最初に戻る
    if (FBoardCnt < (FBoardNo + 1)) then
    begin
      FBoardNo := 0;
    end;

    // ループ終了時間チェック
    Check_LoopEndTime;

    // ボードリストの処理
    Proc_BoardList;
  end;
end;
```

11-① 11-② 11-③

6-4. ループ終了確認処理と終了処理

次に**【図11】**にあるループ終了確認処理と終了処理について説明を行う。

ループ終了確認処理は**【ソース12】**のように記述する。**【ソース3】**(3-①)の処理で取得した「パターン登録ファイル(PATRNF)」の終了時間が過ぎた場合は次のパターンのループへ入り(12-②)、「パターン登録ファイル(PATRNF)」のパターンを全てループした場合はSelf.Closeで画面を閉じてアプリを終了(12-①)するという処理を行っている。

こうすることで指定時間内は画面や画像、動画を表示させ続けることができる。逆に指定時間外にアプリケーションを自動で終了させることができる。

ソース12

Check_LoopEndTime(ループ終了確認処理と終了処理)

```
procedure TfrmMain01.Check_LoopEndTime;
begin
  // ループ終了時間を過ぎた場合
  if (FLoopEnd <> StrToTime('00:00:00')) and
     (FLoopEnd < Time) then
  begin
    // 次のループの処理へ
    Inc(FLoopNo);

    // ループが一周したら終了
    if (FLoopCnt < (FLoopNo + 1)) then
    begin
      // アプリを終了する
      Self.Close;
    end
    else
    begin
      // ループリストの処理
      Proc_LoopList;
    end;
  end;
end;
```

12-① 12-②

6-5. アプリの実行

それでは開発したアプリを実行するとどのように表示されるか確認してみる。

【図6】で登録したパターン登録ファイルの「パターン3」として実行した場合、【図17】のように画面が切り替わる。

パターン3で使用する画像フォルダ（IMAGE01）、動画フォルダ（VIDEO01）は【図16】のようになっており、それぞれ2つのファイルを配置している。（画像、動画フォルダは表示内容設定ファイル【図7】の項目「保存場所」で指定したパスに配置）

この状態で実行すると【図17】のように画面が切り替わる。

図 16　画像、動画フォルダイメージ

図 17 **アプリの実行**

Delphi画面（30秒間表示したら次へ）

DISP01画面

DISP02画面

画像（20秒間表示したら次へ）

01_来客案内.tif

02_今月の目標.tif

動画（再生終了したら次へ）

01_落下物注意.mp4

02_熱中症注意.mp4

熱中症注意の動画が終了すると
DISP01画面へ戻る

7.さいごに

本稿ではDelphi/400を利用したデジタルサイネージアプリの開発方法として、Delphi画面や、指定画像を表示、動画を再生する方法を紹介した。
簡単なものであれば、開発コストをそれほど掛けることなく効果的な情報を表示するアプリを作成することができる。
デジタルサイネージの利点を活用し、円滑に情報を共有する手段として利用してもらえると幸いである。

Delphi/400

アプリケーションテザリングと モバイルカメラを利用した業務の効率化

株式会社ミガロ.
システム事業部 2課
前坂　誠二

略　歴

生年月日:1989年3月21日
最終学歴:2011年　関西大学 文学部卒業
入社年月:2011年04月　株式会社ミガロ. 入社
社内経歴:2011年04月　システム事業部配属

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を担当。
Delphi、Delphi/400の開発経験を積みながら、
日々スキルを磨いている。

1.はじめに

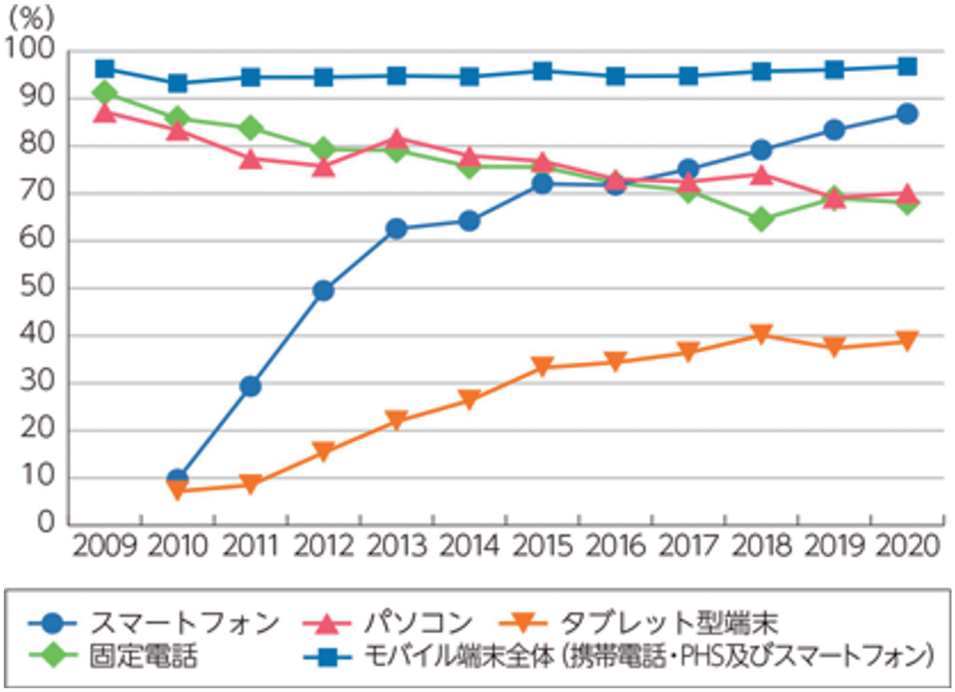
近年、モバイル端末の普及率は目まぐるしく上昇しており、スマートフォンに至っては世帯での保有率が8割を超えている【図1】。

【図1】は世帯での保有率のグラフであるが、企業においてもスマートフォン、タブレット端末の利用率は半数を超えている【図2】。また、モバイル端末の普及と併せて、各端末自身の性能やカメラ機能についても向上している。特にカメラ機能においては、その利便性からデジタルカメラやビデオカメラの代わりに使用されるユーザーも多いのではないだろうか。

Delphi/400では、過去のミガロ.テクニカルレポートで執筆されているように、簡単にカメラを利用したアプリケーションを作成することが可能である。

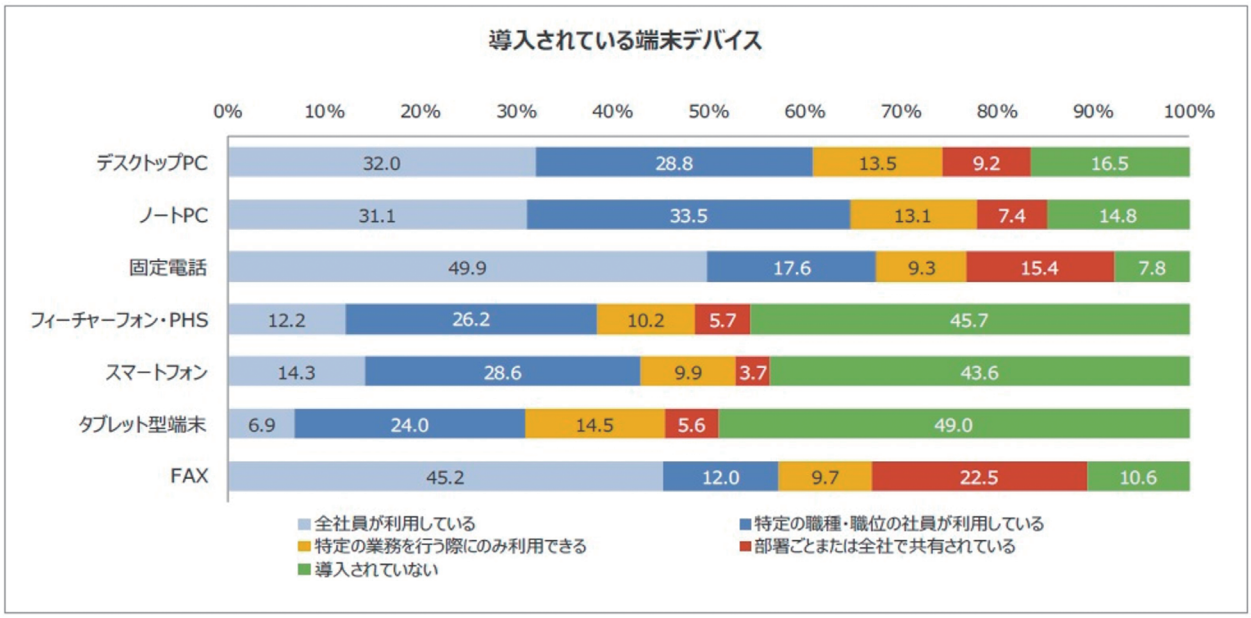
本稿では、モバイル端末のカメラ機能とDelphi/400アプリケーションの連携で、業務の効率化を図る方法を紹介する。モバイル端末を未導入の状態から導入した場合を想定して執筆しているため、これから導入を検討されている方にもぜひ一読いただければ幸いである。

図 1 情報通信機器の世帯保有率



出典：総務省ホームページ
(https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/html/nd111100.html)

図 2 導入されている端末デバイス



出典：平成30年度　デジタル化による生活・働き方への影響に関する調査研究　成果報告書
(https://www.soumu.go.jp/johotsusintokei/linkdata/r01_02_houkoku.pdf)

2.モバイル端末の導入例

本稿では、【図3】をモバイル端末導入前の例とする。【図3】では、各作業現場で物品の写真撮影と状態チェックを行い、事務所にチェック内容を伝達する。最後に事務員が伝達内容を基に画像保存とデータ入力＆更新を行うという業務の流れを想定している。

【図3】に対して、モバイル端末を導入し、本稿で紹介する技術を活用した場合、【図4】のような業務の流れに置き換わる。【図4】では、モバイル端末のカメラ映像を事務所と共有し、モバイル端末から送信される情報と映像内容を基に事務員がデータ修正＆更新するという流れを想定している。

本例におけるモバイル端末導入前後での大きな違いは、作業員の物理的な移動が不要となる点である。データ内容に不備があった場合でも事務所側で現場の確認とデータの修正が可能となる。また、モバイル端末導入後はカメラ映像が共有され、同じ映像を見ながらの会話が可能となる。そのため、作業員と事務員間でのコミュニケーションもとりやすくなる。

図 3　モバイル端末導入前フロー

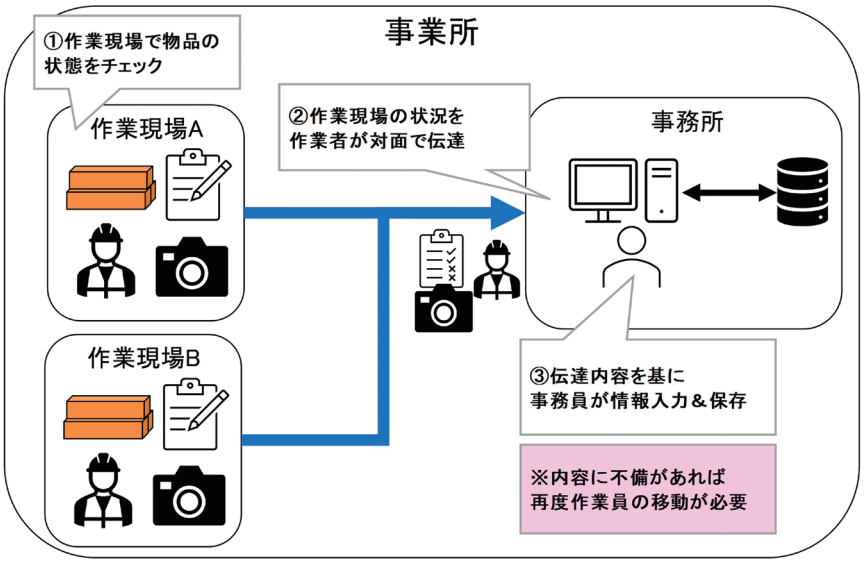
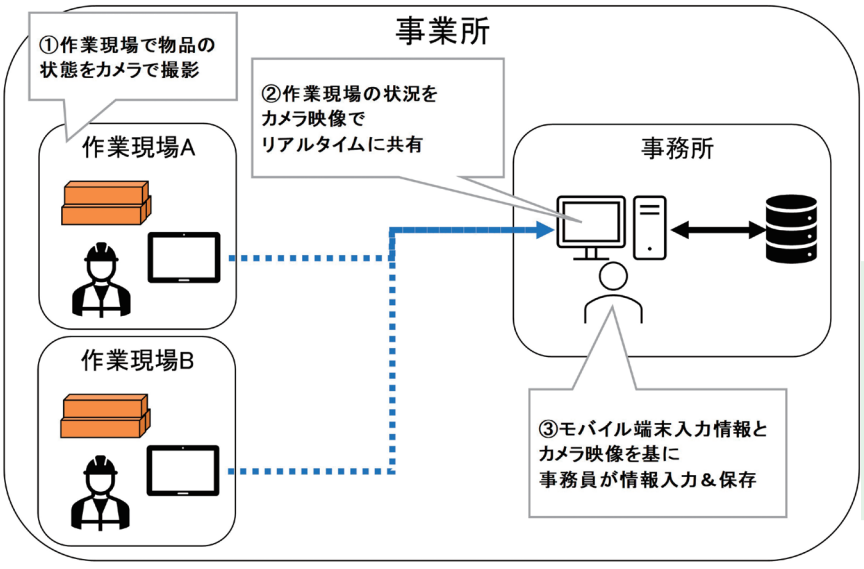


図 4　モバイル端末導入後フロー



3.アプリケーションテザリングの利用

【図4】の実現のために、アプリケーションテザリングという技術を利用する。

アプリケーションテザリングはDelphi/400XE7以降で利用可能な技術である。同一ネットワーク内やBluetoothで通信可能な環境であれば、コンポーネントの設定と簡単な処理の実装だけで端末間でのデータや処理の共有が可能となる。詳細については過去のミガロ.テクニカルレポート「アプリケーションテザリングを利用したPC&モバイルアプリケーション連携」を参考にしていただきたい。

https://www.migaro.co.jp/contents/support/technical_report_search/no10/tech/10_01_05.pdf

アプリケーションテザリングでは、端末間のやりとりに中間サーバーが不要であるため、シンプルな構造でアプリケーションの作成が可能である。また、VCL、FireMonkeyのどちらのフレームワークで作成したアプリケーションであっても、相互の連携が可能である。例えば、現在使用している業務アプリケーションがVCLフレームワークで作成されたものでも、FireMonkeyフレームワークで作り直す必要がなくそのまま活用可能である。

本稿の例は、モバイル端末導入前は事務所で【図5】のようなVCLアプリケーションを使用していると想定する。モバイル端末導入後は【図6】のようにカメラ映像を表示する画面を追加する。

また、モバイル端末側のアプリケーションには、入力項目とカメラ機能を実装する【図7】。モバイル端末から送られたカメラ映像や入力情報はPC端末側にリアルタイムに共有される【図8】。次章より具体的な実装方法について紹介する。

図5 モバイル端末導入前アプリケーション

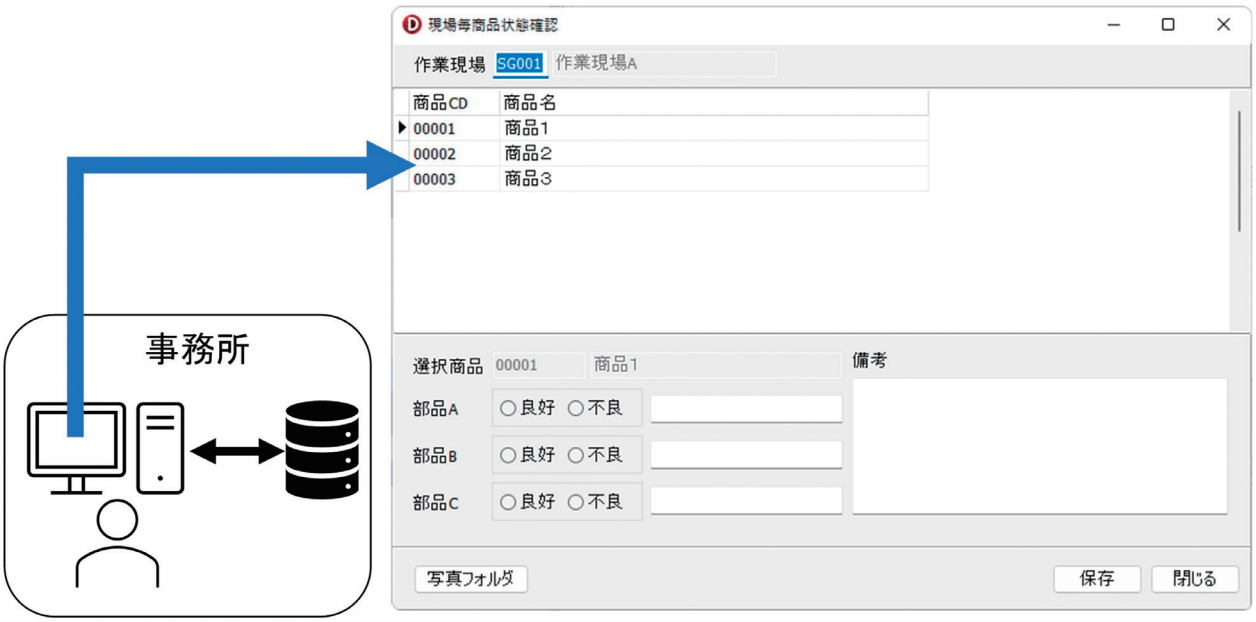


図 6　モバイル導入後アプリケーション（PC）

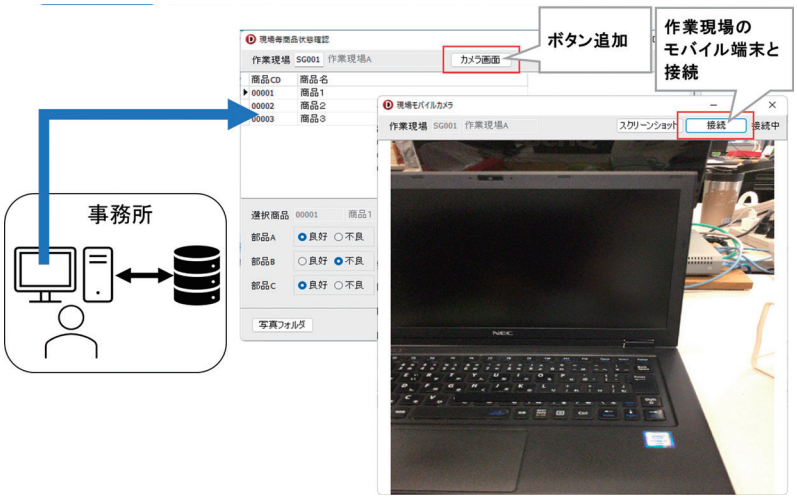


図 7　モバイル端末導入後アプリケーション（iPad）

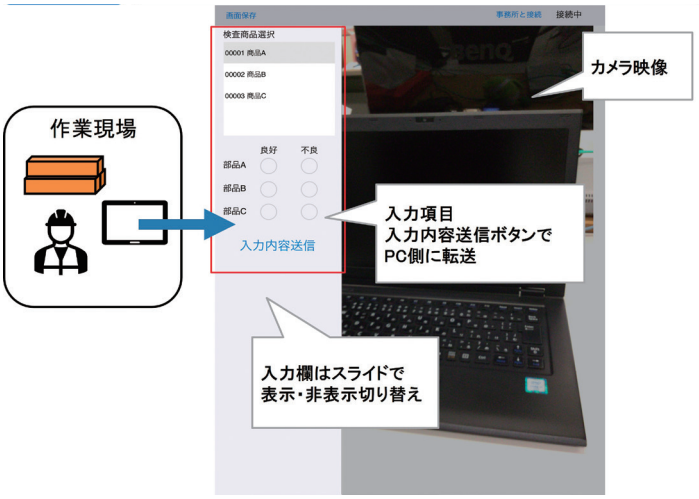


図 8　モバイル端末→PC端末へのデータ共有

4.データ共有処理の実装

利用端末は作業現場側のモバイル端末をiPadとし、事務所側をWindowsのPC端末とする。iPadはFireMonkeyフレームワークで開発を行い、PC端末はVCLフレームワークで開発を行う。尚、本サンプルのDelphi/400バージョンは11 Alexandriaを使用する。

4-1.モバイル端末の画面レイアウト作成

Delphiの開発画面からファイルの「新規作成||マルチデバイスアプリケーション」を選択し、画面レイアウトの作成から行う。

【図９】のように、TLabelやTButtonなどの必要なコンポーネントを配置する。また、カメラ映像を表示するためにはTImageが必要となる。AlignをClientで設定して配置する。

次に非表示コンポーネントを配置する**【図１０】**。今回、アプリケーションテザリングを利用するために必須となるのがTTetheringManagerとTTetheringAppProfileのコンポーネントである。

TTetheringManagerはアプリケーション同士の接続を管理するためのコンポーネントである。AllowedAdaptersプロパティで同一ネットワークでの接続（NetWork）かBluetooth通信による接続（Bluetooth）かを選択可能である。本稿ではモバイル端末とPCは同一ネットワークに存在すると仮定し、デフォルト値であるNetworkとする。

TTetheringAppProfileは、アプリケーション間で共有するリソースを管理するコンポーネントである。Groupプロパティでは、共有対象のグループ名を入力する。Resourcesには、実際に共有を行いたい内容を設定する。追加したリソースに対し、Kindプロパティでは、Shared（デフォルト値）またはMirrorを設定可能である。データを共有する場合はSharedを設定し、データが共有される場合はMirrorを設定する。本稿ではモバイル端末がデータを共有する側のため、Sharedの設定とする。ResTypeプロパティでは、共有するデータ型を設定でき、Data（デフォルト値）またはStreamを指定可能である。

リソース0を追加し、カメラ映像の共有に使用する。リソース0のプロパティ値については**【図１１】**のとおりとする。MemoryStreamにてデータの連携を行うため、ResTypeをStreamとする点がポイントである。

また、その他の非表示コンポーネントとしてTTimerは接続状況とのチェックのために使用し、TCameraComponentはカメラ映像の表示のために使用する。

今回、カメラ映像の表示については画面起動時に行い、CameraComponent1のSampleBufferReadyイベントでimgCameraに表示する**【ソース１～３】**。

図 9　モバイル端末：画面レイアウト

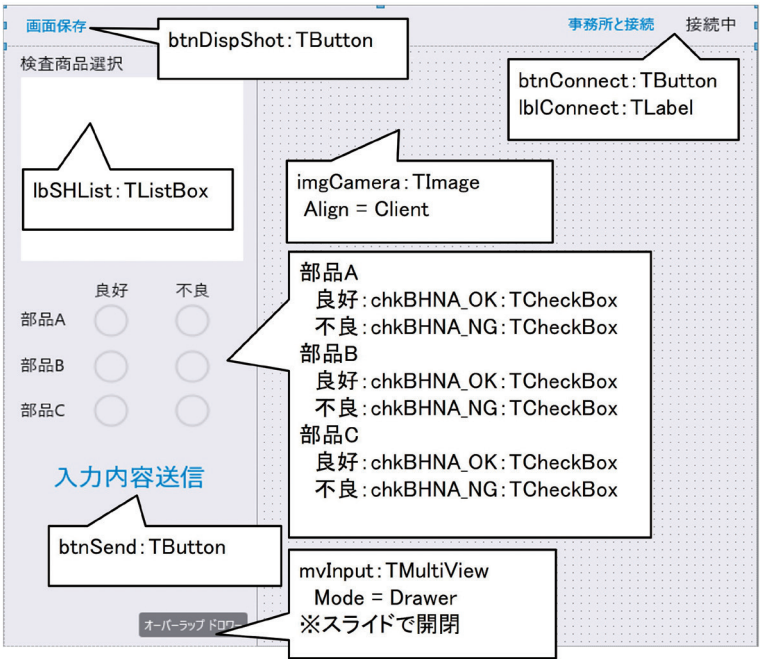
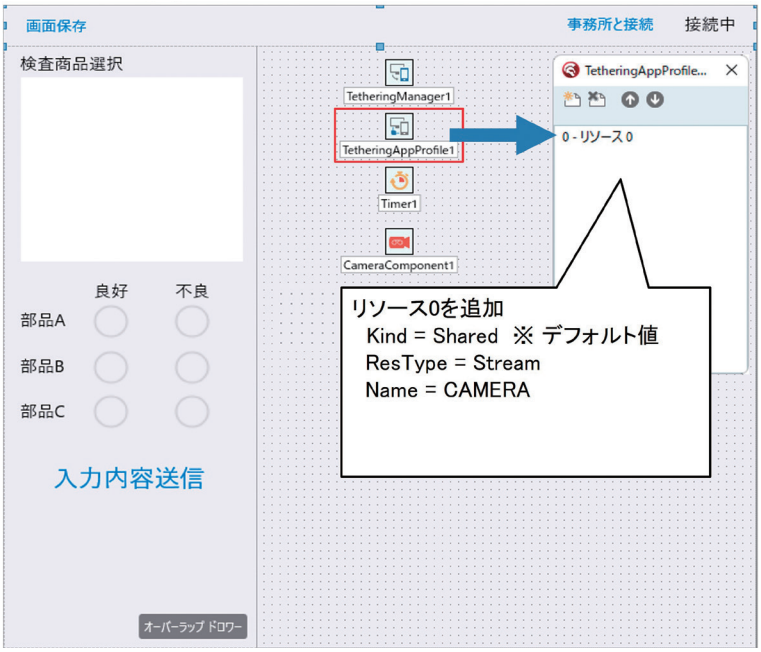


図 10　モバイル端末：非表示コンポーネント

図 11　モバイル端末:リソースの設定



ソース 1

FormCreateイベント

```
{*******************************************************************************
  目的 : 画面生成時処理
  引数 :
  戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.FormCreate(Sender: TObject);
var
  APPEventService:IFMXApplicationEventService;
begin
  // カメラの設定
  CameraComponent1.Kind := TCameraKind.BackCamera; // 背面カメラ
  CameraComponent1.Quality := TVideoCaptureQuality.PhotoQuality; // 高解像度写真レベル

  if TPlatformServices.Current.SupportsPlatformService(
       IFMXApplicationEventService) then
  begin
      APPEventService:=IFMXApplicationEventService(
        TPlatformServices.Current.GetPlatformService(
          IFMXApplicationEventService)
      );
  end;

  if (APPEventService <> nil) then
  begin
    APPEventService.SetApplicationEventHandler(AppEvent);
  end;
end;
```

ソース2で記述

ソース 2

AppEvent処理(privateで定義)

```
{*******************************************************************************
  目的 : ApplicationEvent処理
  引数 :
  戻値 :
*******************************************************************************}
function TfrmGENBAMobile.AppEvent(iAppEvent: TApplicationEvent; iContext: TObject): Boolean;
begin
  Result := False;
  case iAppEvent of
    TApplicationEvent.BecameActive:
    begin
      // Focus取得時
      CameraComponent1.Active := True;
      Sleep(100);
      CameraComponent1.Active := False;
      Sleep(400);

      // オートフォーカスモードに設定する
      CameraComponent1.FocusMode := TFocusMode.ContinuousAutoFocus;
      CameraComponent1.Active := True;
    end;

    TApplicationEvent.WillBecomeInactive:
    begin
      //focus喪失時
      CameraComponent1.Active := False;
    end;
  end;
end;
```

ソース 3

OnSampleBufferReadyイベント（カメラ映像の表示）

```
{*************************************************************************
  目的　：カメラ映像表示処理
  引数　：
  戻値　：
*************************************************************************}
procedure TfrmGENBAMobile.CameraComponent1SampleBufferReady(Sender: TObject; const ATime: TMediaTime);
begin
  TThread.Synchronize(
    TThread.CurrentThread,
    procedure
    begin
      // 画面にカメラ映像を表示
      CameraComponent1.SampleBufferToBitmap(imgCamera.Bitmap, True);
    end
  );
end;
```

4-2.PCの画面レイアウト作成

本稿では、現行のアプリケーションに改修を行う想定である。新しく追加した画面にてカメラ映像の共有を行う。**【図12】**のように、TLabelやTButtonなど必要なコンポーネントを配置し、カメラ映像表示のため、iPad側と同様にTImageを配置する。

次に非表示コンポーネントを配置する**【図13】**。

TetheringAppProfile1のGroupプロパティについてはiPad側と同一の値を設定する。また、Resourcesには同様にリソース0を追加するが、Kindプロパティには Mirrorを設定する**【図14】**。

商品照会画面にボタンを配置し、Showメソッドで起動可能にしておく**【図15】**。

図 12　PC端末：画面レイアウト

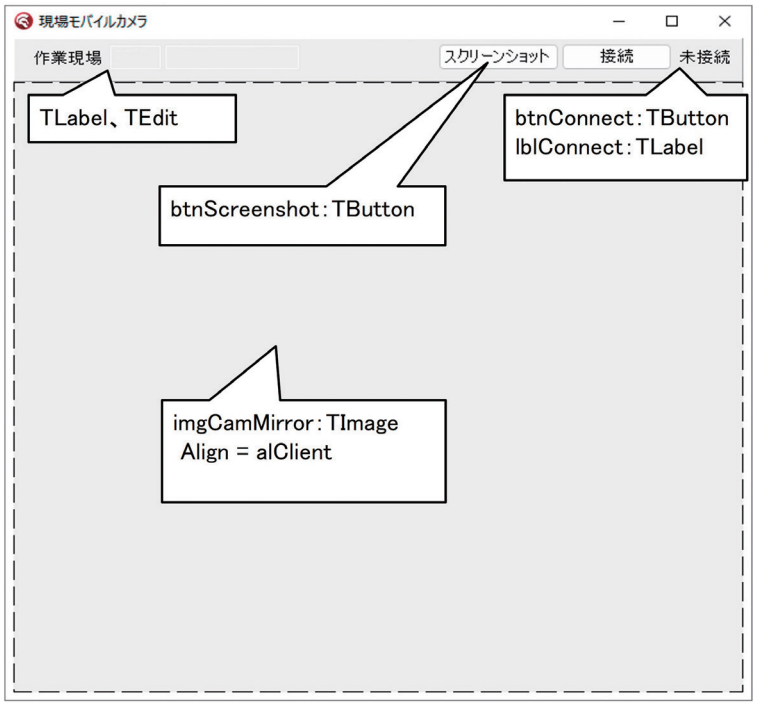


図 13 PC端末:非表示コンポーネント

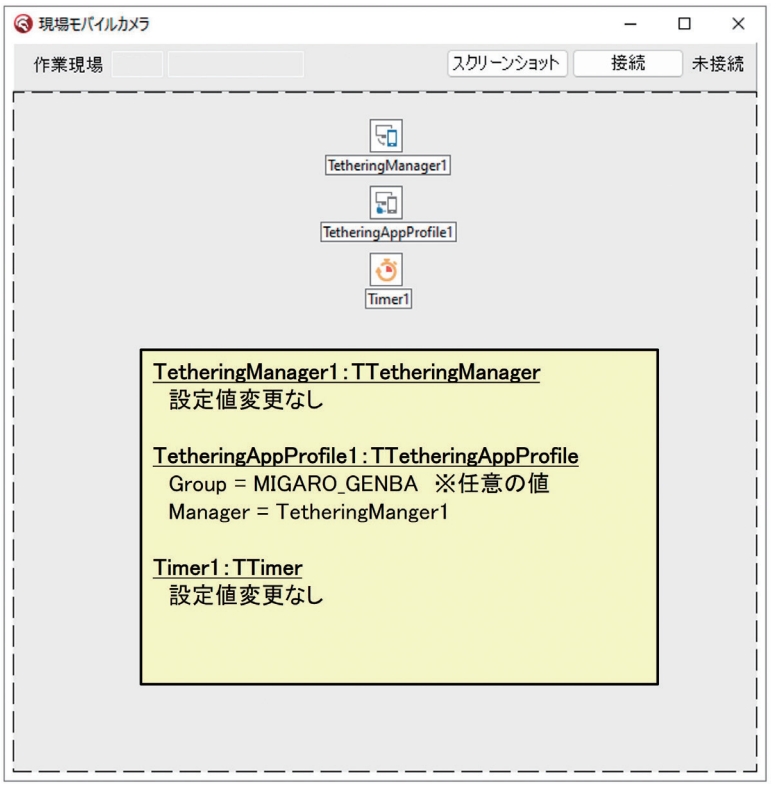


図 14 PC端末:リソースの設定

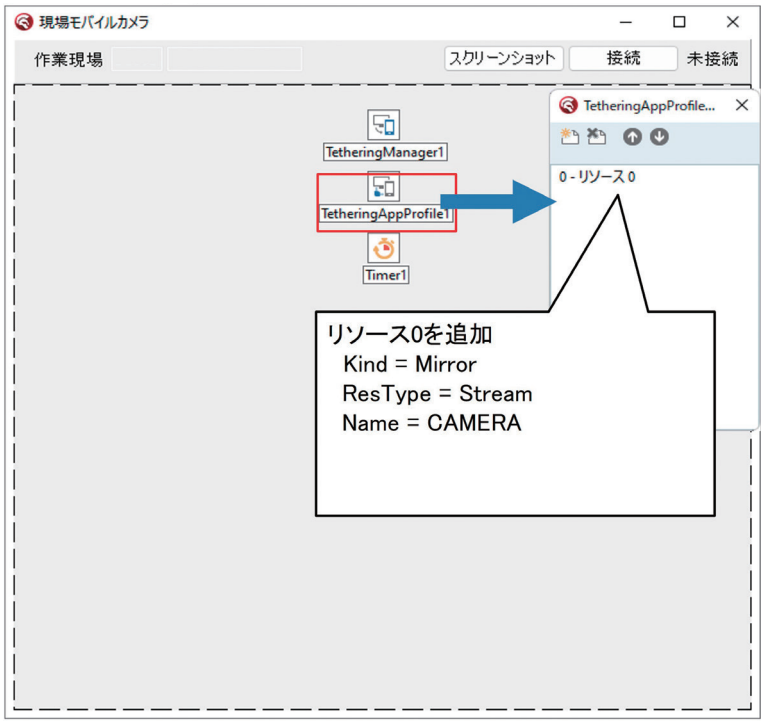
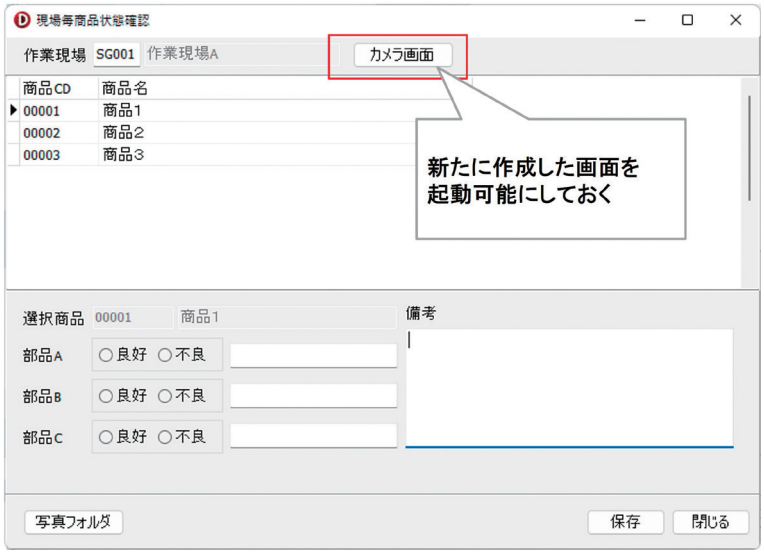


図 15　現行使用画面の改修



4-3.接続処理の実装

今回、接続ボタン押下で端末間の接続を実施する。端末間の接続はTetheringManager1のAutoConnectメソッドをそれぞれ呼び出すだけで可能である【ソース４】。但し、AutoConnectメソッドはそれぞれのGroupプロパティが同一でなければならないため注意が必要である。AutoConnectメソッドでは、２つ引数を指定可能である。１つ目はタイムアウト時間であり、接続可能対象の検知をどれくらいの時間実施するかをミリ秒で指定する。２つ目は接続対象の指定であり、AllowedAdaptersがNetWorkであればIPアドレス、Bluetoothであれば Bluetoothデバイス名を指定する。尚、この２つの引数については、指定無しでも実装可能である。

次にTimer1のOnTimer処理に接続状態の判定処理を追加する。判定には、TetheringManager1で検知されているプロファイルリスト（接続先情報）の数を利用しており、検知数が1以上の場合に接続中とする【ソース５】。

ソース4

OnClickイベント（接続ボタン押下時処理）

・モバイル端末

```
{*******************************************************************************
 目的 : 接続ボタン押下時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.btnConnectClick(Sender: TObject);
begin
  TetheringManager1.AutoConnect;
end;
```

・PC端末

```
{*******************************************************************************
 目的 : 接続ボタン押下時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.btnConnectClick(Sender: TObject);
begin
  TetheringManager1.AutoConnect;
end;
```

ソース5

OnTimerイベント（接続状態の監視）

・モバイル端末

```
{*******************************************************************************
 目的 : Timer処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.Timer1Timer(Sender: TObject);
begin
  if TetheringManager1.RemoteProfiles.Count > 0 then
  begin
    lblConnect.Text:='接続中';
  end
  else
  begin
    lblConnect.Text:='未接続';
  end;
end;
```

・PC端末

```
{*******************************************************************************
 目的 : Timer処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.Timer1Timer(Sender: TObject);
begin
  if TetheringManager1.RemoteProfiles.Count > 0 then
  begin
    lblConnect.Caption:='接続中'
  end
  else
  begin
    lblConnect.Caption:='未接続';
  end;
end;
```

4-4.文字列共有の実装

今回の例では、iPad側の入力内容送信ボタンで、選択している商品と各部品の状態（良好or不良）をPC側に送信する。PC端末側では、情報を受け取ったタイミングでダイアログ表示し、モバイル端末で入力した内容を商品の照会画面に反映する。

まず、文字列の送信側（iPad）ではTetheringManager1で取得したプロファイルリストをループする。合致するGroupとTextが見つかれば、SendStringメソッドを呼び出す。SendStringメソッドの引数には対象のプロファイル情報、送信文字列の説明（String）、送信文字列（String）を指定可能である。本稿では、送信文字列の説明部分にダイアログメッセージの内容を指定し、送信文字列に選択した商品内容と各部品の状態をカンマ区切りにて指定する**【ソース6】**。

一方、文字列の受信側（PC端末）ではTetheringAppProfile1のAcceptResourceイベントで受信許可の条件を設定する。今回は常に受信を許可するため、AcceptResourceにTrueを設定する一文だけを記述する**【ソース7】**。次にTetheringAppProfile1のResourceReceivedイベントでモバイル端末側から受け取った送信文字列を画面値に反映する処理を記述する**【ソース8】**。引数のAResourceには、送信側のSendStringで指定した内容が保持されている。AResource.Hintで送信文字列の説明、AResource.Valueで送信文字内容を取得可能である。

ソース6

OnClickイベント処理（入力内容送信ボタン押下時）

```
{*******************************************************************************
 目的 : 入力内容送信ボタン押下時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.btnSendClick(Sender: TObject);
var
  i: Integer;
  sInputInfo: String;
begin
  for i := 0 to TetheringManager1.RemoteProfiles.Count - 1 do
  begin
    // 送信対象のグループ、プロファイルテキストを探索
    if (TetheringManager1.RemoteProfiles[i].ProfileGroup = 'MIGARO_GENBA') and
       (TetheringManager1.RemoteProfiles[i].ProfileText = TetheringAppProfile1.Text) then
    begin
      // 入力情報（カンマ区切り）
      sInputInfo :=
        lbSHList.Items[lbSHList.ItemIndex] + ','  // 選択商品
      + BoolToStr(chkBHNA_OK.IsChecked) + ','     // 部品A
      + BoolToStr(chkBHNB_OK.IsChecked) + ','     // 部品B
      + BoolToStr(chkBHNC_OK.IsChecked);          // 部品C

      // 入力情報の送信
      TetheringAppProfile1.SendString(
        TetheringManager1.RemoteProfiles[i],
        lbSHList.Items[lbSHList.ItemIndex] + 'の検査完了',  // 送信内容の説明
        sInputInfo                                          // 送信内容
        );
    end;
  end;
end;
```

ソース7

AcceptResourceイベント（受信リソースの受取許可）

```
{*******************************************************************************
 目的 : 受信リソースの受取許可
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.TetheringAppProfile1AcceptResource(
  const Sender: TObject; const AProfileId: string;
  const AResource: TCustomRemoteItem; var AcceptResource: Boolean);
begin
  AcceptResource := True;
end;
```

ソース8

ResourceReceivedイベント（データ受取時処理）

```
{*******************************************************************************
 目的 : データ受取時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.TetheringAppProfile1ResourceReceived(const Sender: TObject; const AResource: TRemoteResource);
var
  slInputData: TStringList;
begin

  // 現場で入力送信実施のメッセージ表示
  ShowMessage(AResource.Hint);

  { 入力情報をStringListに保持
     slInputData[0] : 選択商品
     slInputData[1] : 部品A  良好(-1) or 不良(0)
     slInputData[2] : 部品B  良好(-1) or 不良(0)
     slInputData[3] : 部品C  良好(-1) or 不良(0)  }
  slInputData := TStringList.Create;
  slInputData.CommaText := AResource.Value.AsString;

  // 前画面：照会画面に値を反映する
  FmemSHNList.DisableControls;
  try
    // 編集対象の商品CDにレコード移動
    FmemSHNList.Locate('SHCD', Copy(slInputData[0], 1, 5), []);

    // 照会画面の値を編集
    FmemSHNList.Edit;
    FmemSHNList.FieldByName('BHNA').AsString := slInputData[1];
    FmemSHNList.FieldByName('BHNB').AsString := slInputData[2];
    FmemSHNList.FieldByName('BHNC').AsString := slInputData[3];
    FmemSHNList.Post;

    // レコード移動で画面値を反映
    FmemSHNList.First;
    FmemSHNList.Locate('SHCD', Copy(slInputData[0], 1, 5), []);
  finally
    FreeAndNil(slInputData);
    FmemSHNList.EnableControls;
  end;
end;
```

4-5.カメラ映像共有の実装

カメラ映像の共有はMemoryStreamでデータの受け渡しを行う。カメラ映像の送信側（iPad）では「4-1」で実装したCameraComponent1のOnSampleBufferReadyイベントに、TetheringAppProfile1のリソース0（Name:CAMERA）へデータ転送する処理を組み込む。リソース0はResTypeがStreamのため、画面のBitmap内容をStream型に変換する必要がある。その際、TBitmapのSaveToStreamでは受け取り側のVCLフレームワークで、正しく表示ができないため、JPEG形式への変換後、Stream保存を行う【ソース9】。

カメラ映像の受信側（PC端末）では、定義しているリソースのResourcesReceivedイベントに処理を記述する。今回はTThreadの機能を使用し、受け取ったStream情報（引数：AResource）をimgCamMirrorに描画する。Stream情報はJPEG形式で渡されているため、まずはTJPEGImage型の変数でLoadFromStreamメソッドを呼び出して読み込む。その後、サイズや位置を設定した後、imgCamMirrorのStretchDrawメソッドで画面に描画する【ソース10】。

ソース9

OnSampleBufferReadyイベント（情報送信処理の追加）

```
{*************************************************************************
 目的　：カメラ映像表示処理
 引数　：
 戻値　：
*************************************************************************}
procedure TfrmGENBAMobile.CameraComponent1SampleBufferReady(Sender: TObject; const ATime: TMediaTime);
var
  bmps :TBitmapSurface;
  pm   :TBitmapCodecSaveParams;
begin
  TThread.Synchronize(
    TThread.CurrentThread,
    procedure
    begin
      // 画面にカメラ映像を表示
      CameraComponent1.SampleBufferToBitmap(imgCamera.Bitmap, True);

      // JPEG形式に変換のため、TBitmapSurface形式で保持
      bmps := TBitmapSurface.Create;
      bmps.Assign(imgCamera.Bitmap);

      if FCameraStrm <> nil then
      begin
        FCameraStrm.DisposeOf;
      end;

      FCameraStrm := TMemoryStream.Create;

      // 品質80で保存(0-100)
      pm.Quality := 80;

      // JPEG形式に変換
      TBitmapCodecManager.SaveToStream(FCameraStrm, bmps, '.jpg' ,@pm);

      // 情報送信
      FCameraStrm.Position := 0;
      if TetheringManager1.RemoteProfiles.Count > 0 then
      begin
        TetheringAppProfile1.Resources.FindByName('CAMERA').Value := FCameraStrm;
      end;

      bmps.DisposeOf;
    end
  );
end;
```

ロジック追加

FCameraStrm：
private変数で
TMemoryStream型で
定義しておく

ソース10

ResourcesReceivedイベント（カメラ映像データ受取時処理）

```
{*******************************************************************************
 目的　：カメラ映像データ受取時処理
 引数　：
 戻値　：
*******************************************************************************}
procedure TfrmGNBCamera.TetheringAppProfile1ResourcesOResourceReceived(
  const Sender: TObject; const AResource: TRemoteResource);
var
  bmpTemp: TBitmap;
  jpgTemp: TJPEGImage;
  imgRect: TRect;
begin
  TThread.Synchronize(nil,procedure
  begin
    // クリア
    imgCamMirror.Picture.Bitmap := nil;
    AResource.Value.AsStream.Position := 0;

    bmpTemp := TBitmap.Create;
    jpgTemp := TJPEGImage.Create;
    try
      // JPEG形式で渡されたStreamを読み込む
      jpgTemp.LoadFromStream(AResource.Value.AsStream);

      // JPEGから一時Bitmapに描画
      bmpTemp.PixelFormat := pf32bit;
      bmpTemp.Width       := jpgTemp.Width;
      bmpTemp.Height      := jpgTemp.Height;
      bmpTemp.Canvas.Draw(0, 0, jpgTemp);

      // 位置調整
      imgRect.Left   := 20;
      imgRect.Top    := 0;
      imgRect.Right  := imgCamMirror.Width - 20;
      imgRect.Bottom := imgCamMirror.Height;

      // 一時Bitmapから画面に表示
      imgCamMirror.Canvas.StretchDraw(imgRect, bmpTemp);
    finally
      FreeAndNil(jpgTemp);
      FreeAndNil(bmpTemp);
    end;
  end);
end;
```

5.カメラ映像の写真保管

本章では、モバイル端末で表示しているカメラ映像を画像として切り取り、ファイルサーバーへ保存する方法について紹介する。

iPad側、PC端末側の両方にリソースを追加する。本章の処理は次章への準備も兼ねてTActionで処理を実装する。まず、iPad側ではTActionListコンポーネントを配置し、アクションを追加する**【図１６】【図１７】**。追加したacScreenShotアクションのExecute処理で、imgCameraのMakeScreenshot関数を使用し、Bitmap形式で保管する。その後、Stream形式に変換してPC端末側のリソースへ値をセットする**【ソース１１】**。PC端末側では、カメラ映像の共有時と同様にResourcesReceivedイベントに処理を記述する。受け取ったStream情報をTJPEGImageの変数に保持し、その後SaveToFileメソッドで引数に対象の保管先を記述するだけで処理は完成である**【ソース１２】【図１８】**。

図 16 モバイル端末：リソース、アクションの追加

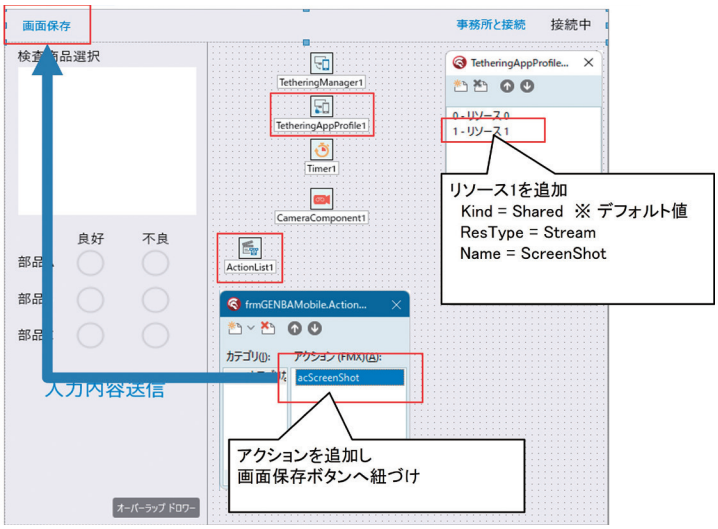
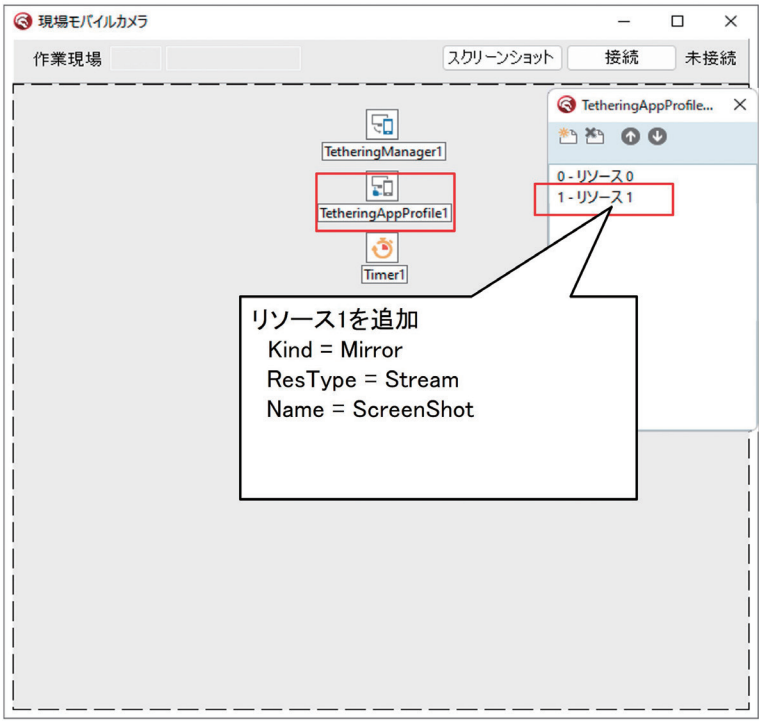


図 17 PC端末：リソースの追加



ソース11

acScreenShotExecuteイベント（カメラ映像を画像化して送信）

```
{*******************************************************************************
 目的 : 画面保存押下時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.acScreenShotExecute(Sender: TObject);
var
  memStream: TMemoryStream;
  bmps :TBitmapSurface;
  pm   :TBitmapCodecSaveParams;
begin
  bmps := TBitmapSurface.Create;

  // 画面画像をbmpsに保管
  bmps.Assign(imgCamera.MakeScreenshot);

  // MemoryStreamに変換
  memStream := TMemoryStream.Create;
  try
    //品質80で保存(0-100)
    memStream.Position := 0;
    pm.Quality := 80;
    TBitmapCodecManager.SaveToStream(memStream, bmps, '.jpg', @pm);

    // 転送
    memStream.Position := 0;
    TetheringAppProfile1.Resources.FindByName('ScreenShot').Value := memStream;
  finally
    bmps.DisposeOf;
    memStream.DisposeOf;
  end;
end;
```

ソース12

ResourcesReceivedイベント（カメラ画像データ受取時処理）

```
{*******************************************************************************
 目的 : カメラ画像データ受取時処理
 引数 :
 戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.TetheringAppProfile1ResourcesResourceReceived(const Sender: TObject;
  const AResource: TRemoteResource);
var
  jpgTemp: TJPEGImage;
begin
  TThread.Synchronize(nil,procedure
  begin
    AResource.Value.AsStream.Position := 0;

    jpgTemp := TJPEGImage.Create;
    try
      // JPEG形式で渡されたStreamを読み込む
      jpgTemp.LoadFromStream(AResource.Value.AsStream);

      // 画像の保管　※本日日付.jpg
      jpgTemp.SaveToFile(
        'C:¥Projects¥Migaro_TecnicalReport¥2022¥ScreenShot¥' + FormatDateTime('yyyymmddhhnnss', Now) + '.jpg');
    finally
      FreeAndNil(jpgTemp);
    end;
  end);
end;
```

図 18　画面保存実施の結果

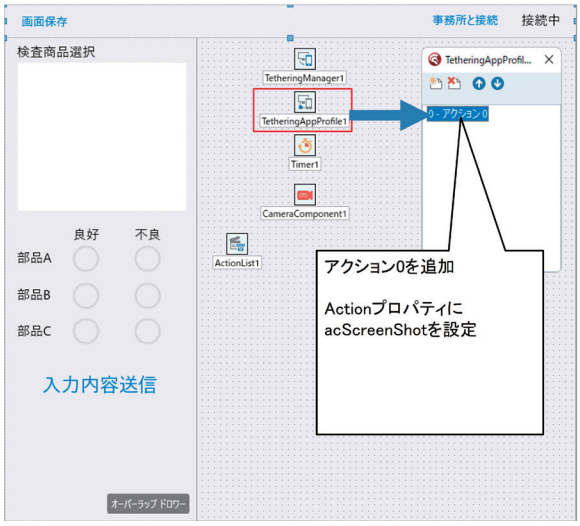
6.アクション処理の共有

アプリケーションテザリングでは、データの共有だけでなくアクション処理の共有も可能である。本章では第５章で作成した画面保存のアクション（acScreenShot）をPC端末側のスクリーンショットボタンからも実行できるように処理を実装する。

まずiPad側TetheringAppProfile1のActionsプロパティからアクション0（TLocalAction）を追加し、Actionプロパティに第５章で作成したacScreenShotを設定する**【図１９】**。

次にPC端末側のスクリーンショットボタンのOnClickイベントにて処理を実装する。アクション処理の実行についてはTetheringAppProfile1のRunRemoteActionメソッドを実行するだけで可能である。第１引数として、対象のプロファイルを指定し、第２引数として実行アクションのNameを指定する**【ソース１３】**。アクション処理はこの1文の処理記述のみで共有が可能である。

図 19　モバイル端末：TLocalActionの追加



ソース 13

OnClickイベント（スクリーンショットボタン押下時処理）

```
{*******************************************************************************
  目的　：スクリーンショットボタン押下時処理
  引数　：
  戻値　：
*******************************************************************************}
procedure TfrmGNBCamera.btnScreenShotClick(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to TetheringManager1.RemoteProfiles.Count - 1 do
  begin
    if (TetheringManager1.RemoteProfiles[i].ProfileGroup = 'MIGARO_GENBA') and
       (TetheringManager1.RemoteProfiles[i].ProfileText = 'TetheringAppProfile1') then
    begin
      TetheringAppProfile1.RunRemoteAction(TetheringManager1.RemoteProfiles[i], 'acScreenShot');
    end;
  end;
end;
```

7.複数端末使用時の切り替え処理実装

前章まではAutoConnectメソッドにより接続処理を実施した。AutoConnectメソッドは1文で接続処理が可能であるため、便利である。しかし、接続先を自動決定してしまうため、複数端末との連携時、自身で接続先を決定したい場合には利用が難しい。

本章では、作業現場Aと作業現場Bのカメラ映像をラジオボタンで切り替える処理を実装する。例では、各モバイル端末のIPアドレスを各端末の判定に利用する。

iPad側のFormCreateの処理でIPアドレス取得処理を追加する。取得したIPアドレスは、TetheringManager1とTetheringAppProfile1のTextプロパティにセットする**【ソース14】**。

次にPC端末側では、対象の接続先を選択するためのラジオボタンを追加する。併せて、private変数には

FConnectIPText：String

FManagerInfo：TTetheringManagerInfo

FProfileInfo：TTetheringProfileInfo

を定義しておく。

接続処理では

①リモートプロファイルの接続解除

②接続先ペアの解除

③画面クリア

④接続対象のチェック

⑤接続処理

の順で実装する。①～④は接続ボタン押下時処理で実施し**【ソース15】**、⑤はTetheringManager1のOnEndManagersDiscoveryイベント及びOnEndProfilesDiscoveryイベントにて実施する**【ソース16】**。

①リモートプロファイルの接続解除

TetheringAppProfile1のDisconnectメソッドを実行する。引数には接続解除対象のプロファイルを指定する（変数：FProfileInfo）。

②接続先ペアの解除

TetheringManager1のUnpairManagerメソッドを実行する。引数にはペア解除対象のマネージャーを指定する（変数：FManagerInfo）。

③画面クリア

接続数のクリア及びカメラ映像表示の初期化と再描画を実施する。

④接続対象のチェック

FConnectIPText変数に選択対象のIPアドレスを保持し、TetheringManager1及びTetheringAppProfile1のTextに値を設定する。その後、TetheringManager1のDiscoverManagersメソッドを実行する。DiscoverManagersメソッドの実行により、OnEndManagersDiscoveryイベントが実行される。

⑤接続処理

OnEndManagersDiscoveryイベントでは、④で保持したFConnectIPTextとManagerTextの値が合致する場合にPairManagerメソッドを実行して、ペアリングを行う。ペアリングを行うとOnEndProfilesDiscoveryイベントが実行される。TetheringManager1のProfileTextの値とTetheringAppProfile1のTextプロパティの値が合致するものを対象にConnectメソッドを実行する。

以上の設定で、自身で自由に対象のカメラ映像の切り替えが可能となる**【図20】**。

ソース 14

FormCreateイベント（IPアドレス取得処理を追加）

```
{*******************************************************************************
  目的 : 画面生成時処理
  引数 :
  戻値 :
*******************************************************************************}
procedure TfrmGENBAMobile.FormCreate(Sender: TObject);
var
  LAddr: TIdStackLocalAddress;
  LList: TIdStackLocalAddressList;
  i: Integer;
  sIPAdress: String;

  APPEventService:IFMXApplicationEventService;
begin
  LList := TIdStackLocalAddressList.Create;
  GStack.GetLocalAddressList(LList);
  try
    for i := 0 to LList.Count - 1 do
    begin
      LAddr := LList[i];
      case LAddr.IPVersion of
        Id_IPv4:
        begin
          if Copy(LAddr.IPAddress, 1, 7) = '192.168' then
          begin
            sIPAdress := LAddr.IPAddress;
          end;
        end;
      end;
    end;
  finally
    FreeAndNil(LList);
  end;

  // 各Textに取得したIPアドレスを設定
  TetheringManager1.Text := sIPAdress;
  TetheringAppProfile1.Text := sIPAdress;

  // カメラの設定
  CameraComponent1.Kind := TCameraKind.BackCamera; // 背面カメラ
  CameraComponent1.Quality := TVideoCaptureQuality.PhotoQuality; // 高解像度写真レベル

  if TPlatformServices.Current.SupportsPlatformService(
      IFMXApplicationEventService) then
  begin
      APPEventService:=IFMXApplicationEventService(
        TPlatformServices.Current.GetPlatformService(
          IFMXApplicationEventService)
      );
  end;

  if (APPEventService <> nil) then
  begin
    APPEventService.SetApplicationEventHandler(AppEvent);
  end;
end;
```

IPアドレスを取得

各Textプロパティに設定

ソース 15

OnClickイベント（接続ボタン押下時処理　変更）

・変更前

```
{*******************************************************************************
  目的 : 接続ボタン押下時処理
  引数 :
  戻値 :
*******************************************************************************}
procedure TfrmGNBCamera.btnConnectClick(Sender: TObject);
begin
  TetheringManager1.AutoConnect;
end;
```

・変更後

```
{*******************************************************************************
procedure TfrmGNBCamera.btnConnectClick(Sender: TObject);
begin
//  TetheringManager1.AutoConnect;

  // リモートプロファイルの接続解除
  TetheringAppProfile1.Disconnect(FProfileInfo);

  // 既に接続している ITetheringManager があれば、ペア設定を解除
  TetheringManager1.UnPairManager(FManagerInfo.ManagerIdentifier);

  // 画面値クリア
  TetheringManager1.RemoteProfiles.Count := 0;
  Image1.Picture.Bitmap := nil;
  Image1.Repaint;

  // 現場A接続
  if rbGenbaA.Checked then
  begin
    FConnectIPText := '192.168.0.113';
  end
  else
  // 現場B接続
  if rbGenbaB.Checked then
  begin
    FConnectIPText := '192.168.0.106';
  end;

  TetheringManager1.Text := FConnectIPText;
  TetheringAppProfile1.Text := FConnectIPText;
  TetheringManager1.DiscoverManagers;
end;
```

①

②

③

④

ソース16のイベントが実行される

ソース 16

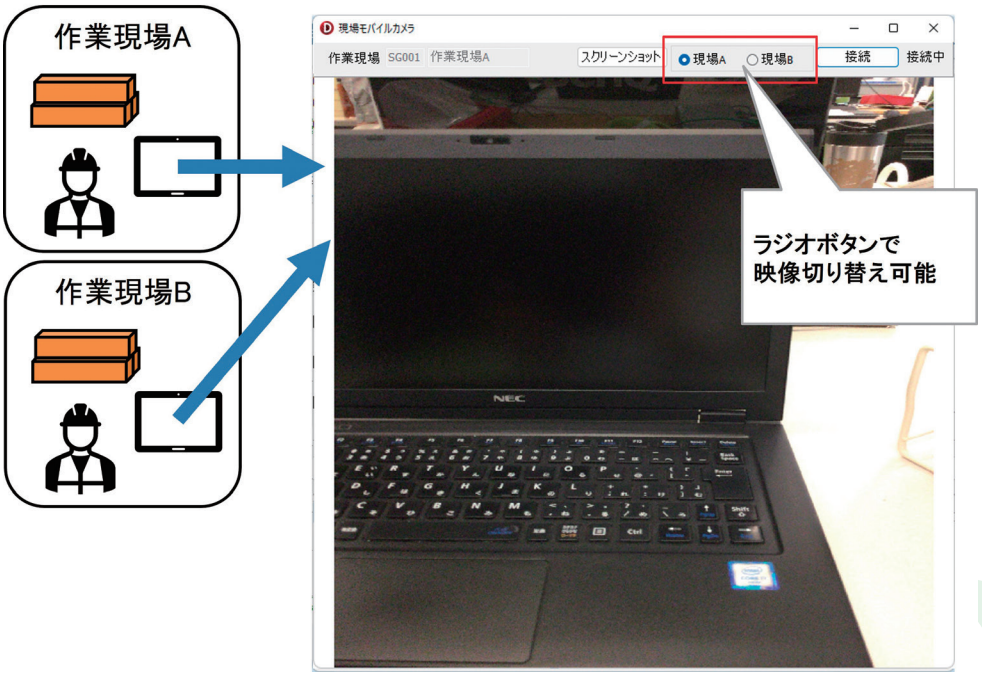
OnEndManagersDiscoveryイベント（ペアリング対象の設定）

```
{*****************************************************************************
  目的 : DiscoverManager完了時
  引数 :
  戻値 :
*****************************************************************************}
procedure TfrmGNBCamera.TetheringManager1EndManagersDiscovery(const Sender: TObject;
  const ARemoteManagers: TTetheringManagerInfoList);
var
  i: Integer;
begin
  for i := 0 to ARemoteManagers.Count - 1 do
  begin
    FManagerInfo := ARemoteManagers[i];
    if FManagerInfo.ManagerText = FConnectIPText then
    begin
      TetheringManager1.PairManager(FManagerInfo);
      Break;
    end;
  end;
end;
```

OnEndProfilesDiscoveryイベント（Connect対象の設定）

```
{*****************************************************************************
  目的 : リモートプロファイル検知完了時
  引数 :
  戻値 :
*****************************************************************************}
procedure TfrmGNBCamera.TetheringManager1EndProfilesDiscovery(const Sender: TObject;
  const ARemoteProfiles: TTetheringProfileInfoList);
var
  i: Integer;
begin
  for i := 0 to TetheringManager1.RemoteProfiles.Count - 1 do
  begin
    FProfileInfo := TetheringManager1.RemoteProfiles[i];
    if TetheringAppProfile1.Text = FProfileInfo.ProfileText then
    begin
      // リモートプロファイルに接続する
      if TetheringAppProfile1.Connect(FProfileInfo) then
      begin
        Break;
      end;
    end;
  end;
end;
```

図 20　複数端末の切り替え



8.おわりに

カメラ映像の共有というと高度な技術が必要に感じたかもしれないが、実際は簡単に処理が実装できた。カメラ映像の共有だけであれば、ビデオ通話アプリなどを使用すれば実現可能である。しかし、Delphi/400を使用すると、現行の業務アプリケーションと直結したアプリケーションを自由に作成できることがお分かりいただけたと思う。

過去、Delphi/400でモバイル端末を使用したアプリケーション開発というとDataSnapサーバーの構築を併せてご紹介するケースが多かった。

しかし、アプリケーションテザリングの技術を利用するとDataSnapサーバーの用意が不要であるため、気軽に開発を行うことができる。同一ネットワークの環境下であれば、アプリケーションテザリングを利用してアプリケーション開発を行うのもひとつの選択肢である。既にDataSnapサーバーを構築されている場合であれば、部分的にアプリケーションテザリングの技術を採用するのもよいだろう。

今回はVCLアプリケーションとFireMonkeyアプリケーションを連携例としたが、VCLアプリケーション同士やFireMonkeyアプリケーション同士の連携ももちろん可能である。

第7章で複数端末を使用した一例をご紹介したが、チャットアプリケーションのような複数対複数でのデータのやり取りも可能である。本稿で紹介した内容により、アプリケーション開発の幅が広がれば幸いである。

Delphi/400

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。

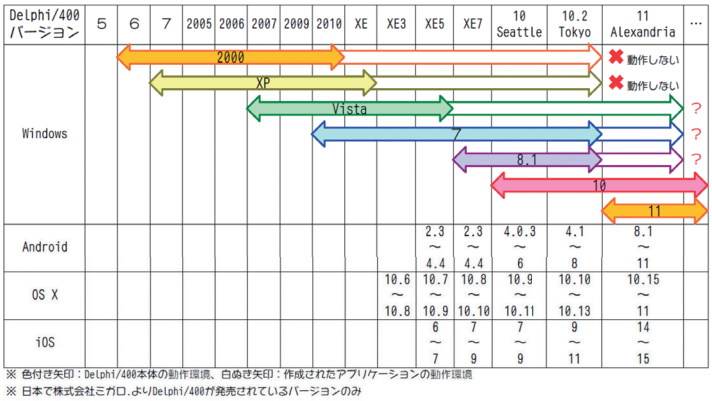
本稿では最新のDelphi/400 11 Alexandria(Update 2)を使用し、2022年9月時点での開発環境や開発手順、配布・運用のポイントを解説する。

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】**

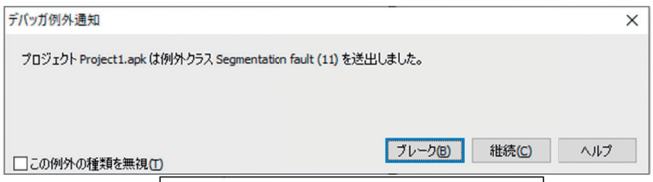
- Windows 11（Windows 7以上・Windows Server 2016以上）
- macOS 12 Monterey（OSX 10.15 Catalina以上）
- iOS 15（iOS 14以上）
- Android 12（Android 8.1以上）

図 1　サポートされているバージョン



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

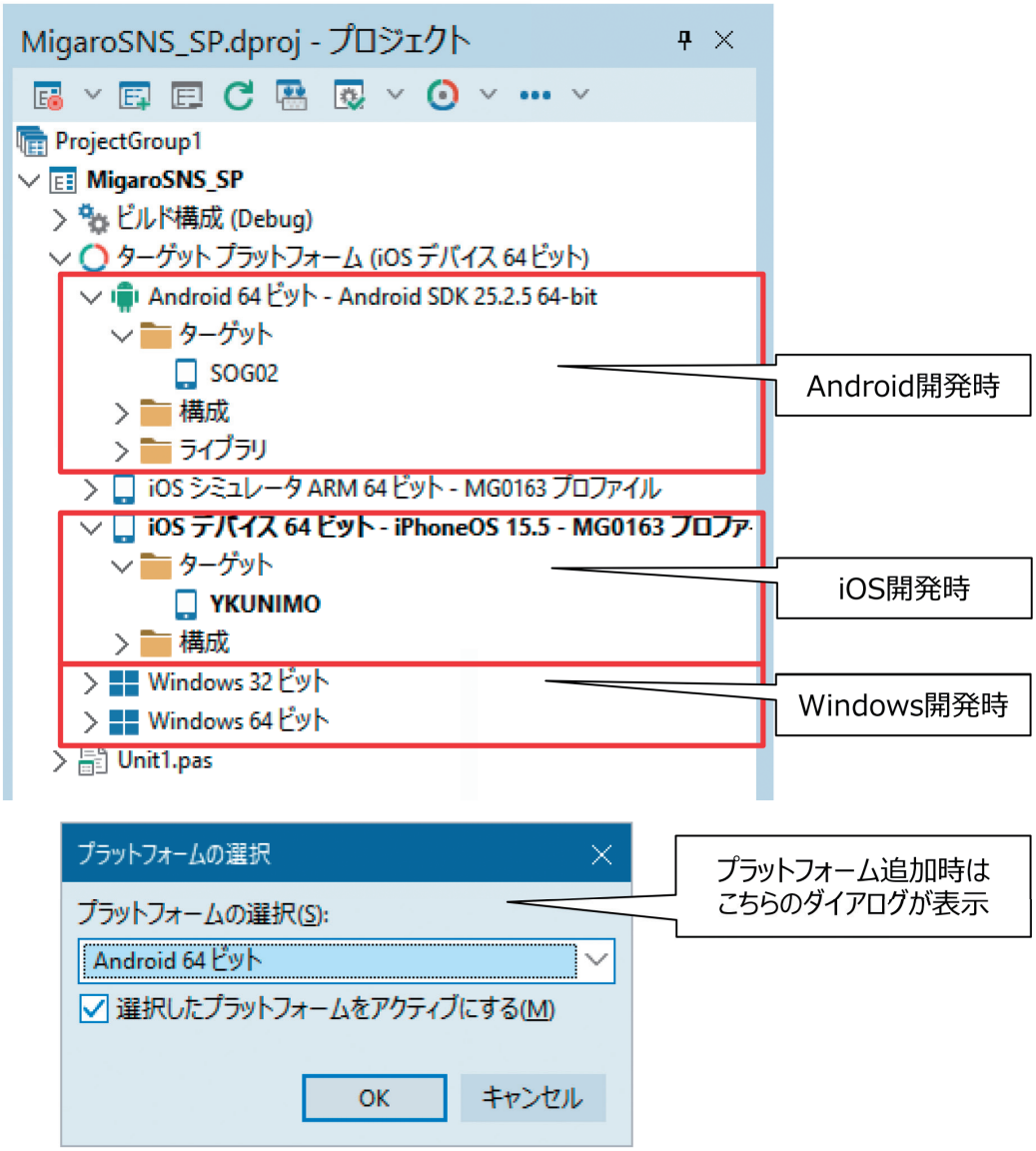
なお、後から動作対象のプラットフォームを追加する必要が生じた場合でも、Delphiを再インストールする必要はない。Delphi開発画面のメニューから［ツール］プラットフォームの管理］を選択することで、【図3】のプラットフォームの選択画面が表示されるので、ここからチェックを切り替えて適用ボタンを押すことで対象項目のインストールが実行される。

＜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）

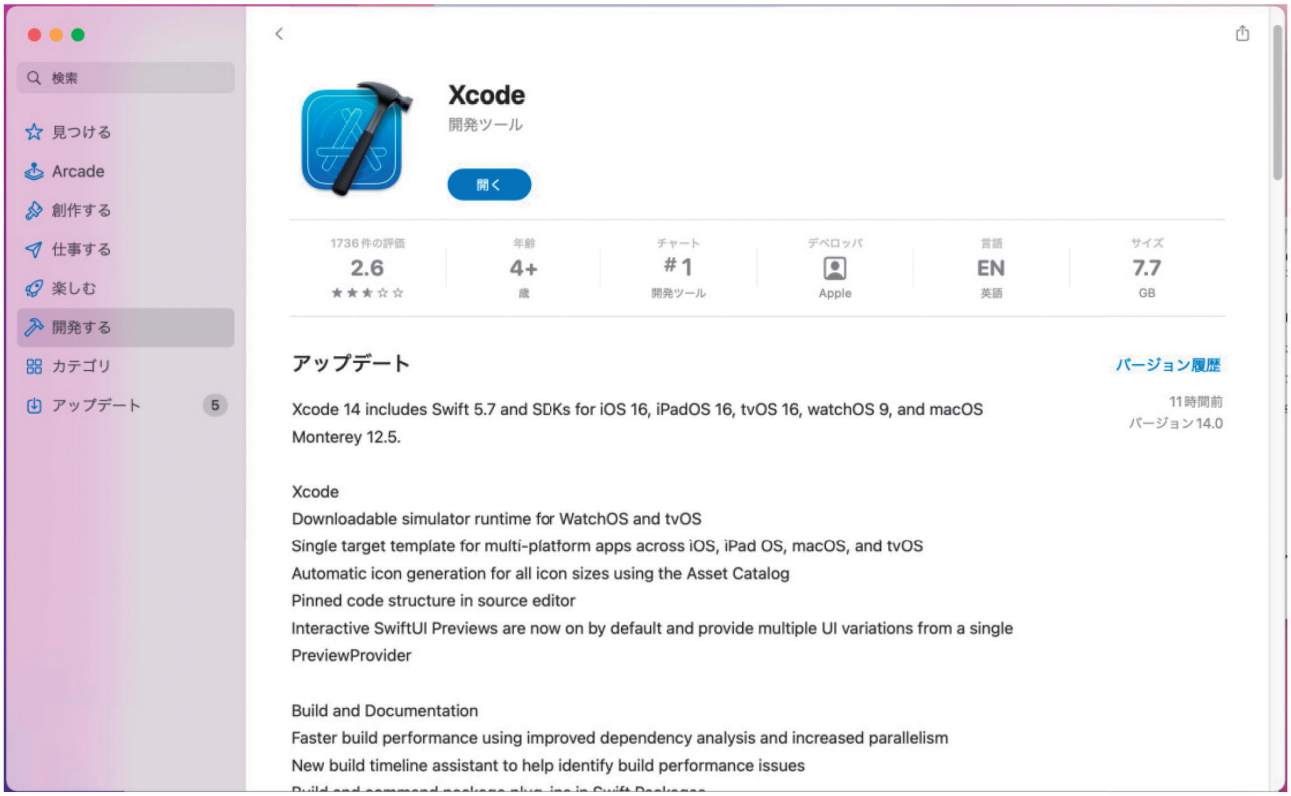
＜Mac環境の構築＞

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、

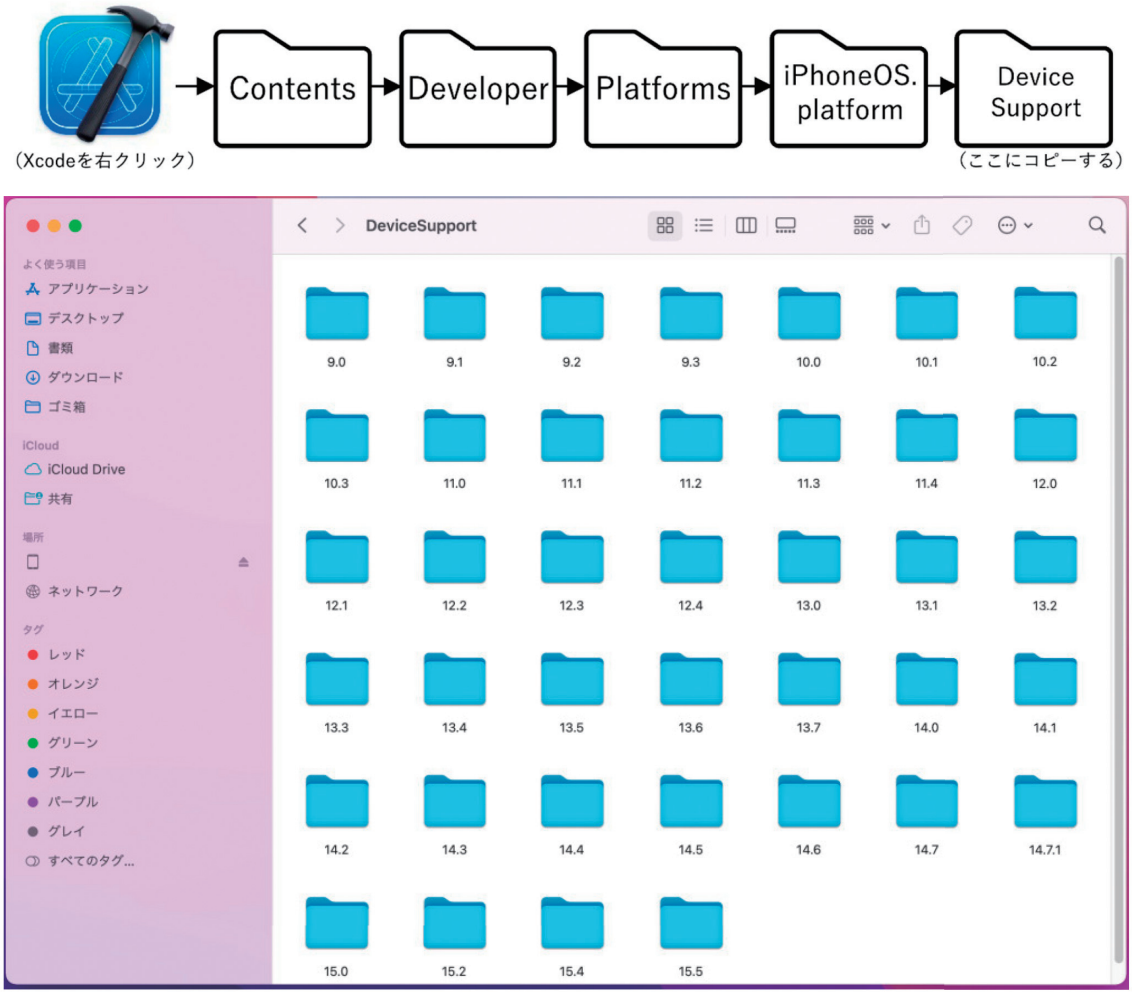
https://github.com/iGhibli/iOS-DeviceSupport/tree/master/DeviceSupport

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

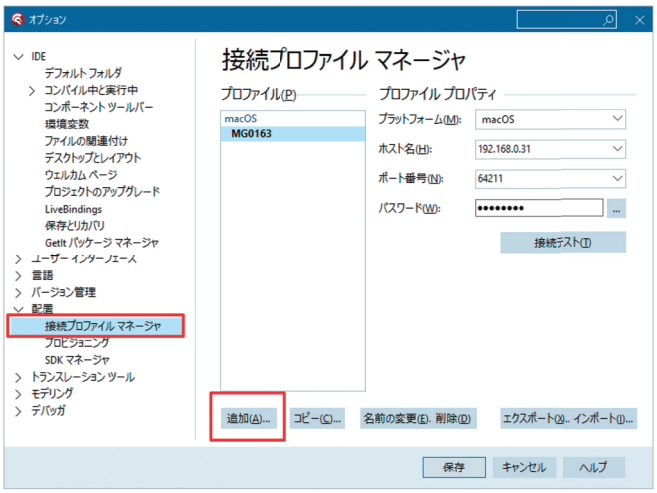
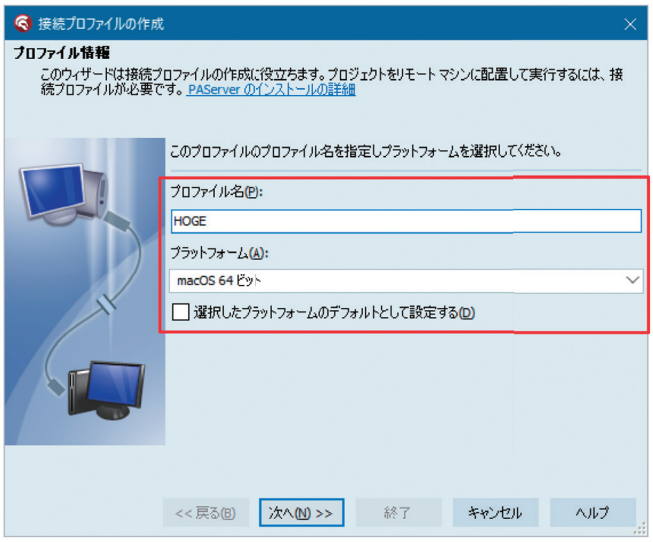


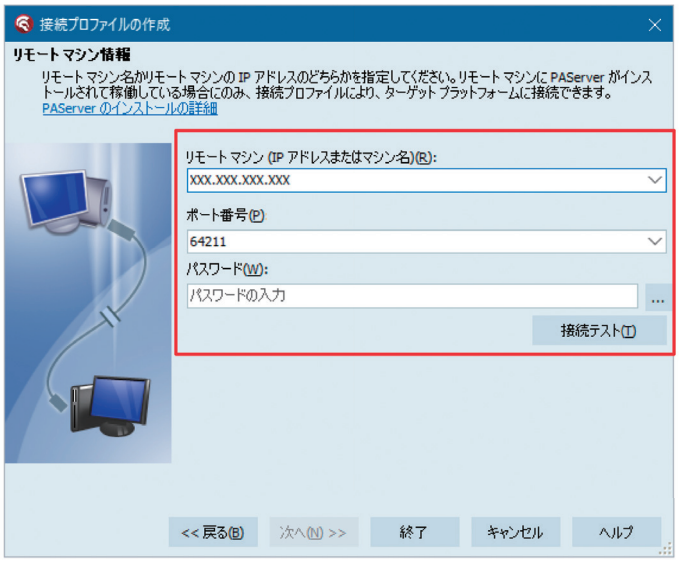
図 8　接続プロファイルマネージャ②



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。

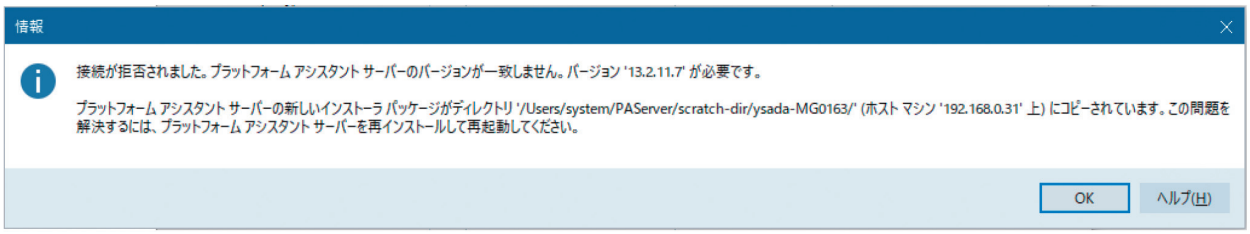
図9　接続プロファイルマネージャ③



＜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。

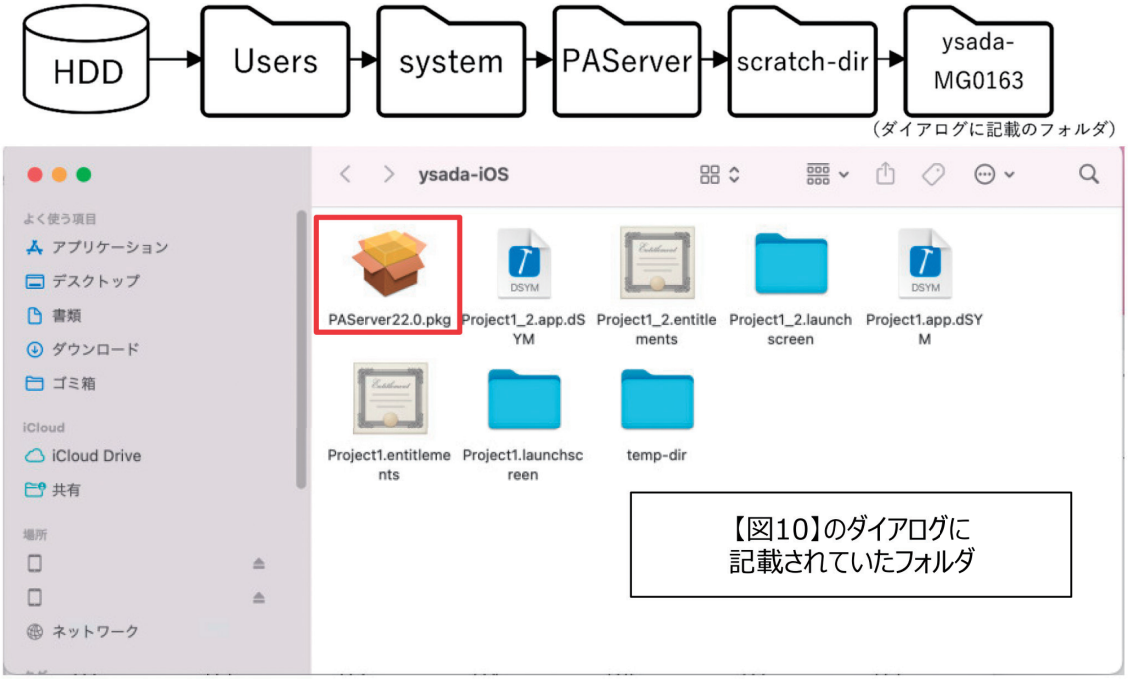
図10　接続プロファイルマネージャ④



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】

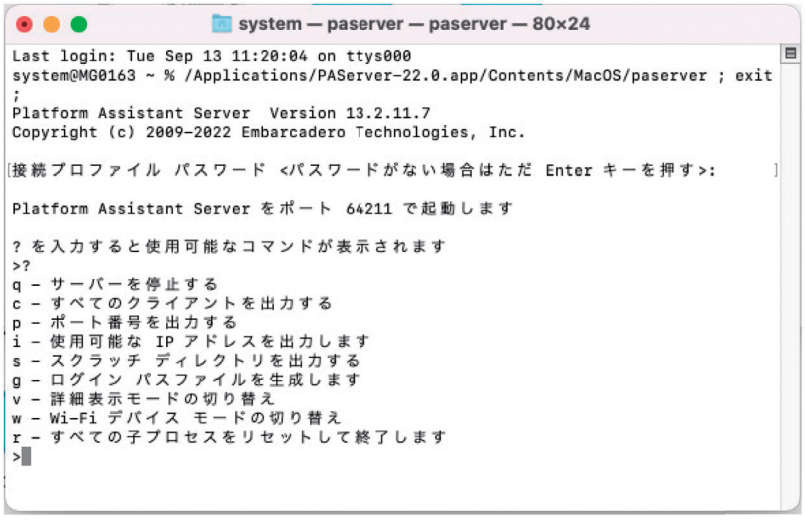
図11 PAServerのインストール



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。

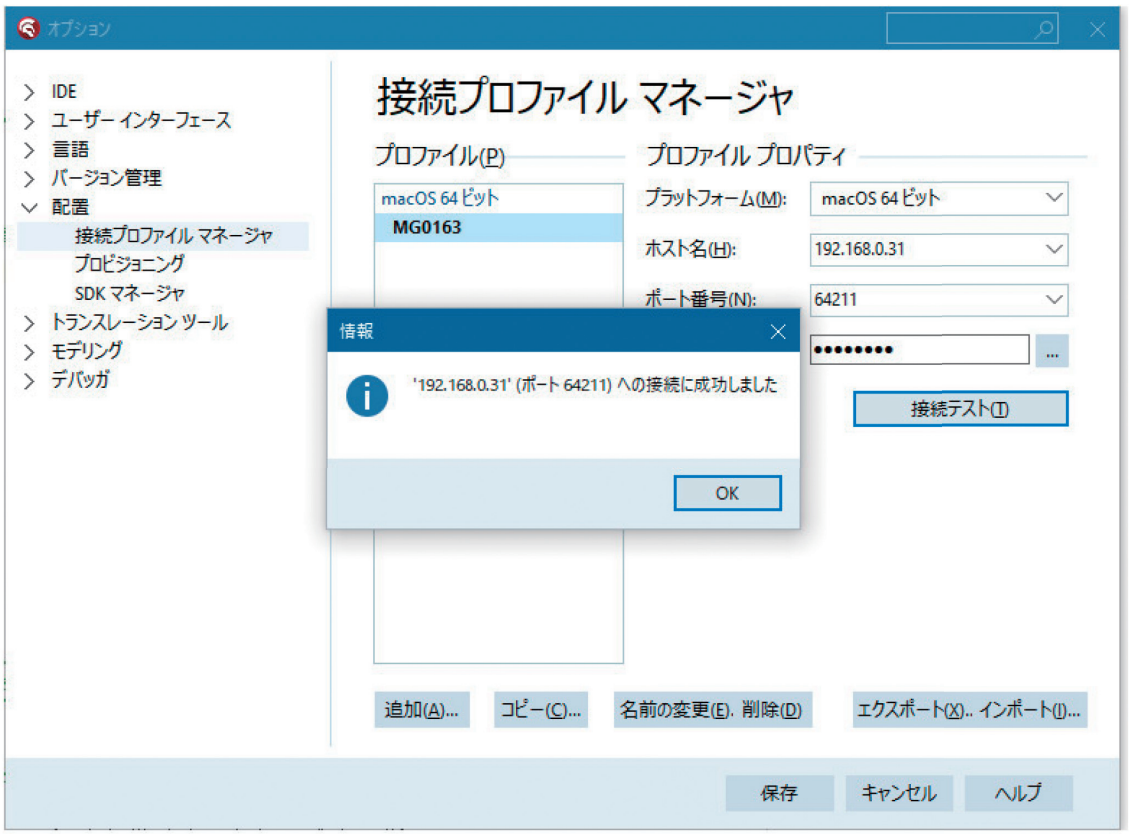
PAServerが起動するとコンソール画面で「Enterキーを押す」と表示されるので、[Enter]キーを押してサービスを開始する。【図12】

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）



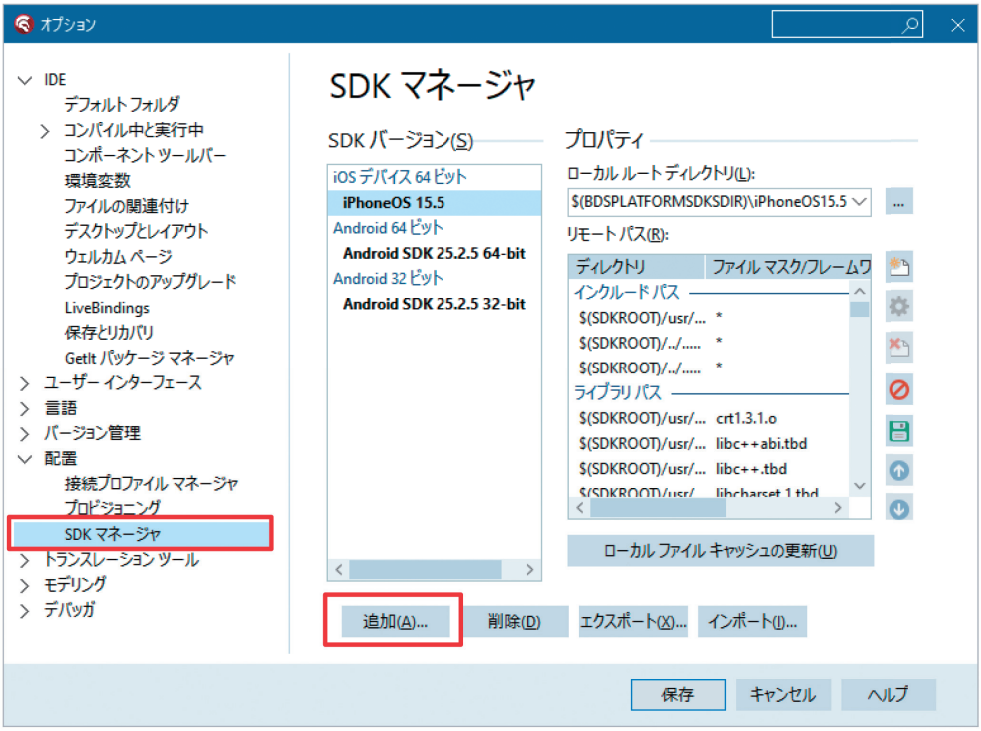
＜SDKの取得＞

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



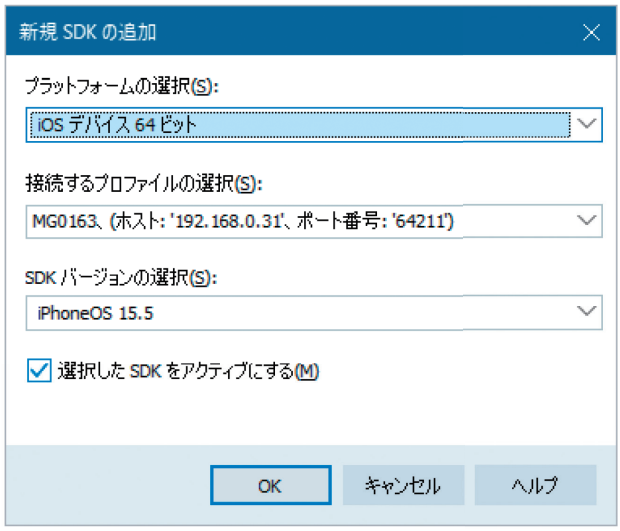
追加ボタンを押すと**【図15】**のダイアログ画面が表示される。

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

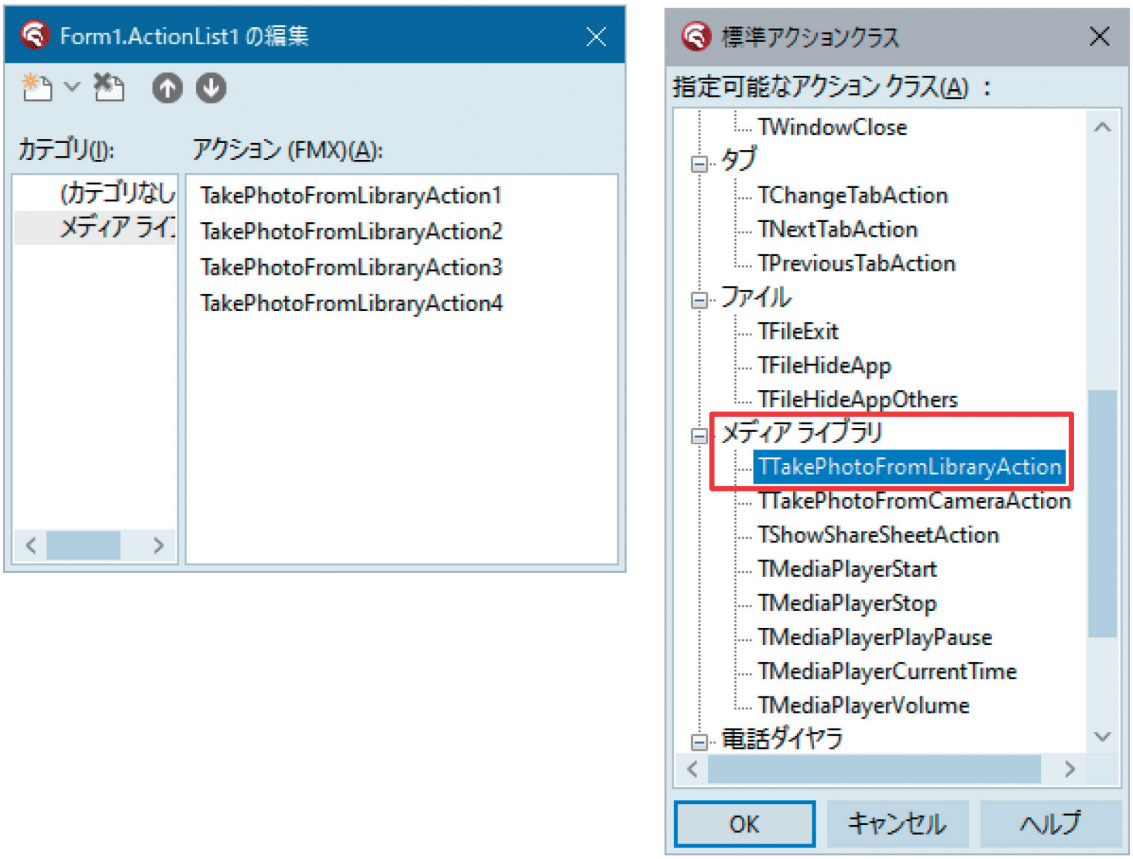
```
{*******************************************************************************
 目的: 写真１～４ボタン 押下時処理 (Windows)
*******************************************************************************}
procedure TForm1.btnPhotoWinClick(Sender: TObject);
var
  img: TImage;
begin
  // 対象コンポーネントの指定
  if (Sender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

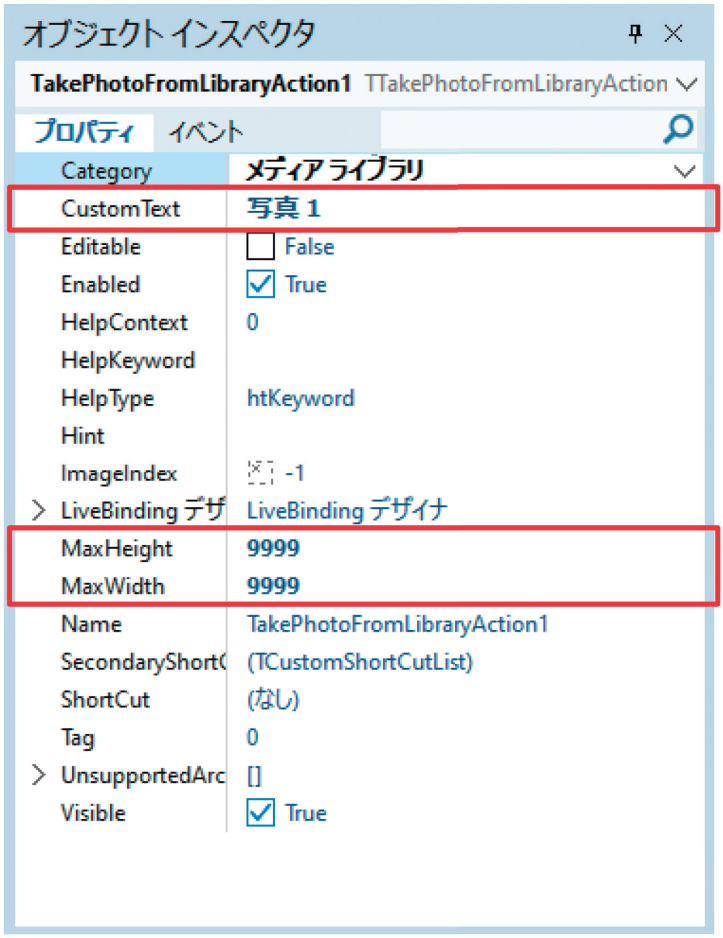


オブジェクトインスペクタにて、作成した各TTakePhoto
FromLibraryActionのプロパティに

- 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

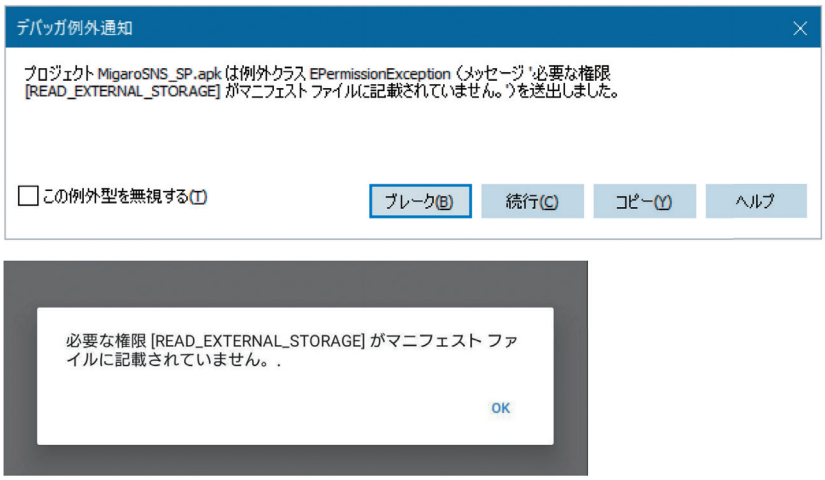
iOS/Androidの写真ボタンアクション押下時処理

```
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;
```

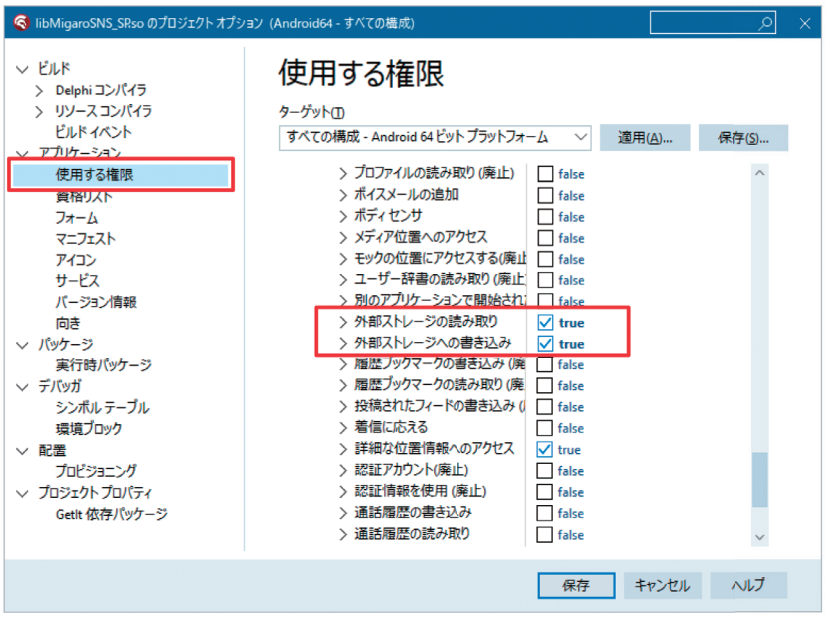
ここまででプログラムをコンパイルして実行すると、Windowsでは正常に動作するが、Android端末ではアクセス権限が不足している旨のエラーになる。**【図20】**

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);
```

ソース10

```
  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

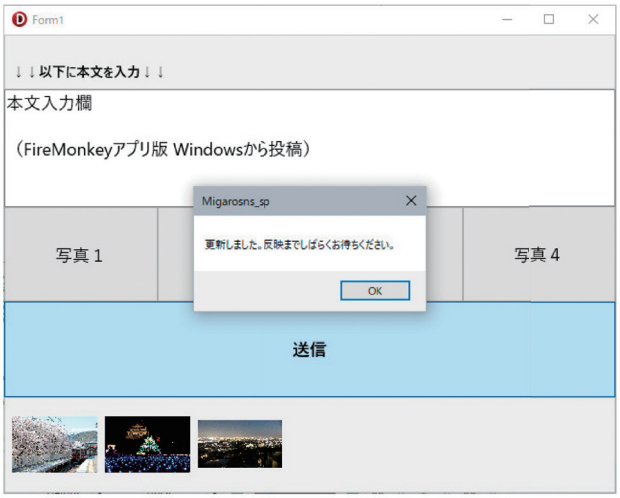


図 22-2　サンプルアプリの実行イメージ

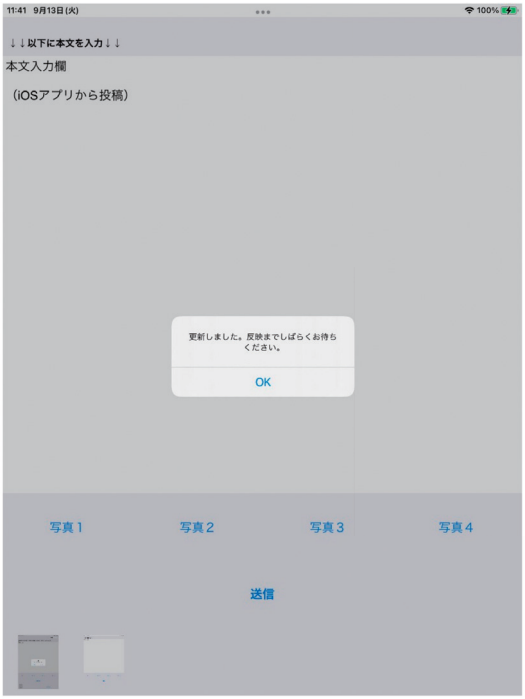
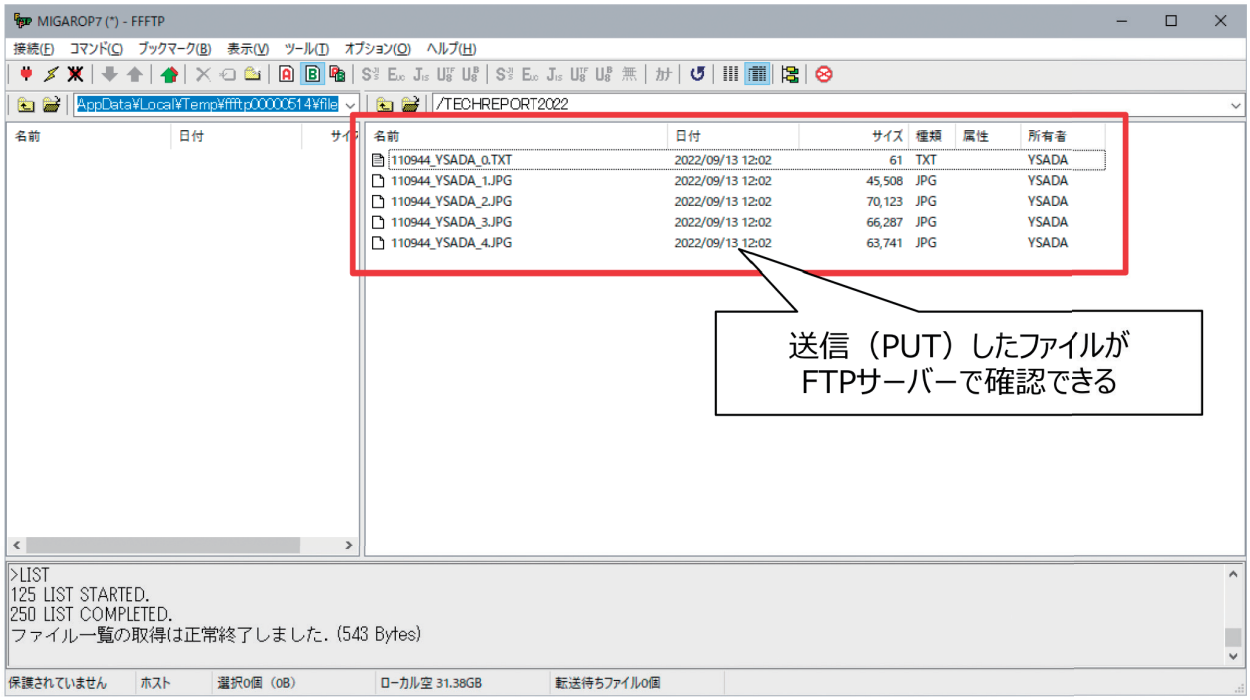


図 22-3　サンプルアプリの実行イメージ

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。

SmartPad4i

SmartPad4iで電子サインを実現する方法

株式会社ミガロ.
プロダクト事業部
技術支援課
國元 祐二

略　歴

生年月日:1979年3月27日
最終学歴:2002年　追手門学院大学　文学部アジア文化学科卒業
入社年月:2010年10月　株式会社ミガロ. 入社
社内経歴:2010年10月　RAD事業部(現プロダクト事業部)配属

現在の仕事内容:

Cobos4i(SP4i、JC/400)、Valenceの製品試験やサポート業務、導入支援などを担当している。

1.はじめに

最近、携帯電話の契約手続きを行う機会があった。
店員からタブレットを見せられながら、料金プランや契約関連の説明を受けた後、タブレットに指でサイン(名前の記入)を求められた。過去、生命保険の手続き時にも同様にタブレットを使用してサインを行ったケースもある。
今まで、紙媒体で実施されてきた手続きが、スマートデバイスに置き換わっていることや、スマートデバイスでの電子サインが、意思確認の手段として普及してきていることを日々の生活の中で実感している。

電子サインを実装できれば、対面の手続き、報告業務などで業務効率化やコスト削減、ペーパーレス化なども実現可能である。
もちろん、SmartPad4iアプリケーションでも電子サインの機能を実装することは可能だ。本稿では、SmartPad4iを使用して、電子サインの機能を実装する方法について説明する。

2.canvas要素の基本

2-1.HTML5のcanvas要素

電子サインのインターフェースにはcanvas要素を使用する。canvas要素はHTML5で追加された要素だ。HTMLでは、canvasタグで描画領域を定義して、canvas要素への描画はJavaScriptで記述する必要がある。タグの例は**【ソース1】**だ。

ソース 1

HTML canvasタグの例

```
<canvas id="CANVAS" width="600px" height="300px">
  <p>お使いのブラウザではCanvasが使用できません</p>
</canvas>
```

canvas要素の描画領域はwidth属性とheight属性で指定する。未指定の場合width属性は300px,height属性は150pxが設定される。CSSのwidthとheight設定では描画領域を指定できず、見た目上のサイズが変更されてしまうため、引き伸ばされた画像となるため注意が必要だ。

また、canvas要素に未対応のブラウザでは、描画した内容を表示することができない。canvasタグの中にhtmlを記述することで、未対応のブラウザで表示するための代替コンテンツを設定できる。

2-2. canvas要素への描画

2-2-1. 図形の描画

まずは、canvas要素で定義した領域に描画をする簡単な例を紹介する。canvas要素への描画は「Canvas API」をJavaScriptから使用する。
四角形を描画する例は**【ソース2】**だ。

ソース2

JavaScript canvasへの描画例（四角形）

```
<script>
 window.onload = function(){
  //canvas要素の取得
  var canvas = document.getElementById('CANVAS'); ----------①
  //描画コンテキストの取得
  var context = canvas.getContext('2d'); ----------②
  //左から250 , 上から75 の位置へ 横幅100 , 縦幅150の四角形を描画
  context.fillRect(250,75,100,150); ----------③
 }
</script>
```

【ソース2】 1行目のwindow.onload = function(){}はHTMLを読み込んだ際に処理を実施するための記述だ。HTMLを読込後、【ソース2】①のdocument.getElementByIdでHTML上のcanvas要素を取得する。

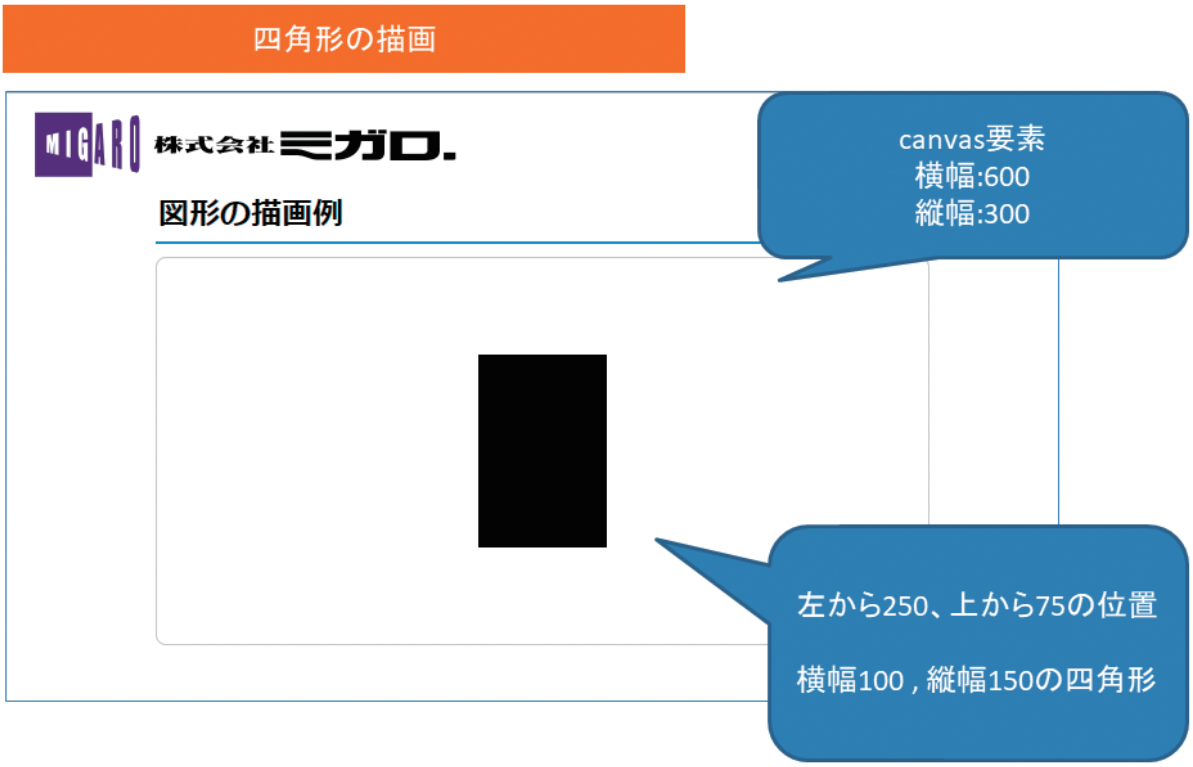
次に、【ソース2】②canvas要素のgetContextメソッドで描画コンテキストを取得する。描画コンテキストはグラフィック描画するためのメソッドや、プロパティを持つオブジェクトだ。

getContextメソッドの引数には、contextType（文字列）を設定する。"2d"を指定すると、2次元のレンダリングを行う描画コンテキストが作成される。"webgl"または"webgl2"では3次元のレンダリングを行う描画コンテキストが作成され、"bitmaprenderer"はcanvas要素の内容を特定のフォーマットに置き換える機能を持つオブジェクトが作成される。

電子サインや四角形の描画では2次元レンダリングを使用するため、getContextの引数に"2d"を指定している。context変数には、「CanvasRenderingContext2D」という描画コンテキストが格納される。

四角形を描画する場合には、【ソース2】③のように描画コンテキストのfillRectメソッドを使用する。引数は4つで、1つ目、2つ目が描画開始点の横座標と縦座標、3つ目が横幅、4つ目が縦幅だ。表示される四角形は【図1】になる。

図1 ソース2で描画される内容



2-2-2. 線の描画

次は線を描画する方法について説明する。線の描画は
【ソース3】だ。

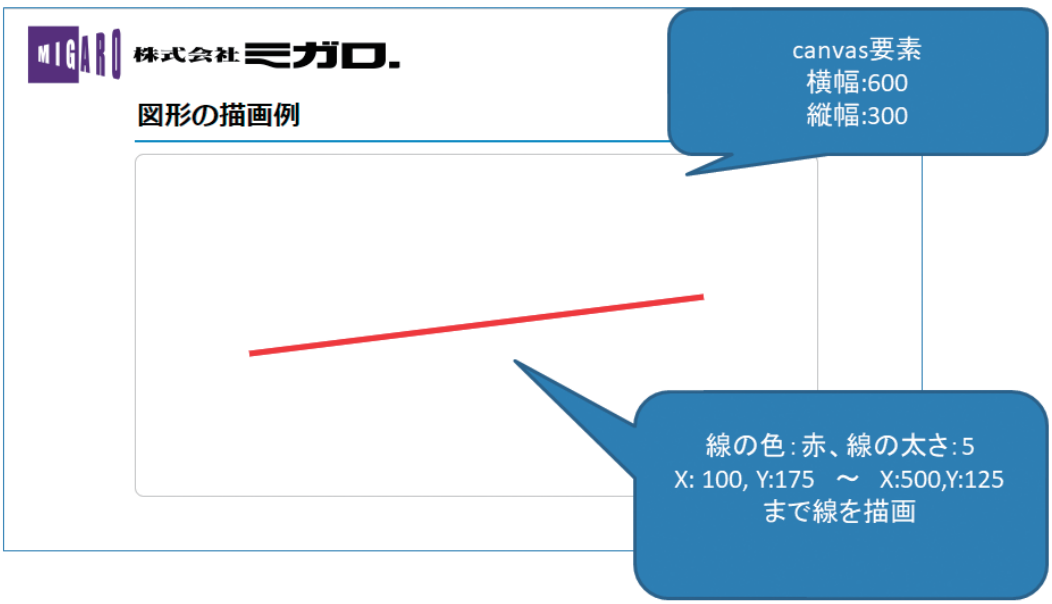
ソース 3

JavaScript canvasへの描画例（線）

```
<script>
  window.onload = function(){
    //canvas要素の取得
    var canvas = document.getElementById('CANVAS');
    //描画コンテキストの取得
    var context = canvas.getContext('2d');
    //線の色
    context.strokeStyle = 'red'; ----------①
    //線の太さ
    context.lineWidth = 5; ----------②
    //線の始点
    context.moveTo( 100 , 175 ); ----------③
    //線の終点
    context.lineTo( 500 , 125 ); ----------④
    //描画
    context.stroke(); ----------⑤
  }
</script>
```

描画コンテキストを取得後、**【ソース3】**①strokeStyleで線の色を指定する。デフォルトは黒で、色の文字列（red, yellow等）、または、RGB値で指定できる。**【ソース3】**②lineWidthでは線の太さを指定する。
線の描画は、**【ソース3】**③～⑤になる。moveToメソッドで線の始点を設定し、次にlineToメソッドで線の終点を指定する。最後にstrokeメソッドを実行するとcanvas要素に、指定した線の色、太さで始点から終点までの線が描画される。**【図2】**

図2　ソース3で描画される内容



3.電子サインの実装方法

3-1. 電子サインの実装例

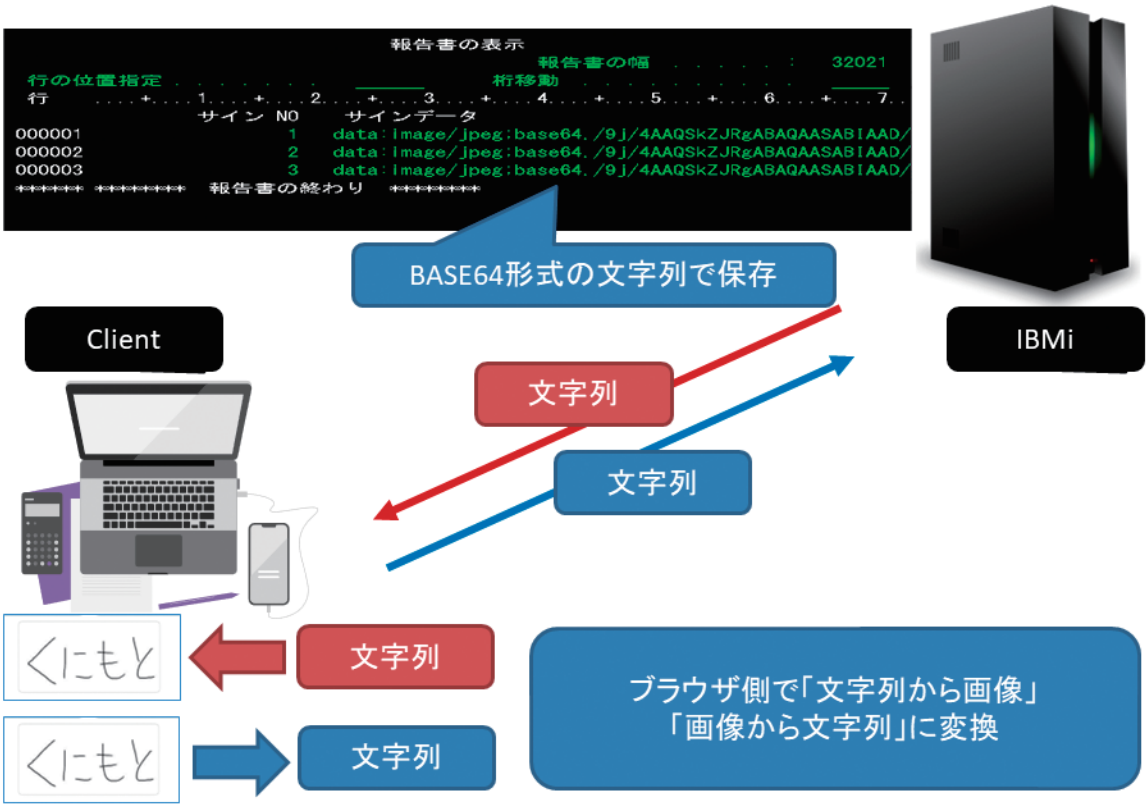
本節では、SmartPad4iで電子サイン機能を実装する方法について説明する。電子サインを実装したアプリケーションの例は**【図3】**、**【図4】**だ。

図3　電子サインの実装（1）

図4　電子サインの実装（2）

例では、電子サインを記入するcanvas要素と操作用の「クリア」ボタン、「登録」ボタンを配置した。
画面が読み込まれると、canvas要素に描画するためのイベント処理が追加されて、canvas要素に指やタブレットペンでサインが記入できる状態になる。サイン記入後「登録」ボタンをクリックすると、記入した電子サインの情報が文字列として、IBMi側に送信され、IBMiのファイルに電子サインが文字列として登録される。**【図5】**

図5　電子サインの実装(3)



また、ファイルに登録した最後のサインを画面に表示できるように実装した。

3-2. フロントエンド実装

3-2-1. HTMLの作成

では、フロントエンドのHTMLとJavaScriptの実装方法から説明する。HTMLの例は**【ソース4】**、**【ソース5】**だ。

ソース 4

HTML 電子サイン 例(1)

```
<form method="post" >
  <div id="container">
    <div id="content">
      <header>
        <div>
          <div>
            <span><img src="img/migaro_logo.png"></span>
          </div>
        </div>
      </header>
      <main>
        【ソース5】
      </main>
    </div>
  </div>
</form>
```

【ソース4】では、画面上部ヘッダーに表示するアイコンの画像を定義している。mainタグの中は**【ソース5】**になる。

ソース 5

HTML 電子サイン 例(2)

```
<div class="box" >
  <section style="border-bottom:solid 2px #3e8ed0">
    <h1>署名</h1>
  </section>
  <canvas id="signCanvas" width="600px" height="300px">
    <p> お使いのブラウザでは電子サインができません</p>
  </canvas>
</div>
<div style="width:100%"></div>
<div>
  <div>
    <input type="button" class="button" style="margin-right: 10px;"
           data-clear="true" value="クリア">
    <input type="button" class="button" value="登録" data-save="true">
    <input type="button" style="display:none" value="登録" id="BTN01">
    <input type="hidden" id="SIGN">
  </div>
</div>
<img src="" data-img="true">
```

①canvas要素
横幅:600 縦幅:300

②操作用ボタン

③隠しボタン
隠し入力欄

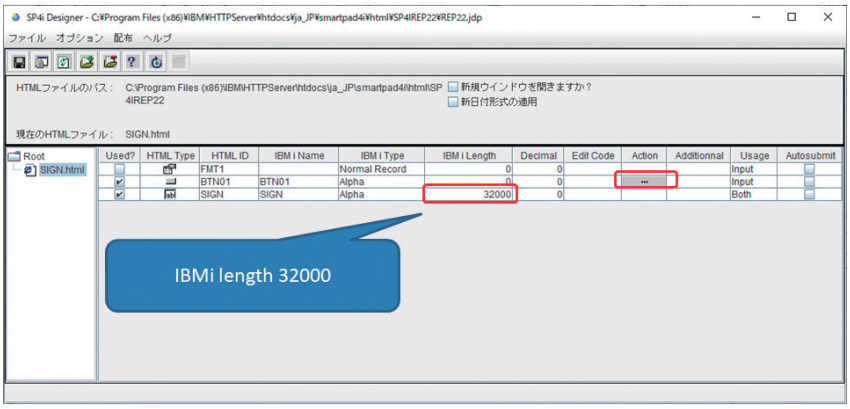
④登録済み電子サイン表示用画像

【ソース5】①でcanvas要素を横幅600縦幅300で定義した。**【ソース5】**②では、操作用ボタンを定義している。canvas要素の描画内容を消去するための「クリア」ボタン、canvas要素の描画内容を登録するための「登録」ボタンだ。クリックされた際に、canvas要素の電子サインを文字列に変換する処理を行い、隠し入力欄に電子サインの文字列を設定後、IBMi側へデータを送信するため、隠しボタンをクリックする。**【ソース5】**③はIBMiにデータを送信するための隠しボタンと隠し入力欄だ。隠し入力欄に文字列データを設定、隠しボタンをクリックすることでIBMi側へ送信している。**【ソース5】**④はIBMi側ファイルに登録された電子サインを表示するための画像を定義した。

3-2-2. Designerの設定

SmartPad4iの開発では、作成したHTMLをDesignerで読み込み、各フィールドのデータ型やデータ長を設定後、IBMiへ配布する。電子サインのデータは、画像データを文字列に変換して入出力する。そのため、入出力する入力欄のIBMi Lengthには、最大長の32,768byteに近い値を設定する。サンプルでは、32,000byteを設定した。**【図6】**

図 6　Designer設定



3-2-3. JavaScriptの作成

次に、canvas要素へ電子サインを描画するためのJavaScriptを説明する。SmartPad4iのinitpage関数実行時にcanvas要素へ電子サインを描画するための処理を記述している**【ソース6】**。

SmartPad4iのinitpage関数は画面が読み込まれて、SmartPad4iの初期処理が呼び出される前に実行される関数だ。

ソース 6

JavaScript 電子サインcanvasへの描画(1)

```
<script>
  function initpage(){
  ～ 省略 ～
    //描画コンテキストの取得
    var canvas = document.getElementById('signCanvas');
    var context = canvas.getContext("2d");
    //線形, 太さ, 色 ,キャンバスクリア
    context.lineCap  = "round";
    context.lineWidth = 3;
    context.strokeStyle = "black";
    clearCanvas();
    var isDragging = false;
    var lastPosition = { x: null, y: null };
    //PC
    canvas.addEventListener("mousedown", dragStart);
    canvas.addEventListener("mouseup",   dragEnd);
    canvas.addEventListener("mouseout",  dragEnd);
    canvas.addEventListener("mousemove", dragMovePC);
    //SmartDevice
    canvas.addEventListener("touchstart",  dragStart);
    canvas.addEventListener("touchcancel", dragEnd);
    canvas.addEventListener("touchend",    dragEnd);
    canvas.addEventListener("touchmove",   dragMoveSD);
  }
</script>
```

①コンテキスト取得
②描画設定と初期化
③制御用
④PC用イベント
⑤Smartデバイス用イベント

【ソース6】は電子サインを描画するための準備処理に当たる箇所だ。initpage関数の前半に「～省略～」と記載した箇所には、**【ソース6】**の中で実行される関数が定義されている。

関数は**【ソース7】**、**【ソース8】**、**【ソース9】**になる。

ソース 7

JavaScript 電子サインcanvasへの描画(2)

```
//canvasクリア
function clearCanvas(){
  context.clearRect(0, 0, canvas.width, canvas.height); //canvasクリア
  context.fillStyle = "#FFFFFF"; //塗りを白色に設定
  context.fillRect(0,0,canvas.width,canvas.height); //四角を描く
}

//描画の開始
function dragStart(event) {
  context.beginPath();
  isDragging = true;
}

//描画の終了
function dragEnd(event) {
  isDragging = false;
  //座標リセット
  lastPosition.x = null;
  lastPosition.y = null;
}
```

①canvas初期化

②描画の開始

③描画の終了

ソース 8

JavaScript 電子サインcanvasへの描画(3)

```
//描画呼び出し(PC)
function dragMovePC(event) {
  draw(
    event.layerX - canvas.getBoundingClientRect().left - window.scrollX ,
    event.layerY - canvas.getBoundingClientRect().top - window.scrollY
  );
}

//描画呼び出し(SD)
function dragMoveSD(event) {
  event.preventDefault();
  let x = event.touches[0].clientX - canvas.getBoundingClientRect().left
    - document.body.scrollLeft;
  let y = event.touches[0].clientY - canvas.getBoundingClientRect().top
    - document.body.scrollTop;
    draw(x, y);
  }
```

①PC用描画処理

②Smartデバイス用描画処理

ソース 9

JavaScript 電子サインcanvasへの描画(4)

```
//描画処理
function draw(x, y) {
  if (!isDragging) {
    return;
  }
  if (lastPosition.x === null || lastPosition.y === null) {
    context.moveTo(x, y);
  } else {
    context.moveTo(lastPosition.x, lastPosition.y);
  }
  context.lineTo(x, y);
  context.stroke();
  // 座標保持
  lastPosition.x = x;
  lastPosition.y = y;
}
```

①ドラッグ処理

②開始点

③終了点

④座標保持

【ソース6】①で描画コンテキストを取得し、**【ソース6】**②で線の描画設定をしている。また、**【ソース7】**①のclearCanvas関数を呼び出し、canvas要素をクリアする。clearCanvas関数では、描画コンテキストのclearRectメソッドを呼び出し、canvas要素をクリアしている。

clearRectメソッドは矩形の形状で描画内容を消去する処理だ。引数はクリアする開始点x,y座標とクリアする横幅、縦幅の4つを指定する。

また、消去後にfillStyleへ白を設定、fillRectでcanvas要素全体に背景用の白い四角形を描画することでcanvas要素をクリアしている。

【ソース6】③は制御用のオブジェクトで、isDraggingはドラッグ処理中のフラグ、lastPositionは描画時の最終x,y座標位置を保持している。

【ソース6】④と⑤はマウス操作や、タッチ操作時の処理になる。マウス操作のPC用と、タッチ操作のSmartデバイス用を定義した。

addEventListenerは要素にイベントを追加する際に使用するメソッドだ。**【ソース6】**④では、canvas要素上のマウスダウン（mousedown）イベントで**【ソース7】**②で定義しているdragStart関数を呼び出す。

dragStart関数は描画の開始処理になり、描画コンテキストのbeginPathメソッドを呼び出し、現在描画中のパスをクリア後、制御用変数のisDraggingフラグにtrueを設定する。

【ソース6】④では、マウスのボタンが離された際のマウスアップ（mouseup）イベント、マウスが要素上から外れた場合に発生するマウスアウト（mouseout）イベントで、**【ソース7】**③で定義しているdragEnd関数を呼び出す。

dragEnd関数では、isDraggingフラグをfalseに設定することで、ドラッグ操作が終了したことを設定している。また、lastPositionオブジェクトのx,y座標にnullを設定して初期化する。

【ソース6】④のマウスが移動している際に発生するマウスムーブ（mousemove）イベントでは、**【ソース8】**①のdragMovePC関数を呼び出している。

dragMovePC関数では、**【ソース9】**の実際に描画処理を実施している、draw関数を呼び出す際の座標を取得する。eventオブジェクトに格納されているlayerXからマウスのx座標、layerYからマウスのy座標を取得、マウス位置からcanvas要素のgetBoundingClientRectメソッドでcanvas要素のx,y座標を取得して、それぞれ減算することで、canvas要素上の描画位置を算出している。window.scrollX,scrollYは画面がスクロールしている場合の計算処理だ、Smartデバイスでは、document.body.scrollLeftと document.body.scrollTopになる。

【ソース6】⑤は**【ソース6】**④のSmartデバイス向けのタッチ操作イベントにそれぞれ関数を設定している。PCブラウザと座標位置の取得が異なるため、タッチ操作（touchMove）イベント時には、**【ソース8】**②のdragMoveSD関数を呼び出している。PCブラウザとの違いは、eventオブジェクトのpreventDefaultを呼び出して、イベントの伝搬を終了している点だ。この処理により、電子サイン記入時には、タブレットや、スマートフォンの画面が上下左右に動かず、canvas要素上でサインができる。

最後に**【ソース9】**のdraw関数を呼び出してcanvas要素に描画を行う。**【ソース9】**のdraw関数では、**【ソース9】**①でドラッグ操作中であるかを判断して、ドラッグ操作が終了している場合にはreturnで関数処理を終了している。

ドラッグ操作時には**【ソース9】**②で線を描画する開始点を指定する。

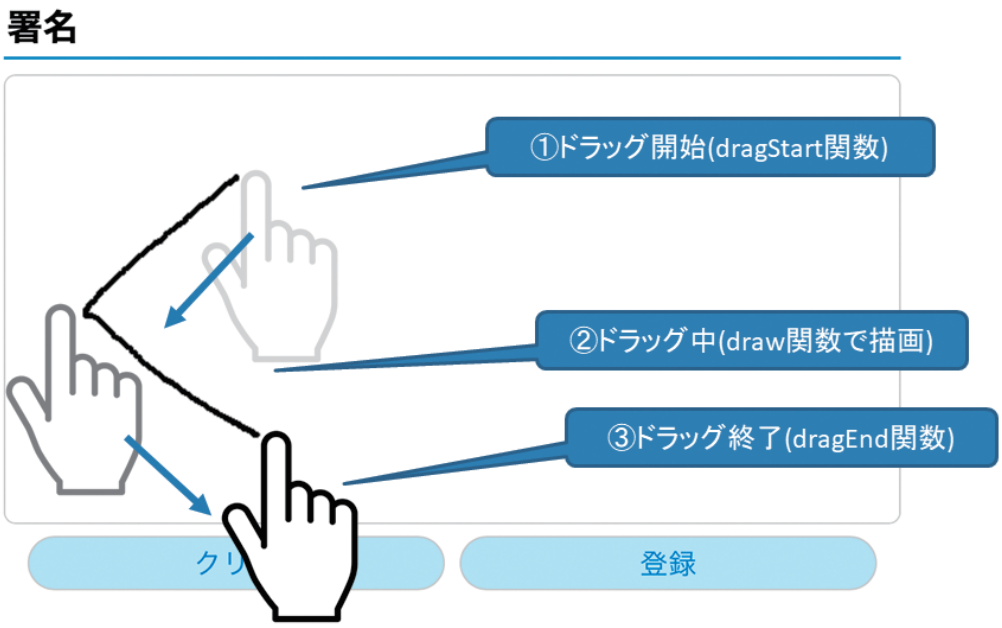
【ソース9】③では、線の描画の終了点を設定後、描画コンテキストのstrokeメソッドを呼び出して、canvas要素に描画を行う。最後に、**【ソース9】**④で最後の座標位置を保存している。

要するに、タッチ操作/マウスダウン操作時に描画開始の処理が実行されて、canvas要素上をタッチ操作/マウス操作で移動中に常にdraw関数で描画を繰り返している。

タッチ操作の終了（指やタッチペンをcanvas要素から離す）/マウス操作の終了（マウスボタンを離す）時に、描画処理が終了することになる。**【図7】**

図 7　電子サインの実装(4)

株式会社ミガロ.



canvas要素へ描画した内容を、クリアボタンでクリアする処理と、登録ボタンでIBMi側プログラムへ送信する処理は**【ソース10】**だ。

ソース 10

JavaScript 電子サインcanvasへの描画(5)

```
//クリアボタン処理
var clearButton = document.querySelector('[data-clear]');
clearButton.onclick = function(){
  clearCanvas();
}

//登録ボタン処理
var saveButton = document.querySelector('[data-save]');
saveButton.onclick = function(){
  var RESIZE_RATIO = 0.5;
  var JPG_QUORITY = 0.5;
  var resizeCanvas = document.createElement('canvas');
  var rCtx = resizeCanvas.getContext('2d');
  //サイズを調整 1で同サイズ/ 0.5で半分
  rCtx.width = canvas.width * RESIZE_RATIO;
  rCtx.height = canvas.height * RESIZE_RATIO;
  rCtx.drawImage(canvas, 0, 0, canvas.width, canvas.height,
      0, 0, rCtx.width, rCtx.height);
  //レコードデータ更新
  var sing   = SP4i.getElementById('SIGN');
  sing.value = resizeCanvas.toDataURL('image/jpeg' ,JPG_QUORITY);
  var btn01  = SP4i.getElementById('BTN01');
  btn01.click();
}
```

①クリアボタン

②リサイズ用コンテキスト

③半分のサイズに変換

④データ変換と送信処理

【ソース10】①「クリア」ボタンの要素取得には、カスタムデータ属性を使用している。カスタムデータ属性は、data-*の形式で、*の箇所に独自で命名した名前を指定して属性を追加できる便利な機能だ。document.querySelectorメソッドでカスタムデータ属性にdata-clearが設定されている要素を取得している。

document.querySelectorメソッドはCSS セレクターを引数に設定することにより、条件に一致する最初の要素を取得することのできるメソッドだ。

取得したボタンは、clearButton変数に格納され、クリックイベント処理時に、**【ソース7】**①のclearCanvas関数を呼び出すことで、canvas要素をクリアしている。

【ソース10】「登録」ボタンの処理では、document.querySelectorメソッドでカスタムデータ属性にdata-saveが設定されている要素を取得している。要素は、saveButton変数に格納後、クリック時の処理を追加している。

「登録」ボタンのクリック時には、**【ソース10】**②の処理でリサイズ用の描画コンテキストを作成する。

リサイズ用の描画コンテキストの横幅、縦幅を半分に設定後、リサイズ用描画コンテキストのdrawImageメソッドで電子サインの描画内容をリサイズ用のコンテキストへコピーしている。

drawImageメソッドは、第1引数に描画するイメージを設定する。img要素、canvas要素、video要素を指定可能だ。第2引数から、第5引数には第1引数に設定された要素（コピー元）の使用範囲を指定する。第2引数は、開始x座標、第3引数は開始y座標、第4引数は横幅、第5引数は縦幅になる。第6引数から、第9引数は描画するイメージを配置する座標を指定する。

つまり、**【ソース10】**③の処理でリサイズ用の描画コンテキストに、半分のサイズに圧縮した内容をコピーしている。

【ソース10】④の処理では、IBMiへデータを送信するために、描画した電子サインの情報をtoDataURLメソッドで文字列に変換する。jpegデータ形式のデータをbase64エンコードした文字列に変換して、id属性SIGNが設定された入力欄に格納後、id属性"BTN01"の隠しボタン要素をクリックすることでIBMi側にデータを送信している。

最後に、登録した電子サインを表示するためのJavaScriptは**【ソース11】**だ。

ソース 11

JavaScript BASE64エンコードされた電子サインを画像として表示

```
//登録電子サイン表示
var imgsrc  = SP4i.getElementById('SIGN').value;
var img = document.querySelector('[data-img]');
if(imgsrc){
  img.src = imgsrc;
  img.style.display = 'block';
}
```

【ソース11】は、initpage関数内の最後に追加する。SP4i.getElementByIdメソッドを使用して、id属性"SIGN"の入力欄に設定された値をimgsrc変数に格納している。

次に、document.querySelectorメソッドでデータ属性"data-img"の要素をimg変数に格納する。imgsrc変数に値が設定されている場合、つまり、バックエンド側から電子サインの文字列データが送信されてきている場合には、img変数の画像ソースに電子サインの文字列を設定して、cssのdisplayプロパティに"block"を設定することで画面に表示している。以上が、フロントエンドの処理になる。

3-3. バックエンド実装

フロントエンドから送信された、電子サインの情報は文字列データとしてIBMiプログラムに送信される。
本稿のサンプルでは、JPEG画像データをBASE64エンコードした。
電子サインを保存するファイルのDDSは**【ソース12】**になる。

ソース 12

IBMi DDS 電子サインファイル

```
0001.00      A*********************************************************************
0002.00      A* ID          : F_SIGN
0003.00      A* NAME        : 電子サインファイル
0004.00      A* CREATE      : 2022/09/06 Y.KUNIMOTO
0005.00      A* UPDATE      : 9999/99/99
0006.00      A*********************************************************************
0007.00      A*
0008.00      A                                      UNIQUE
0009.00      A           R F_SIGNR                  TEXT('電子サイン')
0010.00      A             SIGNSNO       14S 0      COLHDG('サインNO')
0011.00      A             SIGNDTA    32000A        COLHDG('サインデータ')
0012.00      A           K SIGNSNO
```

電子サインを保存するフィールドは32,000桁で定義している。
バックエンド側では、送信されてきた電子サインの文字列データをファイルに入出力するだけだ。RPGプログラムは**【ソース13】**になる。

ソース 13

IBMi RPGプログラム（F仕様書）

```
0011.00  *-------------------------------------------------------------
0012.00  * DATABASE FILE DEFINITION
0013.00  *-------------------------------------------------------------
0014.00  * <YOURCODE>
0015.00  * YOUR FILES
0016.00 FF_SIGN    IF A E           K DISK      USROPN
0017.00  * </YOURCODE>
```

IBMi RPGプログラム（書込み呼び出し）

```
0114.00  * <YOURCODE>
0115.00  * CHECK ACTION CODE SPACTN HERE AND PROCESS.
0116.00 C     SPACTN        IFEQ      'A1'
0117.00 C                   EXSR      SB0020
0118.00 C                   ENDIF
0119.00 C     SPACTN        IFEQ      'F3'
0120.00 C                   GOTO      ENDPGM
0121.00 C                   ENDIF
0122.00  * YOUR CODE
0123.00  * </YOURCODE>
```

IBMi RPGプログラム（読込み呼び出し）

```
0352.00 C     YRDATA        BEGSR
0353.00  *
0354.00 C                   EXSR      SB0010
```

ソース 13

IBMi RPGプログラム（読込み）

```
0386.00  *----------------------------------------------------------------
0387.00  * SB0010:電子サインの読込み
0388.00  *----------------------------------------------------------------
0389.00 C     SB0010        BEGSR
0390.00 C                   OPEN      F_SIGN
0391.00 C     *HIVAL        SETGT     F_SIGNR
0392.00 C                   READP     F_SIGNR                              70
0393.00 C     *IN70         IFEQ      *OFF
0394.00 C                   MOVEL(P)  SIGNDTA       OSIGN
0395.00 C                   ENDIF
0396.00 C                   CLOSE     F_SIGN
0397.00 C                   ENDSR
```

IBMi RPGプログラム（書込み）

```
0398.00  *----------------------------------------------------------------
0399.00  * SB0020:電子サインの書込み
0400.00  *----------------------------------------------------------------
0401.00 C     SB0020        BEGSR
0402.00 C                   OPEN      F_SIGN
0403.00 C     *HIVAL        SETGT     F_SIGNR
0404.00 C                   READP     F_SIGNR                              70
0405.00 C     *IN70         IFEQ      *OFF
0406.00 C* サイン番号取得
0407.00 C     SIGNSNO       ADD       1             SIGNSNO
0408.00 C                   ELSE
0409.00 C                   Z-ADD     1             SIGNSNO
0410.00 C                   ENDIF
0411.00 C* 電子サイン書込み
0412.00 C                   MOVEL(P)  ISIGN         SIGNDTA
0413.00 C                   WRITE     F_SIGNR
0414.00 C                   CLOSE     F_SIGN
0415.00 C                   ENDSR
```

【ソース13】では、F仕様書ではファイルを読書き可能として定義している。【ソース13】（書込み呼び出し）ではフロントエンド側からアクションコード"A1"が送信された場合、つまり、登録ボタンが押下された場合に、サブルーチンSB0020を実行してファイルに書込んでいる。

また、読込みは、アプリケーション起動時にSmartPad4iのYRDATAでサブルーチンSB0010を呼び出し読込んでいる。 RPGプログラムは本稿用サンプルのため、バックエンド側のプログラムは使用する業務にあわせて作成してほしい。

4.実装時の注意点

電子サイン情報は文字列データとしてIBMi側で扱う、文字列データは大文字、小文字を区別するため注意が必要だ。また、電子サインのサイズやデータ形式によっては大容量の文字列データとなる場合がある。

SmartPad4iでは、一つのフィールドでデータを送受信することのできる最大長は32,768byteのため、大容量のデータを入出力する場合には、フォーマットを分けて、複数のフィールドを使用して送受信する必要がある。

また、IBMi側ファイルで保存できる文字列の最大長も32,768byteのため、特定のbyte長でデータを区切り、複数のレコードに情報を保存する処理などが必要となるため注意してほしい。

5.おわりに

本稿では、SmartPad4iで電子サインの機能を実装する方法についてまとめた。実装の中でSmartPad4i/Cobos4iでは、HTMLを使用して、自由にインターフェースを作成できることを再認識できた。

電子サインの機能をSmartPad4iアプリケーションで実装することで、業務効率化やコスト削減、ペーパーレス化を進めることの手助けになれば幸いだ。

Valence

Valence モバイルアプリケーション開発テクニック

株式会社ミガロ.
プロダクト事業部技術支援課
尾崎　浩司

略　歴
生年月日:1973年8月16日
最終学歴:1996年　三重大学 工学部卒業
入社年月:1999年10月　株式会社ミガロ. 入社
社内経歴:
1999年10月　システム事業部配属
2013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:
ミガロ. が取り扱う3つの開発ツールのセミナー講師や技術支援を主に担当している。

1.はじめに

Valenceは、IBM i(AS/400)を使用したモダナイゼーションツールである。Valenceは登場当時、Senchaと呼ばれるJavaScriptをベースとしたUIフレームワークを使用して画面を設計し、ビジネスロジックをRPGでコーディングする事でアプリケーションを開発するものであった。当時よりモダンなWebアプリケーションが作成できる環境として好評であったが、開発にはSenchaのスキルが必須で、若干敷居が高かったのも事実である。そこで、より簡単にアプリケーション開発が行えるようにと、2018年に登場したValence 5.2からは、「Valence App Builder」が追加で搭載され、新たにローコード開発機能がサポートされるようになった。Valence App Builderの登場により、Valenceを使用するアプリケーション開発生産性は大幅に向上し、現在ではこれが、Valenceの主力機能になっている。

これまでにも、本テクニカルレポートにて、Valence App Builderの開発手法を紹介してきたが、それらは、主にPCブラウザ(デスクトップ)向けアプリケーションのトピックが中心であった。Valenceは、PCブラウザ向けアプリケーションだけでなく、もちろんモバイルアプリケーションの開発も可能である。そこで、今回は、Valence App Builderを使用したモバイルアプリケーション開発に焦点を絞って、その開発テクニックを紹介する。

2.Valence Mobileアプリの使用方法

Valenceにおいて、スマートフォンやタブレット等のモバイルデバイスを使用する場合、ブラウザではなく、専用のアプリから実行する。Valence5.2以前では、「Valence Portal」という専用アプリを使用していたが、現在はValence 6.0以降に対応した新しい「Valence Mobile」アプリに変更されている。iOSやAndroidの公式アプリストアの中にValence Mobileアプリが登録されているので、始めにアプリをデバイスにダウンロードし、インストールすればよい。**【図1】**

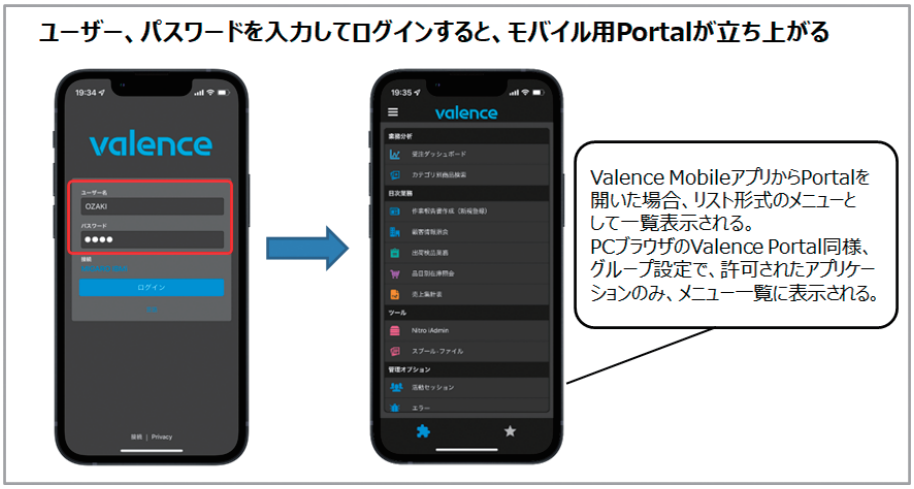
図 1 Valence Mobileアプリ

アプリをインストールしたら、ホーム画面から「Valence Mobile」アイコンをタップしてアプリを起動する。初回起動時に初期設定として、IBM iへの接続先登録画面が表示されるので、「接続先名」と「URL」を登録する。「URL」は、PCブラウザで、Valence Portalにアクセスする場合と同様に、IBM iのIPアドレスおよびValenceを実行するインスタンスに相当するポート番号をURL形式で指定すればよい。**【図2】**

図 2 Valence Mobileアプリ 初期設定

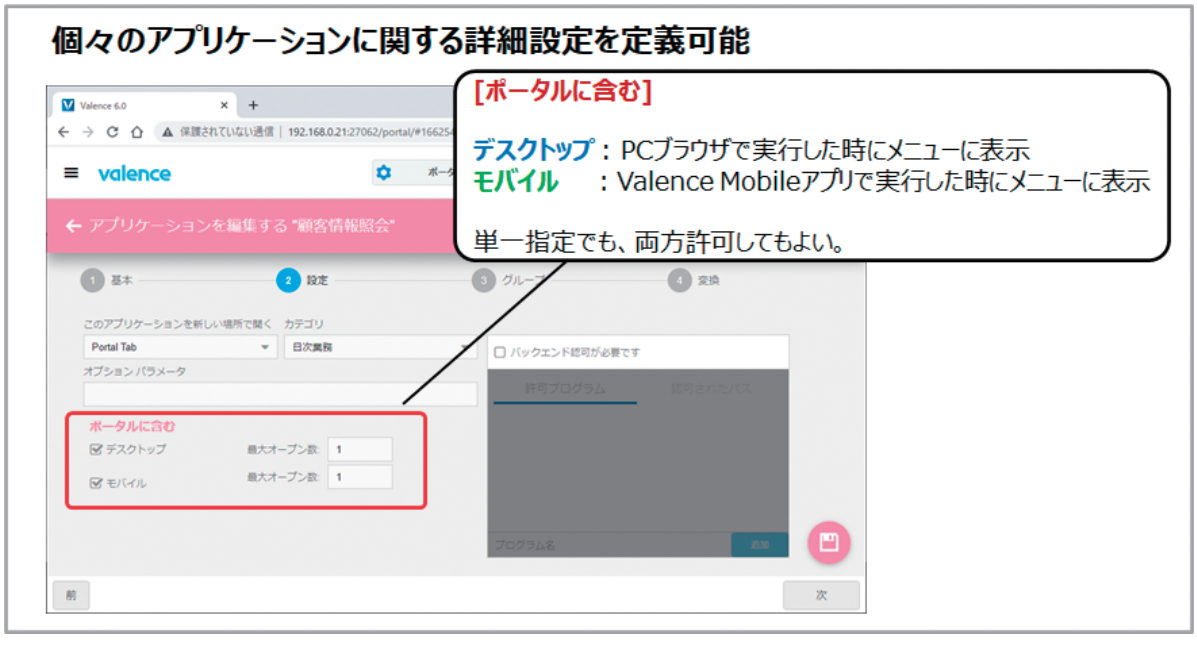
初期設定が完了すると、Valence Portalへのログイン画面が表示されるので、PCブラウザの場合と同様に、有効なユーザーとパスワードを入力してValenceにログインすると、モバイル用のValence Portalが立ち上がる。PCブラウザの場合は、タイル形式のアイコンメニューとして開くが、Valence Mobileアプリの場合は、リスト形式のメニューとして表示される。**【図3】**

図 3　Valence Mobileアプリ Portalへのログイン



もちろん、[ポータル管理]→[グループ]で設定した権限設定が有効な為、許可されたアプリケーションのみがメニューにリスト表示される。さらに[ポータル管理]→[アプリ]の中で、個別のアプリケーション設定を開き、設定画面の中で、[デスクトップ]と[モバイル]を個別にValence Portalに含めるかどうか指定できるようになっている。つまりValence Mobileアプリから利用した時のみ、実行を許可するという設定も可能である。**【図4】**

図 4　[ポータル管理]→[アプリ]設定画面



Valence MobileアプリからValence Portalを開けば、後はPCブラウザ同様に実行したいアプリケーションをメニュー一覧からタップすればよい。

なお、Valence MobileアプリからValenceを利用する場合、Wi-Fiや4G/5G通信回線からのアクセスになる点には注意してほしい。社内ネットワーク内のWi-Fiを使用するのであれば、そのままIBM iに接続できるが、外部から通信回線を使用したアクセスを想定する場合は、予めVPNを設定して、社内ネットワークに事前にログインしておく必要がある。

ちなみにVPNを使用せずに直接インターネット経由でValence Mobileアプリを使用したい場合は、別途SSL環境を構築して運用する方法もあるが、本稿では取り上げない。

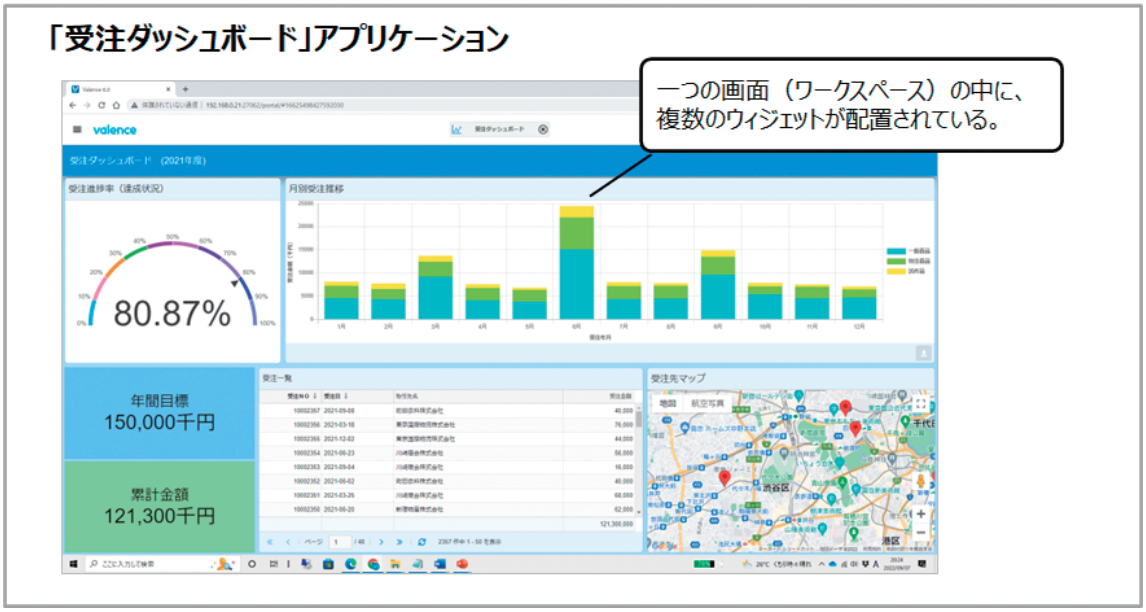
興味のある方は、Valence開発元CNX社のblog記事(https://www.cnxcorp.com/blog/setting-valence-ibm-i-external-access-ssl)等を参考にしてほしい。

3.Valence モバイルアプリケーション作成の基本

本節よりValence App Builderを使用した具体的なモバイルアプリケーション作成方法を紹介する。

まず、PCブラウザ(デスクトップ)向けアプリケーションをそのままモバイルで実行するとどうなるかを確認する。ここでは例として、「受注ダッシュボード」アプリケーションを使用する。**【図5】**

図 5　PCブラウザ(デスクトップ)アプリケーション例



このアプリケーションは、一つの画面（ワークスペース）にChartやGrid、Map、Single KPIといった複数のウィジェットを合計6個配置して、複数の要素を一括表示するものである。

同じアプリケーションをValence Mobileアプリから、スマートフォンで実行する。その実行結果が、**【図6】**である。

図 6　PCブラウザ向けアプリケーションをそのまま実行

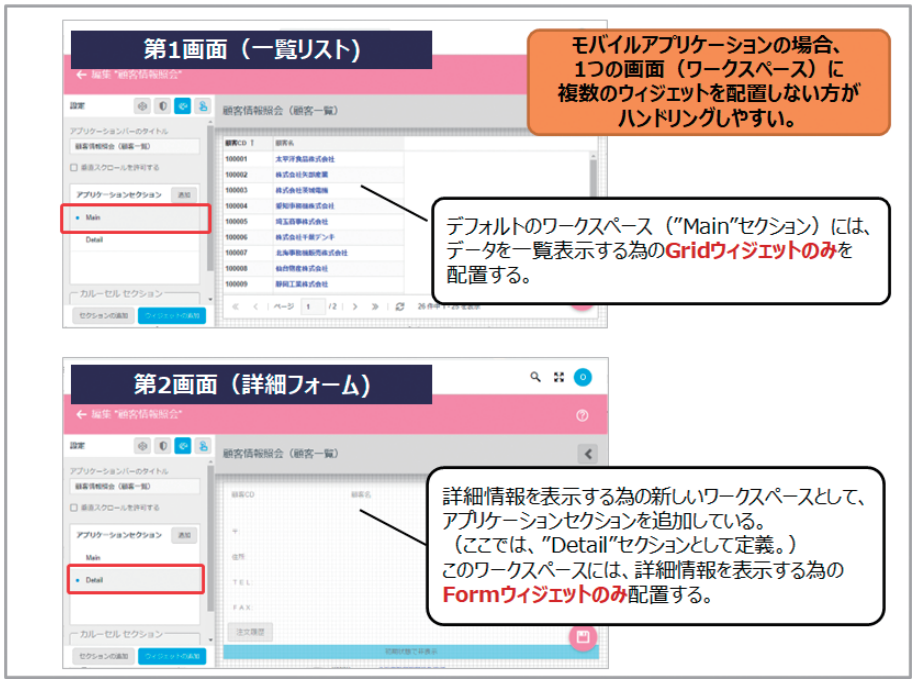
Valence Mobileアプリから実行すると、ワークスペース内の各ウィジェットが自動的に分割され、ウィジェット単位で画面表示される。そして、各ウィジェットはスワイプ操作や画面下部の点（インジケーターアイコン）のタップ操作で画面切り替えを行う形になる。このようにValenceでは、特に意識しなくても、自動的にモバイル用に画面が最適化されるのである。

Valenceでは、実行環境に合わせて表示方法を自動的に変更するので便利である。しかし、画面は自動分割となる為、PCでの画面のレイアウトだけを意識して作成したものを、そのままモバイルデバイスから実行すると意図しないレイアウトや挙動になる可能性もある。

つまり、モバイル向けアプリケーションとして作成する場合、意図的にモバイルでの利用を前提として開発を行った方が、想定通りの動作にできる。

具体的には、モバイルの場合、一つの画面には、同時に一つのウィジェットだけが表示される仕様となる点を意識すればよい。例として、Gridウィジェットに一覧出力されたデータの中から、行を選択し、詳細情報をFormウィジェットで確認するといった動きのアプリケーションを考えてみる。この場合、App Builderのアプリケーション作成ステップにおいて、2つのアプリケーションセクション（ワークスペース）を設定し、1つ目のワークスペースには一覧リストを表示するGridウィジェットを、2つ目のワークスペースには詳細情報を表示するFormウィジェットを配置する構成にすればよい。**【図7】**

図 7　モバイルに適したアプリケーション設定



そして、アプリケーションの[動作内容]で、次のような設定を行う。

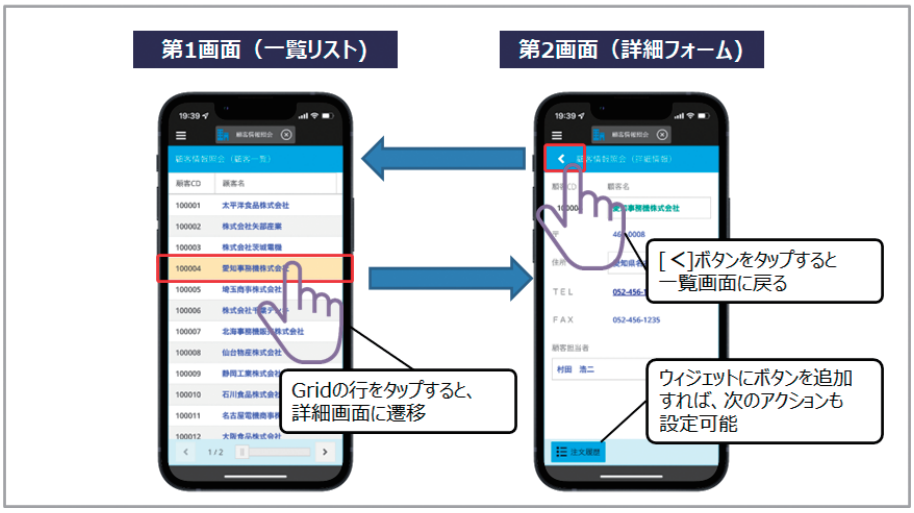
1つ目のワークスペースにあるGridウィジェットの行クリック時に、Formウィジェットに対する[フィルタアクション]と[ウィジェットの表示/非表示]のアクションを設定する。

2つ目のワークスペース上には、前画面に[戻る]ボタン(ここでは、スマートフォンで分かり易いよう[<]マークのアイコンを設定)を追加する。追加したボタンのクリック時に、[ウィジェットの表示/非表示]アクションを設定し、1つ目のワークスペース上のGridウィジェットを再表示させる。**【図8】**

図 8　アプリケーション設定[動作内容]画面

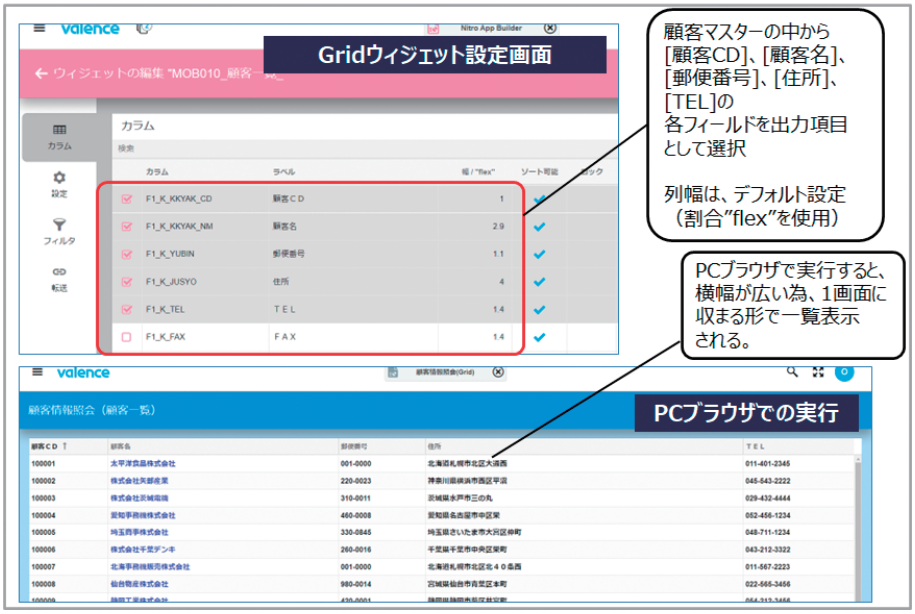
このように、明示的に必要な画面の遷移を[動作内容]に定義することで、意図しない画面遷移を防ぐことができ、モバイルに最適化した画面遷移とする事が可能である。**【図9】**

図 9　モバイルアプリケーション実行例



次にウィジェット作成におけるモバイルでの考慮点を説明する。Valenceのウィジェットの中で一番多用するのが、データを一覧形式で出力するGridウィジェットだろう。例えば、**【図10】**は、顧客情報を一覧表示するGridウィジェットの設定例と、それをPCブラウザで実行した場合の実行画面である。

図 10　Gridウィジェット設定とPCブラウザでの実行例



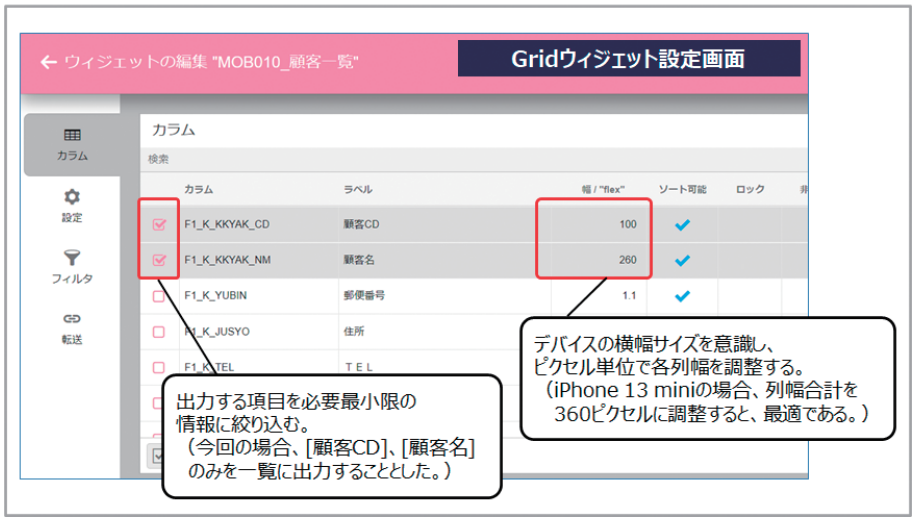
同じアプリケーションをValence Mobileアプリから実行した場合の画面が、【図11】である。同じ内容をもちろん表示できるのだが、特に横方向の画面幅の狭いスマートフォンでは、Grid内の表部分をスワイプ操作で横スクロールしながら閲覧する必要がある。

図11 スマートフォンでの実行例

片手操作が基本となるスマートフォンにおいて、横方向のスワイプによるGridの横スクロールはお世辞にも使い勝手が良いとは言えない。では、Gridの画面設計をどうすればよいかであるが、これは各カラムの列幅をスマートフォンの画面サイズにあわせて、明示的にピクセル値を設定して調整すればよい。

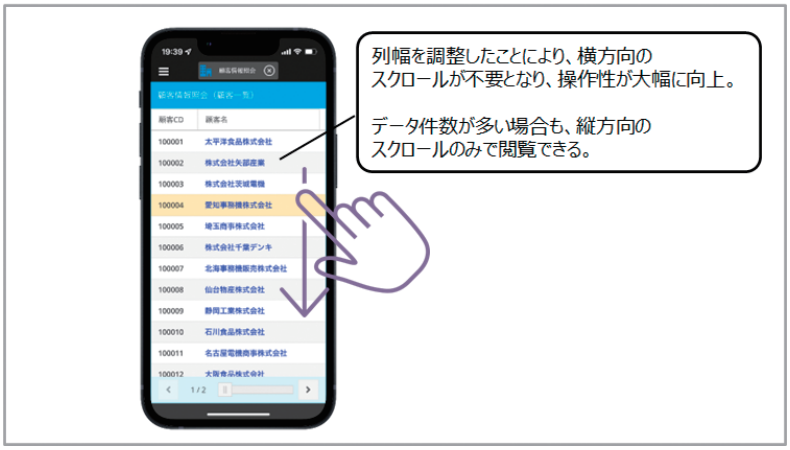
筆者が使用しているiPhone13miniをターゲットデバイスとした場合では、各カラムの列幅の合計値を360ピクセル程度に収めると、横スクロール無しに、データを表示させる事が可能である。ここでは、出力する項目を2項目に絞り込み、実際に出力されるデータ文字長から勘案して、[顧客CD]を100ピクセルに、[顧客名]を260ピクセルに設定した。【図12】

図12 スマートフォンに最適化したGridウィジェット設定



スマートフォン用に調整した顧客一覧画面では、横方向のスクロールは無くなり、縦方向のみのスクロールでデータが参照できる。**【図13】**

図13　列幅調整後のスマートフォンでの実行例



モバイルの場合、デバイスにより画面のサイズや解像度が異なる為、開発の際には、ターゲットデバイスによる実機テストを行う事を推奨するが、一般的なiOSやAndroidのスマートフォンであれば、横幅は、合計で300～400ピクセル程度になるように設定すればよいだろう。

Gridウィジェットの次に使用頻度が高いのが、詳細情報を表示するのに最適なFormウィジェットだろう。Formウィジェットも、基本的な考え方は、同様だが、原則としては、一つの項目を1行に配置し、横方向に複数項目を配置しない事である。次の図は、顧客に関する詳細情報を表示するFormウィジェットの設定例である。**【図14】**

図14　スマートフォンに最適化したFromウィジェット設定

Formウィジェットの場合、横幅をピクセル設定しなくても、1行に1項目であれば、Flex（割合）=1としておけば、デバイスにあわせて横幅が調整される。1行に2項目の場合でも、例えば、1:2のような割合で指定すればよい。

【図14】のFormウィジェットの設定をValence Mobileアプリから実行した画面が【図15】である。スマートフォン上で違和感のない配置になっている事がわかる。

図15 Fromウィジェットのスマートフォンでの実行例

本節では、主にスマートフォンからValence Mobileアプリを実行する事を前提に作成ポイントを紹介した。スマートフォンの場合、デバイスを縦向きで使用する事が前提となるが、タブレットの場合、横向きでの使用が可能な場合もある。例えば、本節冒頭で例にあげた「受注ダッシュボード」アプリケーションは、iPadであれば横向きで実行する事により、PCブラウザの場合と同様に複数ウィジェットが同時表示される事がわかる。【図16】

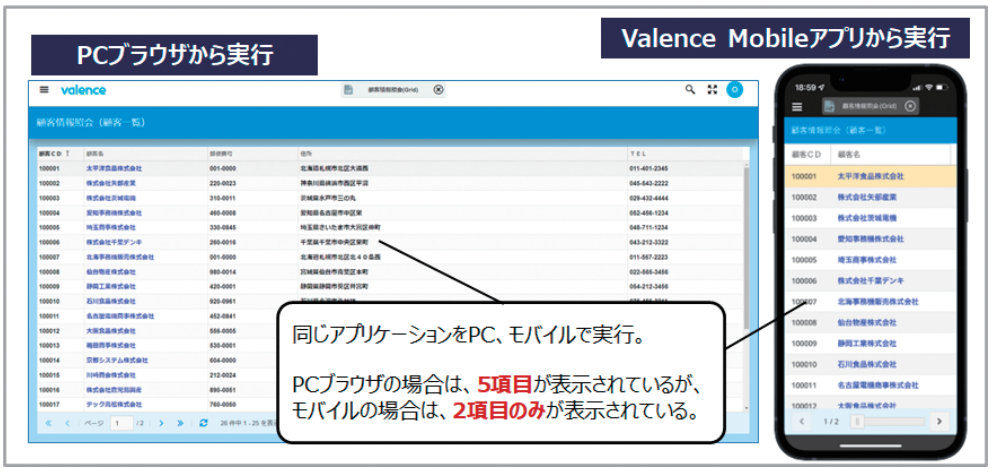
図16 タブレット（iPad）横向きでの利用

4.デスクトップとモバイルの共用テクニック

前節では、モバイルに特化したアプリケーションを作成する場合の手法を紹介した。モバイル専用のアプリケーションを作成する場合は問題ないが、一つの同じアプリケーションをPCブラウザとモバイルとで共用しながらも、実行環境にあわせて画面を自動的に最適化したい場合もあるだろう。本節では、アプリケーションをPCブラウザ(デスクトップ)とモバイルで共用するテクニックを紹介する。

ここでは、前節での例でも使用した顧客情報を一覧リスト出力するGridウィジェットを用いたアプリケーションを使用する。**【図17】**が完成したアプリケーションの実行例である。同じアプリケーションであるにも関わらず、PCブラウザで実行した時と、Valence Mobileアプリから実行した時とで、Gridウィジェットの出力カラムが異なる事がわかる。

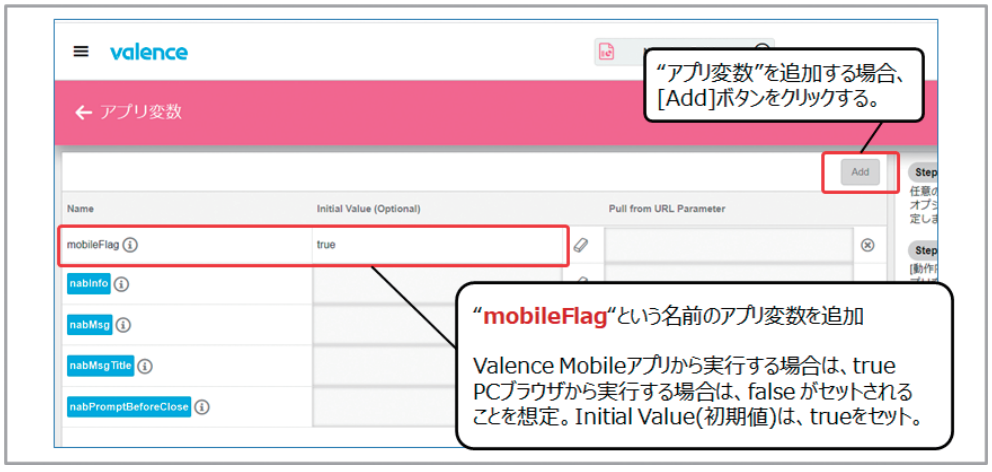
図 17　アプリケーションの共用(デバイスにあわせた最適化)



これは、実行する環境(デスクトップかモバイルか)にあわせて項目単位で、表示/非表示を動的に切り替える事で実現をしている。その実装方法を紹介する。

Valence App Builderではアプリケーション毎に[アプリ変数]と呼ばれる独自の変数を定義し利用する事が可能である。ここでは、実行環境を判別する為のフラグとなる[mobileFlag]変数を一つ追加する。**【図18】**

図 18　アプリ変数を追加



次に、アプリケーションの[動作内容]画面を開く。そして画面上部にある[Startup]イベントに、[スクリプトの実行]アクションを追加する。[Startup]イベントは、アプリケーション起動時に初期実行されるものである。ここに処理を追加することで、実行環境の判別を行う。**【図19】**

図 19　[Starup]イベントに[スクリプトの実行]を追加

Valence App Builderでは、[スクリプトの実行]アクションを使用すれば、ブラウザあるいはValence Mobile アプリ上で実行可能なJavaScript処理を追加する事ができる。単にJavaScriptが記述できるだけでなく、Valenceの為の専用APIも用意されており、これらを活用する事がポイントである。ここでは、実行環境の判別が可能なAPIである”Valence.mobile.Access.isNativePortal”メソッドを使用する。このメソッドは、Valence Mobileアプリから実行した場合に、trueが返却されるものである。(PCブラウザから実行した場合は、falseが返却される。)
このAPIを使用したJavaScriptの記述例が**【ソース1】**である。

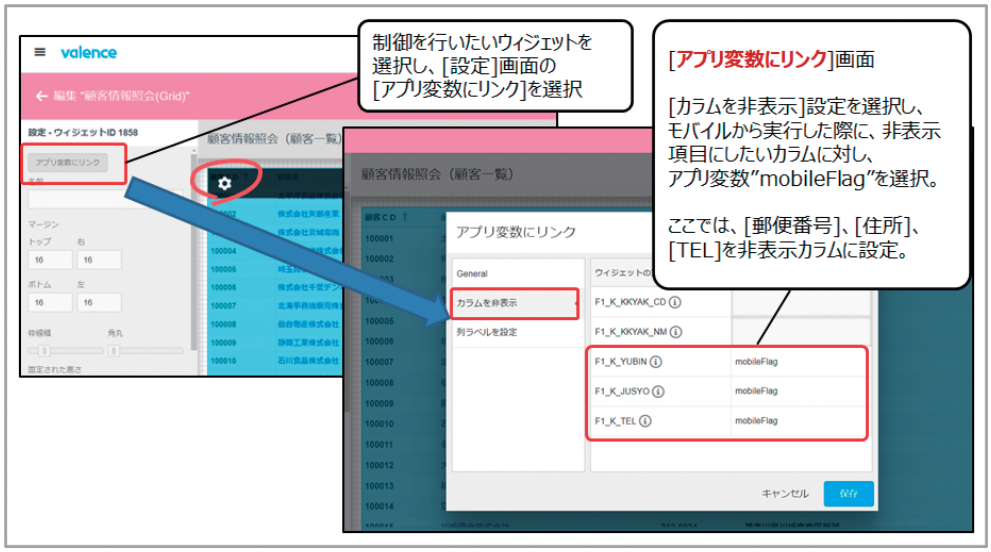
ソース1　Valence Mobileアプリからの実行かどうかを判別

```
//----- Valence Mobileアプリからの実行かどうかを判定
//  ※PCブラウザで実行した場合は、アプリ変数mobileFlagをFalseに変更
if (!Valence.mobile.Access.isNativePortal()) {
    setAppVar('mobileFlag', false);
}
```

このプログラムにより、アプリケーションをPCブラウザから実行した場合、setAppVarメソッドを使用して、アプリ変数”mobileFlag”（初期値は、true）の値をfalseに変更する事ができる。

あとは、アプリケーションの設計画面で表示に関する制御を行いたいウィジェットを選択し、[アプリ変数にリンク]機能を設定すれば良い。今回の場合、顧客一覧リストを表示するGridウィジェットに対し、[アプリ変数にリンク]画面の”カラムを非表示”設定画面を使用し、Valence Mobileアプリから実行時に非表示とする列（カラム）に対し、アプリ変数”mobileFlag”を割り当てればよい。**【図20】**

図 20　アプリ変数を使用した非表示カラムの設定



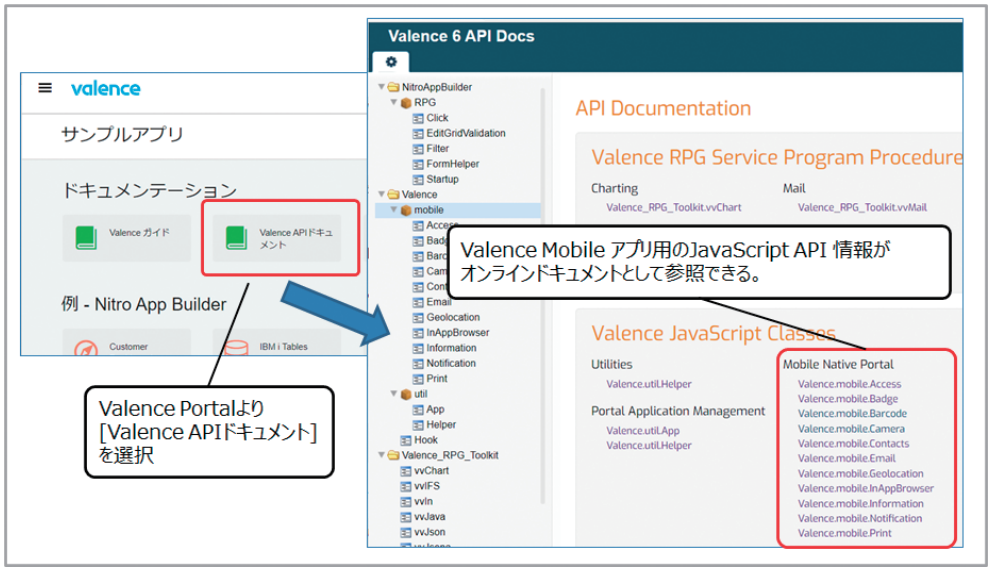
“カラムを非表示”設定画面では、アプリ変数の値が”true”となるカラムを非表示にする事ができる。この設定により、Valence Mobileアプリから実行した場合に、一部の列（カラム）のみを表示するようにカスタマイズできるのである。

このように、アプリ変数を使用することで、ウィジェットの表示方法を制御できる。工夫をすれば、PCブラウザ（デスクトップ）とモバイルとで、一つの同じアプリケーションを共用しながらも、それぞれの実行環境にあわせた画面のカスタマイズが可能となる。今回は、Gridウィジェットのカラムについて制御を行ったが、他のウィジェットもアプリ変数による制御が可能なので、試してみてほしい。

5.デバイス機能活用テクニック

モバイルデバイスを業務で活用するメリットの一つが、PCでは難しかったカメラ等のデバイス機能が活用できることである。Valenceには、Valence Mobileアプリから使用できる専用のJavaScript APIが用意されている。PCブラウザのValence Portalから、[Valence APIドキュメント]を開くと、オンラインドキュメントが呼び出せる。ドキュメントトップページの[Valence JavaScript Classes]が、ブラウザあるいはValence Mobileアプリ上で使用できるクライアント側のJavaScript API一覧となる。この中にある[Mobile Native Portal]の部分が、Valence Mobileアプリで使用可能なモバイルデバイス用API一覧である。**【図21】**

図 21　Valence APIドキュメント(英語)

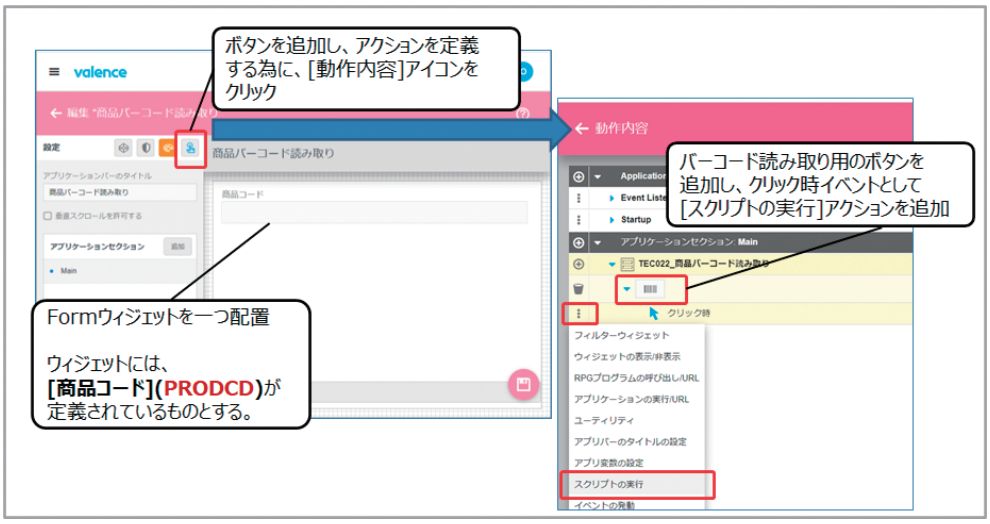


なお、前節の”Valence.mobile.Access.isNativePortal”メソッドも、このモバイルデバイス用APIの一つである。

本節では、デバイス機能の中でも使用頻度が高いと思われるカメラを活用した”バーコードの読み取り機能”の実装および”GPSを使用した現在位置(ロケーション)取得”の実装方法を紹介する。具体的には、”Valence.mobile.Barcode”と”Valence.mobile.Geolocation”の2つのAPIが該当する。

1つ目として、”バーコード読み取り機能”を紹介する。ここでは、[動作内容]画面を使用して、Formウィジェットにある商品コード欄[PRODCD]の箇所にボタンを追加し、このボタンのクリックイベントに[スクリプトの実行]アクションを追加する。**【図22】**

図 22　バーコード読み取り機能の実装（アクション設定）



エディタの中に、”Valence.mobile.Barcode” API を使用したコードを追加する。コードの記述例が、**【ソース 2】**である。

ソース2　バーコード読み取り機能の実装例

```
//===バーコード読み取り処理 (Valence Barcode APIを使用)===
var me = this;
Ext.Viewport.mask();
//----- スキャン開始
Valence.mobile.Barcode.scan({
    scope    : me,
    callback : function (response) {
        Ext.Viewport.unmask();
        //----- レスポンスが空の場合
        if (Ext.isEmpty(response)) {
            Ext.Msg.alert('バーコードスキャン', 'レスポンスがありません');
            return;
        }
        if (response.success) {
            //----- レスポンスがある場合
            if (!response.data.cancelled) {
                //----- 読み取り完了の場合
                cmp.setValues({
                    PRODCD : response.data.text
                });
            } else {
                //----- 読み取りキャンセルの場合
                Ext.Msg.alert('バーコードスキャン', 'キャンセル');
            }
        }
    }
});
```

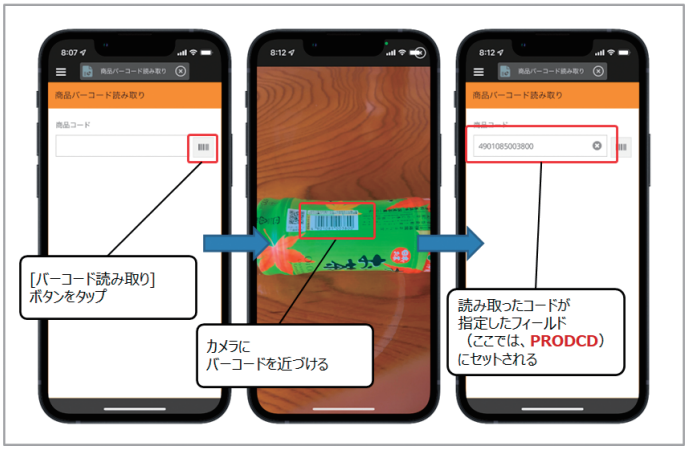
2-①　2-②　2-③

2-①の部分が、APIを使用してカメラを起動しバーコードスキャンを開始する部分である。このメソッドは、バーコードスキャン結果を2-②のようにcallback関数として処理を行う。処理結果をresponse変数として受け取る為、スキャンを正常終了したかどうかをresponse.successで判断する。その上で、読み取りキャンセルでなければ、バーコードの読み取り結果がresuponse.data.textから取得出来る。

2-③のようなsetValuesメソッドを使用すれば、取得した読み取り結果をウィジェット上の特定のフィールド(ここでは"PRODCD"フィールド)の画面値を更新することができる。

このように作成したアプリケーションをValence Mobileアプリから実行すると、デバイスのカメラを活用したバーコードの読み取りが可能になる。**【図23】**

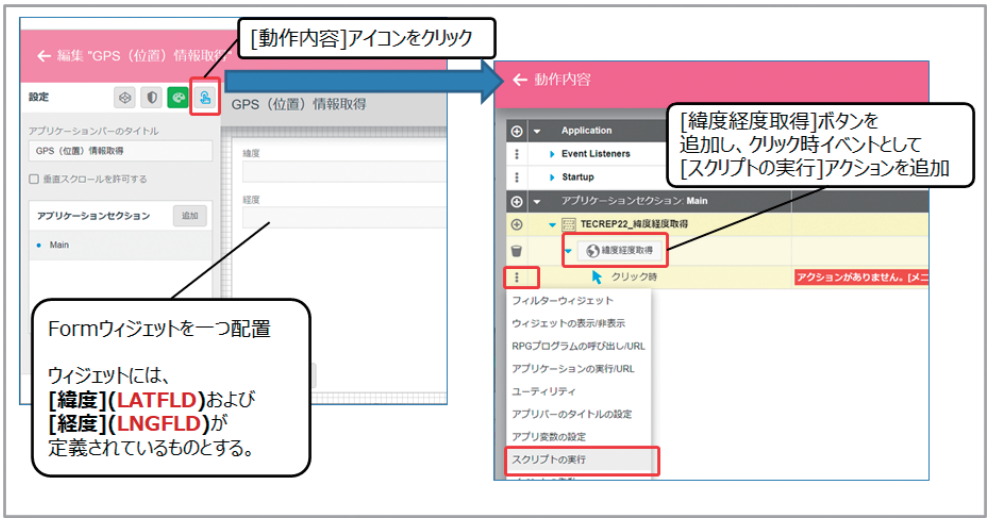
図 23 バーコード読み取りアプリケーション実行例



この実行例では、商品に貼付されたJANコード(1次元バーコード)を読み取っているが、他にもQRコード(2次元バーコード)を読み取ることも可能である。

2つ目に"GPSを使用した現在位置(ロケーション)取得"を紹介する。まず、Formウィジェット上には、GPSから取得したロケーション情報である緯度と経度を保持する為に、2つのフィールド(LATFLD, LNGFLD)を定義する。次に[動作内容]画面で、Formウィジェット上に[緯度経度取得]ボタンを追加し、このボタンのクリックイベントに[スクリプトの実行]アクションを追加する。**【図24】**

図 24 現在位置取得機能の実装(アクション設定)



エディタの中に、”Valence.mobile.Geolocation”APIを使用したコードを追加する。コード記述例が、**【ソース3】**である。

ソース3　現在位置取得（GPS）機能の実装例

```
//----- GPS(現在値)を取得 (Geolocation APIを使用)
var me = this;
Ext.Viewport.mask();
//----- ロケーション取得を開始
Valence.mobile.Geolocation.getCurrentPosition({
    scope    : me,
    callback : function (response) {
        Ext.Viewport.unmask();
        //----- レスポンス無しの場合、処理終了
        if (Ext.isEmpty(response)) {
            return;
        }
        //----- レスポンス結果の確認
        if (response.success) {
            //----- 正しく情報が取得出来た場合
            if (response.coords) {
                //----- 緯度をセット
                if (response.coords.latitude) {
                    cmp.setValues({
                        LATFLD : response.coords.latitude
                    });
                }
                //----- 経度をセット
                if (response.coords.longitude) {
                    cmp.setValues({
                        LNGFLD : response.coords.longitude
                    });
                }
            }
        } else {
            //----- 取得出来なかった場合、エラーメッセージを返す
            Ext.Msg.alert('Error', response.message);
        }
    }
});
```

3-①
3-②
3-③

ソースの考え方は、先程紹介したバーコード読み取りとほぼ同じである。3-①の部分が、APIを使用してGPSを起動し、現在位置の取得を開始する部分である。このメソッドも、取得結果を3-②のようにcallback関数として処理を行う。処理結果をresponse変数として受け取る為、情報の取得が正常終了したかどうかをresponse.successで判断する。その上で、情報が正しく取得できた場合、結果がresponse.coords.latitude（緯度）およびresponse.coords.longitude（経度）から取得出来る。3-③のようにsetValuesメソッドを使用し、取得した結果をウィジェット上のフィールド画面値に反映することができる。

最後に、Formウィジェット上に[Google　Map表示]ボタンを追加し、このボタンのクリックイベントに[アプリケーションの実行/URL]アクションを追加する。このアクションは、任意のURLより外部ブラウザを開く事ができる機能となる。ここで、Formウィジェットにセットされた緯度経度の値をパラメータにGoogle Mapを呼び出せば完成である。**【図25】**

図 25 外部ブラウザ起動(アクション設定)

完成したアプリケーションをValence Mobileアプリから実行する。まず、[緯度経度取得]ボタンをタップすると、GPSを使用して位置情報を取得し、その結果となる緯度経度情報がFormウィジェットに表示される。次に、[GoogleMap表示]ボタンをタップすると、外部ブラウザが立ち上がり、現在位置にピンが打たれた地図が表示されることがわかる。**【図26】**

図 26 現在位置取得(GPS)機能アプリケーション実行例

このように、”Valence.mobile.Geolocation”APIを使用すれば、位置情報を取得出来る為、外部ブラウザへ連動すれば、簡単にGoogle Mapを表示させられることがわかった。

このアプリケーションでは、外部ブラウザを使用してGoogle Mapを表示させているが、なぜValence App Builder標準のMapウィジェットを使用せずに、わざわざ外部ブラウザを呼び出しているのだろうか？
理由は、ValenceのMapウィジェットは、現在位置を表示させる為には、緯度・経度ではなく、その地点の住所情報をセットする必要があるからだ。
つまり、ValenceのAPIで取得した位置情報を元に、住所情報に変換する事ができれば、直接Mapウィジェットを使用する事が可能となる。ただ残念ながら、Valenceの専用APIには、住所変換機能が無い為、住所情報を取得するには、独自に処理を実装する必要がでてくる。本稿の最後のトピックとして、この位置情報から住所情報に変換するテクニックを紹介する。
この変換処理は、Google Map APIを使用すれば実現可能である。Valenceが使用するMapウィジェット自体もGoogle Map APIを使用している為、ValenceでMapウィジェットを使用しているのであれば、Valenceの[ポータル管理]→[設定]の中にある[Google Map用APIキー]を設定しているはずである。この設定があれば、Valence App Builderの[スクリプトの実行]で記述するJavaScriptからでもGoogle Map APIがそのまま使用できる。
今回実装したい地点から住所への変換処理を行う為には、“Geocoding API”を使用すればよい。

具体的な実装手法を紹介する。先程のアプリケーションに[ADDRESS]という名前のアプリ変数を一つ追加し、さらにFormウィジェット上にも、住所を表示する為のフィールド(ADDRESS)を追加する。そして、[現在位置住所取得]ボタンを新たに追加し、そのボタンのクリック時イベントに[スクリプトの実行]を追加する。その中に住所変換ロジックを追加する。ソースの記述例が、**【ソース4】**である。

ソース4　位置情報から住所情報に変換処理を行う実装例

```
//----- フォーム上の緯度と経度を取得
var lat = rec.get('LATFLD');
var lng = rec.get('LNGFLD');

//----- 緯度経度情報を変数として保持
var curlatLng = new google.maps.LatLng(lat, lng);

//----- Google map geocode api を使用して、緯度経度情報から住所を取得
var geocoder = new google.maps.Geocoder();
geocoder.geocode({
    latLng: curlatLng
}, function(results, status) {
    //----- 正常に取得できた場合
    if (status == google.maps.GeocoderStatus.OK) {
        if (results[0].geometry) {
        // 住所を取得
        var adr = results[0].formatted_address;
        //----- フォーム上の[住所]欄に取得した住所をセット
        cmp.setValues({
            ADDRESS : adr
        });
        //----- アプリ変数ADDRESSにも取得した住所をセット
        setAppVar('ADDRESS', adr);
        }
    }
    //----- 正常に取得できなかった場合、ステータスに応じたエラーを返す
    else if (status == google.maps.GeocoderStatus.ZERO_RESULTS) {
        Ext.Msg.alert("住所が見つかりませんでした。");
    } else if (status == google.maps.GeocoderStatus.ERROR) {
        Ext.Msg.alert("サーバ接続に失敗しました。");
    } else if (status == google.maps.GeocoderStatus.INVALID_REQUEST) {
        Ext.Msg.alert("リクエストが無効でした。");
    } else if (status == google.maps.GeocoderStatus.OVER_QUERY_LIMIT) {
        Ext.Msg.alert("リクエストの制限回数を超えました。");
    } else if (status == google.maps.GeocoderStatus.REQUEST_DENIED) {
        Ext.Msg.alert("サービスが使えない状態でした。");
    } else if (status == google.maps.GeocoderStatus.UNKNOWN_ERROR) {
        Ext.Msg.alert("原因不明のエラーが発生しました。");
    }
});
```

4-①　4-②　4-③　4-④　4-⑤

4-①の部分が、Formウィジェットに表示されている緯度と経度を取得する処理である。4-②では、取得した緯度・経度をGoogle Map APIが定めた形式の緯度経度情報の変数に変換している。そして、4-③でGeocoding APIを呼び出して、4-④で変換処理を実行している。geocodeメソッドの実行パラメータとして、4-②で取得した変数値をセットし、結果を関数として受け取っている。受け取った関数内で住所情報を取得するのが、4-⑤の部分である。Results[0].formatted.addressに住所情報が格納される為、この結果をFormウィジェット上の住所欄(ADDRESSフィールド)ならびにアプリ変数"ADDRESS"にセットするという内容である。

これで住所情報を取得できるので、後はアプリ変数"ADDRESS"に格納された住所情報を使って、Mapウィジェットを表示させれば、アプリケーションは完成である。完成したアプリケーションを実行すると、先程同様の手順で緯度経度を取得後に、[現在位置住所取得]ボタンをタップすると、住所欄に変換された住所がセットされることがわかる。最後に、[Map表示]ボタンで、Mapウィジェットを表示している。ここでは、Valenceのウィジェットを使用して、直接アプリケーション上に地図を表示している為、先程の外部ブラウザのパターンのような外付け感のない自然な地図表示になっている事がわかる。**【図27】**

図 27　住所情報を取得するアプリケーション実行例

本節では、デバイス機能の活用として、バーコード読み取りと現在位置取得(GPS)の使用方法を紹介した。ここでは、単体の機能に関する実装方法を紹介したが、最後に実際のアプリケーションでこれらの機能を活用したデモを紹介する。とは言っても実行している実際の様子をこの紙面でお伝えするのは難しい為、今回弊社ホームページ上にある[Migaro. 技術Tips]サイトに、モバイルアプリケーションのデモ動画をアップした。次のURLから動画が確認できるので、Valence Mobileアプリを使用したApp Builderアプリケーションの実装例として是非確認してほしい。

URL:https://www.migaro.co.jp/tips/1640/

6.さいごに

本稿では、Valenceにおけるモバイルアプリケーション開発テクニックとして、アプリケーション作成の基本からモバイルデバイス機能の活用方法までを具体例をあげながら紹介してきた。これまで、ValenceについてはPCブラウザの機能が中心で、モバイル開発に関する情報が少なかったので、本稿を足掛かりとして、皆様もValenceを使用したモバイルアプリ開発にチャレンジ頂ければ幸いである。

MIGARO. Technical Report

既刊号バックナンバー

電子版・書籍(紙)媒体で提供中!

https://www.migaro.co.jp/contents/support/technical_report/

No.1 2008年秋

お客様受賞論文

● 最優秀賞

直感的に理解できるシステムを目指して
一情報の"見える化"の取り組み

石井 裕昭 様／豊鋼材工業株式会社

● ゴールド賞

運用部間にサプライズをもたらしたDelphi/400

春木 治 様／株式会社ロゴスコーポレーション

● シルバー賞

JACi400使用によるWebアプリケーション
開発工数削減

中富 俊典 様／日本梱包運輸倉庫株式会社

Delphi/400 を利用したWeb受注システム

飯田 豊 様／東洋佐々木ガラス株式会社

● 優秀賞

Delphi/400による
販売管理システム(FAINS)について

藤田 建作 様／株式会社船井総合研究所

技研化成の新基幹システム再構築

藤田 健治 様／技研化成株式会社

SE論文

はじめてのDelphi/400プログラミング

畑中 侑／システム事業部 システム2課

Delphi/400とExcelとの連携

中嶋 祥子／ RAD事業部 技術支援課

連携で広がるDelphi/400 活用術

尾崎 浩司／システム事業部 システム2課

フォーム継承による効率向上開発手法

吉原 泰介／ RAD事業部 技術支援課

APIを利用した出力待ち行列情報の取得方法

鶴巣 博行／ RAD事業部 技術支援課

Delphiテクニカルエッセンス Q&A 集

吉原 泰介／ RAD事業部 技術支援課

JACi400を使ってRPGでWeb画面を制御する方法

松尾 悦郎／システム事業部 システム2課

あなたはブラインドタッチができますか?

福井 和彦／システム事業部 システム1課

No.2 2009年秋

お客様受賞論文

●最優秀賞

JACi400で既存Webサービスの内製化を実現

佐々木 仁志 様／株式会社ジャストオートリーシング

●ゴールド賞

.NET環境でのDelphi/400の活用

福田 祐之 様／林兼コンピューター株式会社

●シルバー賞

5250で動作する「中古車在庫照会プログラム」の
GUI 化

佐久間 雄 様／株式会社ケーユー

●優秀賞

Delphiによる輸入システム「MISYS」の再構築

秦 榮禧 様／株式会社モトックス

Delphi/400による物流システムの再構築

仲井 学 様／西川リビング株式会社

Delphi/400で開発し
3台のオフコンを1台のIBM iへ統合

島根 英行 様／シルフ

SE論文

JACi400環境でマッシュアップ!

岩田 真和／ RAD事業部 技術支援課

Delphi/400を利用したはじめてのWeb開発

福岡 浩行／システム事業部 システム2課

Delphi/400を使用した
Webサービスアプリケーション

尾崎 浩司／システム事業部 システム3課

Delphi/400によるネイティブ資産の応用活用

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

RPGでパフォーマンスを制御

松尾 悦郎／システム事業部 システム1課

MKS Integrityを利用したシステム開発

宮坂 優大・田村 洋一郎／システム事業部 システム1課

No.3 2010年秋

お客様受賞論文

●最優秀賞

建物のクレーム情報管理システム「アフターサービスDB」について

大橋 良之　様／東レ建設株式会社

●ゴールド賞

Delphi/400 で「写真管理ソフト」と「スプールファイルのPDF化ソフト」を自社開発

寒河江 幸喜　様／日綜産業株式会社

●シルバー賞

Delphi/400で鉄鋼受発注業務を統一し鉄鋼EDIも実現

柿本 直樹　様／合鐵産業株式会社

●優秀賞

Delphi/400で EIS（Executive Information System）の高速化

小島 栄一 様／西川計測株式会社

イントラでのPHP-Delphi-RPG連携

仲井 学 様／西川リビング株式会社

Delphi/400を使った取引先管理システム

大崎 貴昭 様／森定興商株式会社

SE論文

Delphi/400 ローカルキャッシュ活用術

中嶋 祥子／ RAD事業部 技術支援課

Delphi/400 帳票開発ノウハウ公開

尾崎 浩司／システム事業部 システム3課

Delphi/400でドラッグ＆ドロップを制御

辻林 涼子／システム事業部 システム2課

Delphi/400のモジュールバージョン管理手法

前田 和寛／システム事業部 システム2課

Delphi/400 WebからのPDF出力

福井 和彦・清水 孝将／システム事業部システム3課・システム2課

Delphi/400でFlash 動画の実装

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

No.4 2011年秋［創立20周年記念号］

お客様受賞論文

●最優秀賞

全社の経費処理業務を効率化した「e総務システム」

鈴木 英明 様／阪和興業株式会社

●ゴールド賞

「Web進捗管理システム」でリアルタイム性を実現

堀内 一弘 様／エスケーロジ株式会社

●シルバー賞

「営業奨励金申請書」をたった2日間で開発

簑島 宏明 様／株式会社ケーユーホールディングス

液体輸送における「配車支援システム」の構築

桂 哲 様／ライオン流通サービス株式会社

SE論文

グラフ活用リファレンス

中嶋 祥子／ RAD事業部 技術支援課

Webサービスを利用して機能UP！

福井 和彦・畑中 侑／システム事業部 システム2課

OpenOffice実践活用

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

VCL for the Web 活用TIPS紹介

尾崎 浩司／システム事業部 プロジェクト推進室

JC/400でJavaScript 活用

清水 孝将／システム事業部 システム1課

jQuery連携で機能拡張

國元 祐二／ RAD事業部 技術支援課 顧客サポート

MIGARO. Technical Report
既刊号バックナンバー

No.5 2012年秋[創刊5周年記念]

お客様受賞論文

【部門1】

●最優秀賞

JC/400による取引先とのWeb-EDI システム構築

久保田 佳裕 様／極東産機株式会社

●ゴールド賞

DelphiとExcelを使用した帳票コストの削減

大久保 治高 様／合鐵産業株式会社

もっと見やすく、もっと使いやすい画面を

新谷 直正 様／株式会社アダル

【部門2】

●優秀賞

Delphi/400で確認業務の効率化

為国 順子 様／ベネトンジャパン株式会社

取引先申請システムでの稟議書作成ワークフロー

大崎 貴昭 様／森定興商株式会社

Delphi/400でIBM iのストアードプロシージャを利用し、SQL処理を高速化

島根 英行 様／シルフ

SE論文

InstallAwareを使ったDelphi/400運用環境の構築

中嶋 祥子／ RAD事業部 技術支援課 顧客サポート

カスタマイズコンポーネント入門 Delphi/400開発効率向上

前田 和寛／システム事業部 システム2課

Delphi/400スマートデバイスアプリケーション開発

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

DataSnapを使用した3層アプリケーション構築技法

尾崎 浩司／システム事業部 プロジェクト推進室

JC/400でポップアップウィンドウの制御＆活用ノウハウ

清水 孝将・伊地知 聖貴／システム事業部 システム1課

【創刊5 周年記念】

ミガロ.SE座談会 ―お客様と共に歩む、お客様への熱い思い

No.6 2013年秋

お客様受賞論文

【部門1】

●最優秀賞

自社用開発フレームワークの構築

駒田 純也 様／ユサコ株式会社

●ゴールド賞

Delphi/400でCTI開発および関連機能組み込み

仲井 正人 様／株式会社スマイル・ジャパン

●シルバー賞

IBM WebFacingからJC/400への移行・リニューアル手法

八木 秀樹 様／極東産機株式会社

Delphi/400とDelphiを利用したIBM i資源の有効活用

小山 祐二 様／澁谷工業株式会社

発注システムをVBからDelphiへ移植しリニューアル

川島 寛 様／株式会社タツミヤ

【部門2】

●優秀賞

5250画面を使用せずにAS/400スプールファイルをコントロールする

白井 昌哉 様／太陽セメント工業株式会社

Delphi/400を利用した承認フロー導入によるIT内部統制構築

塚本 圭一 様／ライオン流通サービス株式会社

SE論文

FastReportを使用した帳票作成入門

尾崎 浩司／ RAD事業部 営業推進課

Delphi/400で開発する64bitアプリケーション

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

Webコンポーネントのカスタマイズ入門

佐田 雄一／システム事業部 システム1課

Indyを利用したメール送信機能開発

辻野 健・前坂 誠二／システム事業部 システム2課

Windowsテキストファイル操作ノウハウ

小杉 智昭／システム事業部 プロジェクト推進室

JC/400 Webアプリケーションのユーザー管理・メニュー管理活用術

吉原 泰介・國元 裕二／ RAD事業部 技術支援課 顧客サポート

No.7 2014年秋

お客様受賞論文

【部門1】

●最優秀賞

Delphi/400による生産スケジューラの再構築

柿村 実 様／東洋佐々木ガラス株式会社

●ゴールド賞

**Delphi/400およびDelphiを利用した
オンライン個人別メニューの構築**

小山 祐二 様／澁谷工業株式会社

●シルバー賞

IBM iとDelphi/400のコラボレーション

新谷 直正 様／株式会社アダル

●シルバー賞

荷札発行システムリプレースについて

仲井 学 様／西川リビング株式会社

【部門2】

●優秀賞

**Delphi/400バージョンアップのための
クライアント環境構築**

普入 弘 様／株式会社エイエステクノロジー

●優秀賞

外出先からメールでリアルタイム在庫を問い合せ

島根 英行 様／シルフ

SE論文

iOS/Androidネイティブアプリケーション入門

吉原 泰介／ RAD事業部 技術支援課

ファイル加工プログラミングテクニック

小杉 智昭／ システム事業部 プロジェクト推進室

FastReportを使用した帳票作成テクニック

前坂 誠二／ システム事業部 システム2課

大量データ処理テクニック

佐田 雄一／システム事業部　システム１課

スマートデバイスWEBアプリケーション入門

尾崎 浩司／ RAD事業部 営業推進課
國元 祐二／ RAD事業部 技術支援課

No.8 2015年秋

お客様受賞論文

【部門1】

●最優秀賞

iPod Touchの業務利用開発と検証

石井 裕昭 様／豊鋼材工業株式会社

●ゴールド賞

ブランク加工図管理システムの構築

小山 祐二 様／澁谷工業株式会社

●シルバー賞

**Delphi/400でスプールファイル管理
（WRKSPLF コマンドの活用）**

三好 誠 様／ユサコ株式会社

●シルバー賞

予算管理システムの構築

川島 寛 様／株式会社タツミヤ

●シルバー賞

送状データ送信システムのWeb化について

仲井 学 様／西川リビング株式会社

【部門2】

●優秀賞

**繰り返しDB参照時の
ClientDataSetのFirst 機能について**

牛嶋 信之 様／株式会社佐賀鉄工所

●優秀賞

IBM iのカレンダーを基準に他のシステムを稼働

福島 利昭 様／株式会社ランドコンピュータ

SE論文

フレームを利用した開発手法

前坂 誠二／システム事業部 システム2課

**Windowsタブレット用に
カスタムソフトウェアキーボードを実装**

福井 和彦／システム事業部 プロジェクト推進室

マルチスレッドを使用したレスポンスタイム向上

尾崎 浩司／ RAD事業部 営業・営業推進課

AndroidアプリケーションのNFC機能活用

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

スマートデバイス開発で役立つ画面拡張テクニック

國元 祐二／ RAD事業部 技術支援課 顧客サポート

MIGARO. Technical Report
既刊号バックナンバー

No.9 2016年秋

お客様受賞論文

【部門1】

●最優秀賞

IBM iの見える化で実現するアジャイル開発

吉岡 延泰 様／日本調理機株式会社社

●ゴールド賞

Windows Like 5250への道のり

小山 祐二 様／澁谷工業株式会社

●シルバー賞

Delphiプログラム管理ソフトの開発

牛嶋 信之 様／株式会社佐賀鉄工所

【部門2】

●優秀賞

Delphi/400を利用した定型業務のPDF化

佐藤 岳 様／ライオン流通サービス株式会社

●優秀賞

ちょい足しモバイル

仲井 正人 様／株式会社スマイル・ジャパン

●優秀賞

AS/400の受注データをWebで社員に公開

福島 利昭 様／株式会社ランドコンピュータ

SE論文

iOSモバイルアプリ開発のデザイニングテクニック

前坂 誠二／システム事業部 システム2課

新データベースエンジンFireDACを使ってみよう！

福井 和彦／システム事業部 プロジェクト推進室

Delphi/400 最新プログラム文法の活用法

尾崎 浩司／ RAD事業部 営業・営業推進課

FastReportを活用した電子帳票作成テクニック

宮坂 優大／システム事業部 システム1課

Beacon技術によるIoT活用の第一歩

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

Web&ハイブリッドアプリ開発で役立つ IBM i&ブラウザデバッグテクニック

國元 祐二／ RAD事業部 技術支援課 顧客サポート

No.10 2017年秋

Migaro.Technical Report 創刊10周年記念

パートナー様からの祝辞

武藤 和博 様／日本アイ・ビー・エム株式会社

藤井 等 様／エンバカデロ・テクノロジーズ日本法人

Serge Charbit 様／ SystemObjects Corporation

飯田 恭子 様／アイマガジン株式会社

お客様からの祝辞・お客様の声

石井 裕昭 様／豊鋼材工業株式会社

牛嶋 信之 様／株式会社佐賀鉄工所

大崎 貴昭 様／森定興商株式会社

川島 寛 様／株式会社タツミヤ

久保田 佳裕 様／極東産機株式会社

駒田 純也 様／ユサコ株式会社

小山 祐二 様／澁谷工業株式会社

寒河江 幸喜 様／日綜産業株式会社

佐々木 仁志 様／株式会社ジャストオートリーシング

佐藤 岳 様／ライオン流通サービス株式会社

白井 昌哉 様／太陽エコブロックス株式会社

仲井 学 様／西川リビング株式会社

福島 利昭 様／株式会社ランドコンピュータ

お客様座談会

石井 裕昭 様／豊鋼材工業株式会社

駒田 純也 様／ユサコ株式会社

寒河江 幸喜 様／日綜産業株式会社

仲井 学 様／西川リビング株式会社

上甲 將隆／株式会社ミガロ.

司会　飯田 恭子 様／アイマガジン株式会社

お客様受賞論文

【部門1】

●最優秀賞

貸金庫と鍵のマッチング業務をDelphi/400で実現
―文字認識データと基幹システムデータを統合

佐藤 正 様／株式会社富士精工本社

●ゴールド賞

Windowsタブレット導入による
工作部材料受入業務改革

小山 祐二 様／澁谷工業株式会社

【部門2】

●優秀賞

Delphi/400を利用した
各拠点PINGコマンド簡素化

松垣 秀昭 様／ライオン流通サービス株式会社

汎用的な帳票出力画面

牛嶋 信之 様／株式会社佐賀鉄工所

バーコードリーダー読み取り後、
次の入力位置にカーソルを自動遷移させる技術

上総 龍央 様／キョーラクシステムクリエート株式会社

IBM iのスプールファイル参照機能を
Webで構築

福島 利昭 様／株式会社ランドコンピュータ

SE論文

デスクトップアプリケーション開発でも役立つ
FireMonkey活用入門

尾崎 浩司／ RAD事業部 営業・営業推進課

Delphi/400バージョンアップに伴う
文字コードの違いと制御

宮坂 優大／システム事業部 システム1課

FastReportへの
効率的な帳票レイアウトコンバート

畑中 侑／システム事業部 システム2課

IBM iトリガー機能を活かした
セキュリティログ対応

八木沼 幸一／システム事業部 プロジェクト推進室

アプリケーションテザリングを利用した
PC&モバイルアプリケーション連携

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

SmartPad4iの運用で役立つWEB サーバー機能

國元 祐二／ RAD事業部 技術支援課 顧客サポート

No.11 2018年秋

お客様受賞論文

【部門1】

●最優秀賞

Excelテンプレートを使用した帳票出力機能の開発

駒田 純也 様／ユサコ株式会社

●ゴールド賞

SP4iの活用による製品検査チェックシステムの構築

八木 秀樹 様／極東産機株式会社

【部門2】

●優秀賞

配車支援システムをDelphi/400で再構築

村上 稔明 様／ライオン流通サービス株式会社

一般シール受注入力業務のDelphi/400化

寺西 健一 様／大阪シーリング印刷株式会社

Delphi/400による無線ハンディターミナルの
データ集約の仕組みの実装

寺西 健一 様／大阪シーリング印刷株式会社

SE論文

OLEを利用したExcel出力の
パフォーマンス向上手法

薬師 尚之／システム事業部 システム2課

FireDAC実践プログラミングテクニック

佐田 雄一／システム事業部 システム1課

RESTによるWebサービスを活用した
機能拡張テクニック

尾崎 浩司／ RAD事業部 営業・営業推進課

Google Maps Platformを使用した
アプリケーション開発テクニック

福井 和彦・小杉 智昭／システム事業部 プロジェクト推進室

RAD Serverを使った
新しい多層アプリケーション構築

吉原 泰介／ RAD事業部 技術支援課

JC/400からSP4iへのマイグレーションノウハウ

吉原 泰介・國元 祐二／ RAD事業部 技術支援課

MIGARO. Technical Report
既刊号バックナンバー

No.12 2019年秋

お客様受賞論文

【部門1】

●最優秀賞

Delphi/400による
電子帳簿保存法スキャナ保存制度への挑戦

石井 裕昭 様／豊鋼材工業株式会社

●ゴールド賞

出荷業務従事者のモチベーションアップ大作戦
—Delphi/400で基幹データの見える化

池田 純子 様／錦城護謨株式会社

●ゴールド賞

iPhoneを利用した
宝飾品の販売系システムをSP4i で構築

島本 佳昭 様／株式会社大月真珠

【部門2】

●優秀賞

SmartPad4iによる、導入後の改修を意識した設計

西村 直也 様／株式会社ジャストオートリーシング

●優秀賞

Valence を使用した 温度・湿度ログ表示
－サーバー室内温度・湿度変化の見える化－

中谷 佳史 様／株式会社保健科学西日本

●優秀賞

RPGソースコードをValence File Editorで改修

福島 利昭 様／株式会社ランドコンピュータ

●特別寄稿論文

統合開発環境Cobosのご紹介
—RPG・SP4iの効率的な開発に

SE論文

IBM iデータベースへのFTP データ転送手法の紹介

田村 洋一郎・宮坂 優大・都地 奈津美／システム事業部 システム1課

FireMonkeyの活用
カメラコンポーネントを使ったアプリ

畑中 侑／システム事業部 システム2課

Subversionを使用したDelphiソース管理

福井 和彦／システム事業部 プロジェクト推進室

Enterprise Connectorsを利用した
クラウド連携テクニック

佐田 雄一／ RAD 事業部 技術支援課

SmartPad4iのインターフェース機能拡張

國元 祐二／ RAD事業部 技術支援課

Valence App Builder RPG連携テクニック

尾崎 浩司／ RAD事業部 技術支援課

No.13 2020年秋

SE論文

Windowsタブレット向け
VCLアプリケーション作成テクニック

都地 奈津美／システム事業部 システム1課

iOSモバイルアプリケーションによる
ファイル閲覧機能作成

前坂 誠二／システム事業部 システム2課

Delphi 10シリーズ
VCLプログラム開発の最新トピックス

佐田 雄一／ RAD事業部 技術支援課

Eclipse(Cobos4i)を使用したSmartPad4i開発術

國元 祐二／ RAD事業部 技術支援課

Valence最新バージョン 進化のポイント

尾崎 浩司／ RAD事業部 技術支援課

No.14 2021年秋

SE論文

新規VCLコントロールを利用した
ユーザーインターフェース改善テクニック

畑中 侑／システム事業部 システム2課

IntraWebを使用したWeb開発のTips紹介

福井 和彦　石山 智也／システム事業部 システム１課

FireDACを活用した
Delphi/400ロジック最新化テクニック

佐田 雄一／RAD事業部 技術支援課

洗練されたUIデザインを簡単に実現！
HTML作成テクニック

國元 祐二／RAD事業部 技術支援課

Valenceにおける帳票出力について

尾崎 浩司／RAD事業部 技術支援課

MIGARO.
Technical Report 2022

No.15 2022年

2022年12月1日 初版発行

発行

株式会社ミガロ.

〒556-0017

大阪府大阪市浪速区湊町 2-1-57

難波サンケイビル 13F

TEL:06(6631)8601

FAX:06(6631)8603

https://www.migaro.co.jp/

発行人

上甲 將隆

編集協力・印刷

芳武印刷株式会社

株式会社ミガロ.

https://www.migaro.co.jp/

本社

〒556-0017

大阪市浪速区湊町2-1-57

難波サンケイビル13F

TEL:06(6631)8601

FAX:06(6631)8603

東京事業所

〒100-0013

東京都千代田区霞が関3-7-1

霞が関東急ビル2F

TEL:03(5510)5701

FAX:03(5510)5702