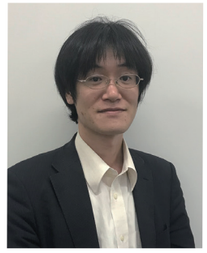
Delphi/400

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】**

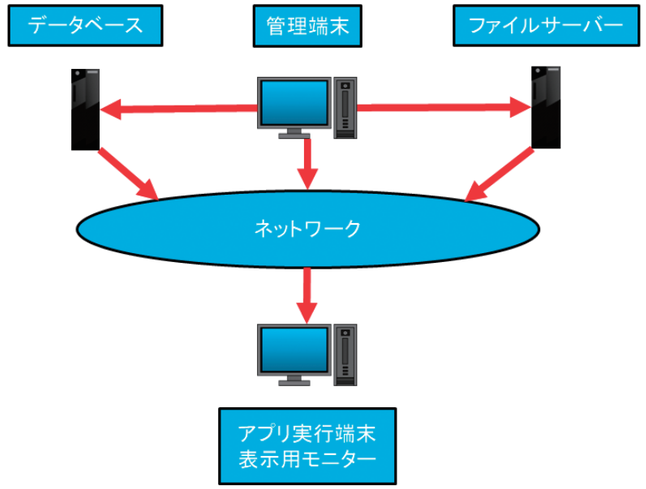
図 1　街中で見られるデジタルサイネージ

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

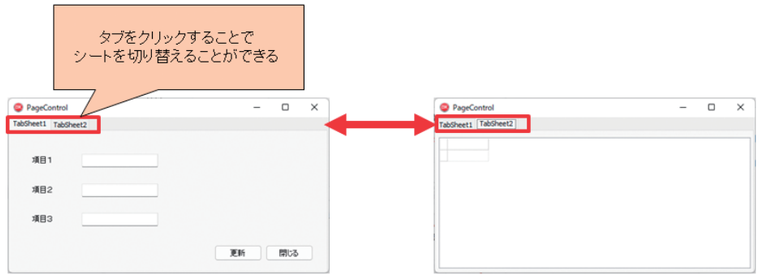
●ファイルサーバー
→デジタルサイネージアプリで表示する画像・動画ファイルを保存しておくサーバー

●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】

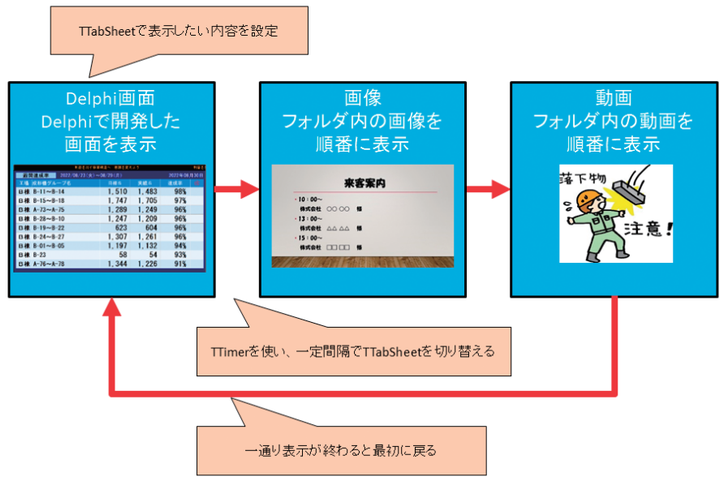
図 3　TpageControlとタブイメージ



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:¥.....

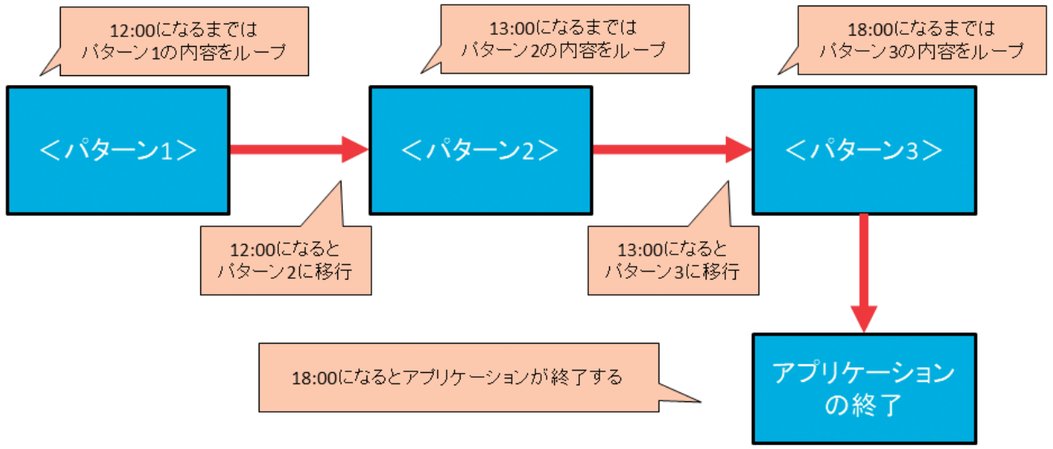
Delphi画面
画像
動画

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

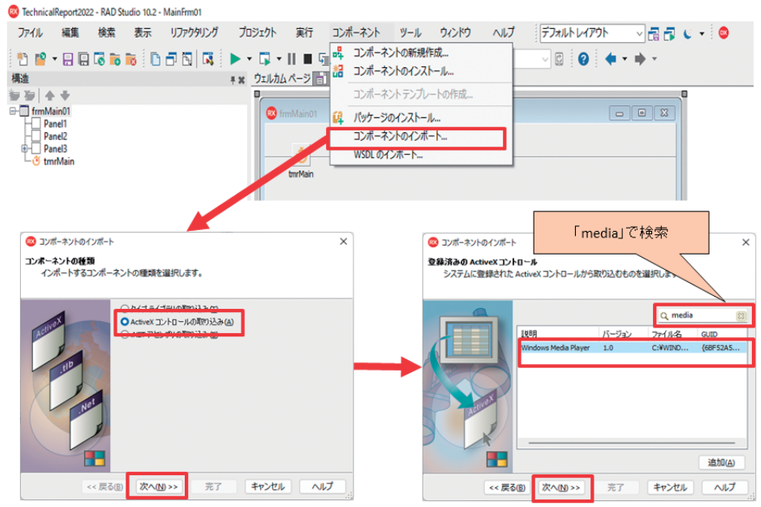
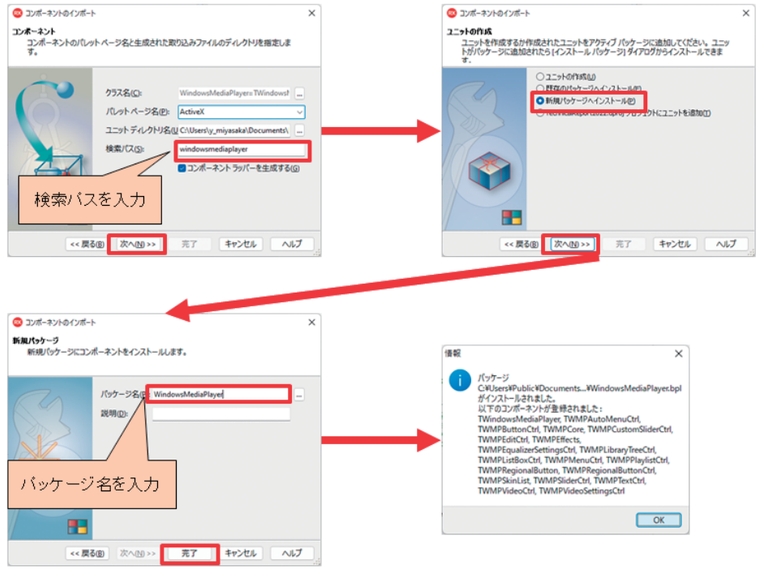


図 10　TWindowsMediaPlayerのインストール2



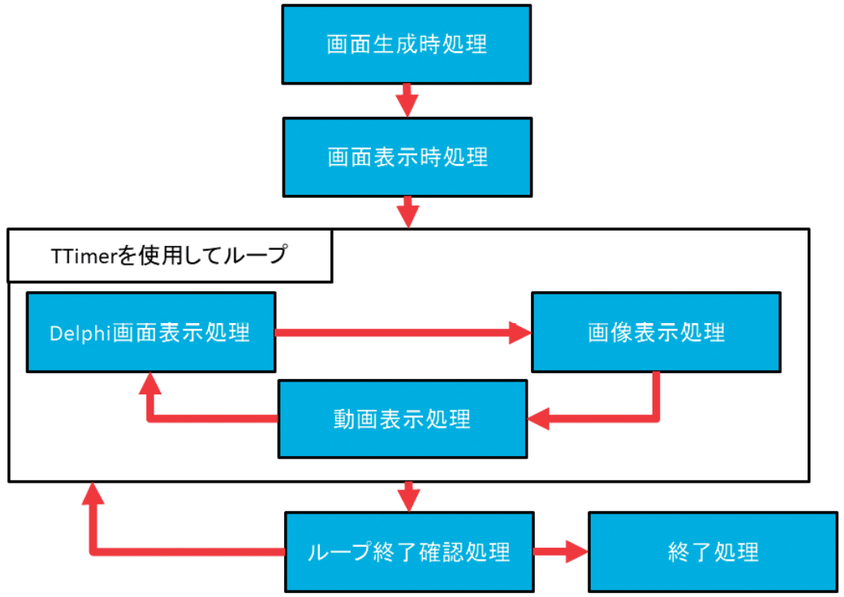
5.開発方法

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。

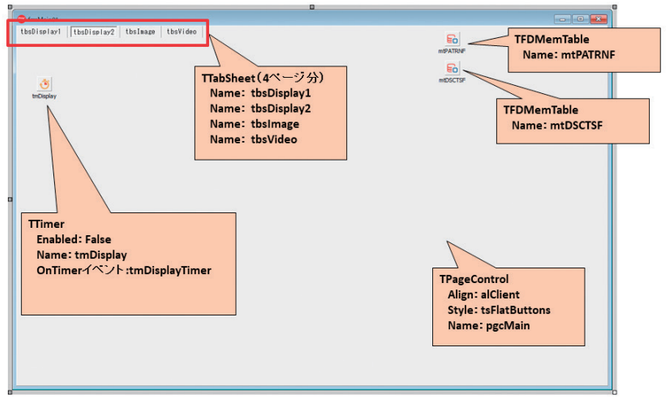
図 11　処理の流れについて



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】

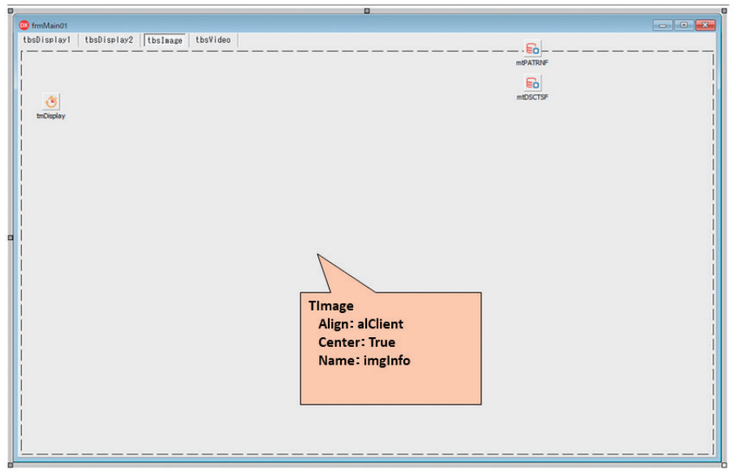
図 13　Delphi画面例

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)

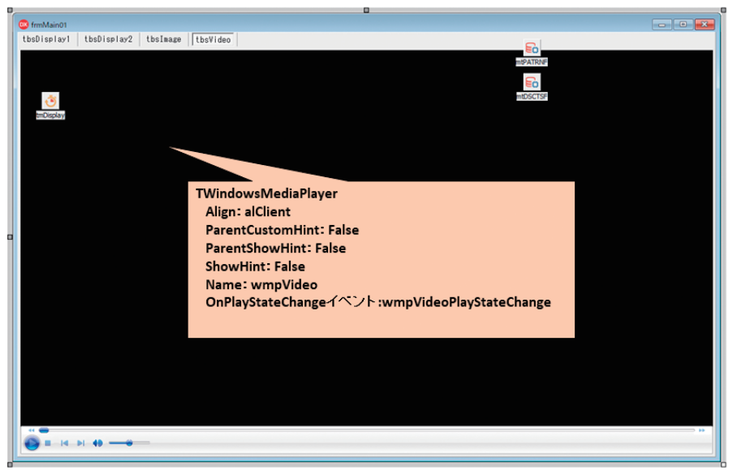
●「tbsImage」
→画像を表示するタブ
【図14】に従ってコンポーネントを配置し各プロパティを
設定する

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;                                              ├ 2-③
    SQL.Text := 'SELECT * FROM DSCTSF ';                    │
    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                                                   ├ 3-①
  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;                                     ├ 3-②
  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;
```

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]);  ← 7-①
  end;
end;
```

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                          ← 8-①
  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                                  ← 8-②
  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;
```

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];   ← 9-①
    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;
```

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

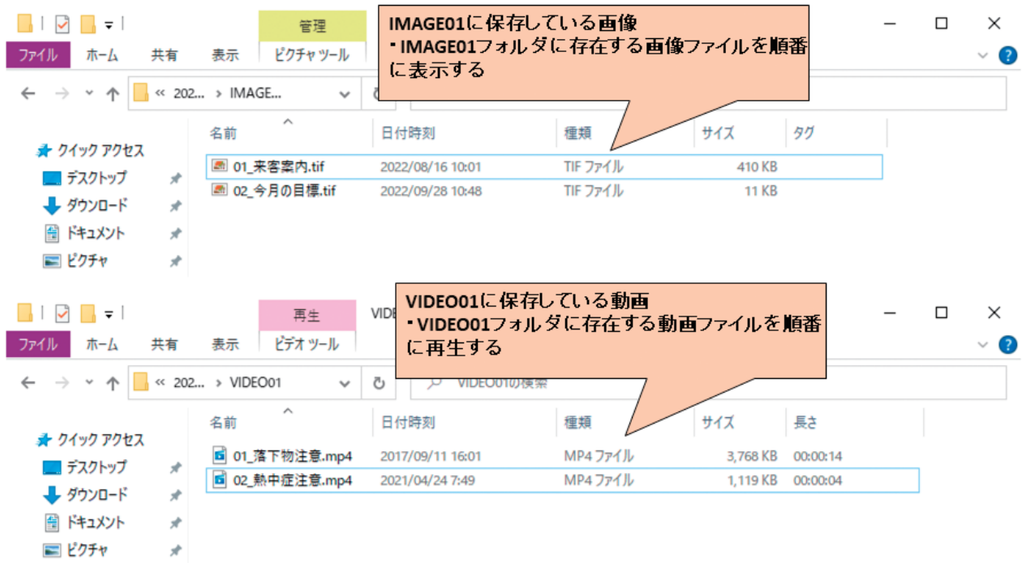


図 17 アプリの実行

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。