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。

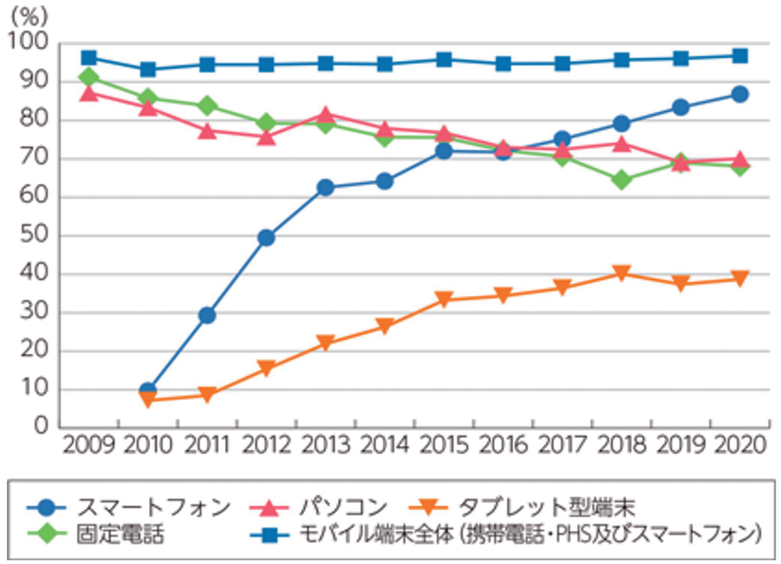
1.はじめに

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。

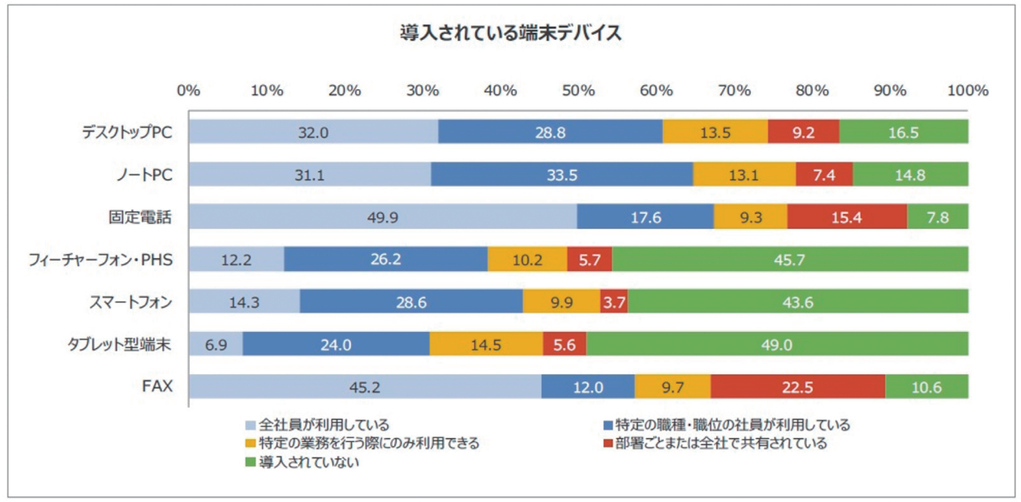
本稿では、モバイル端末のカメラ機能とDelphi/400アプリケーションの連携で、業務の効率化を図る方法を紹介する。モバイル端末を未導入の状態から導入した場合を想定して執筆しているため、これから導入を検討されている方にもぜひ一読いただければ幸いである。

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

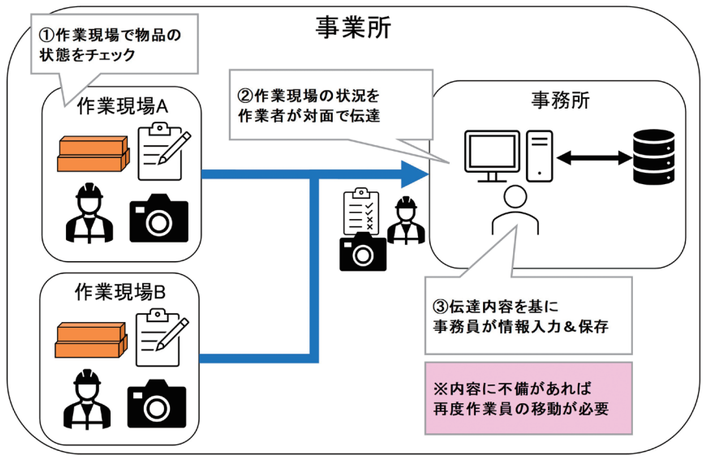
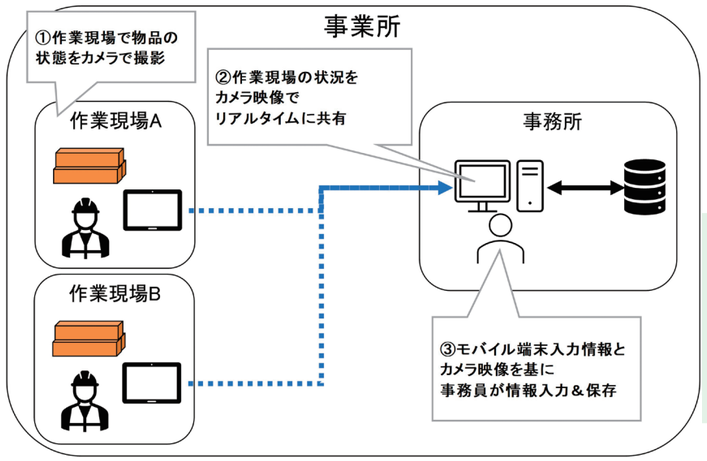


図 4　モバイル端末導入後フロー



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

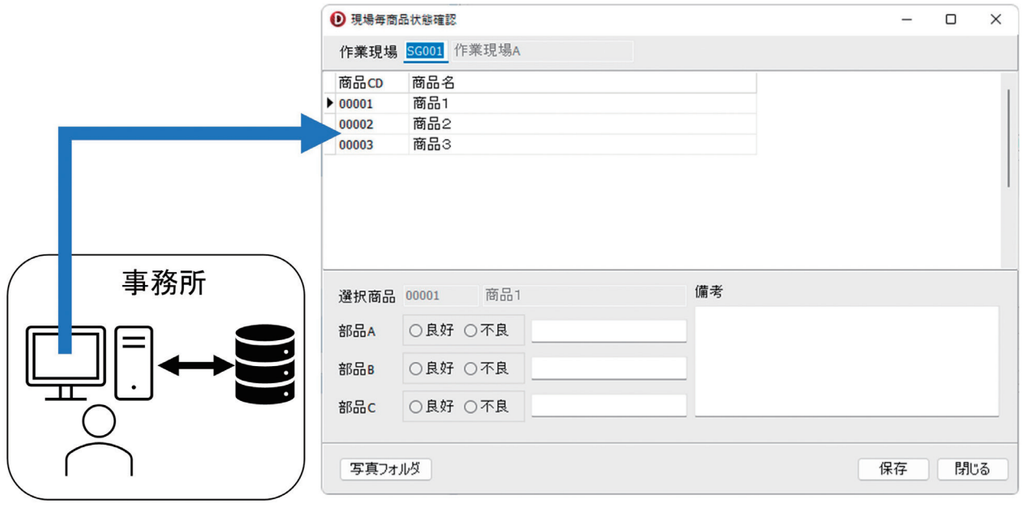


図 6 モバイル導入後アプリケーション（PC）

図 7 モバイル端末導入後アプリケーション（iPad）

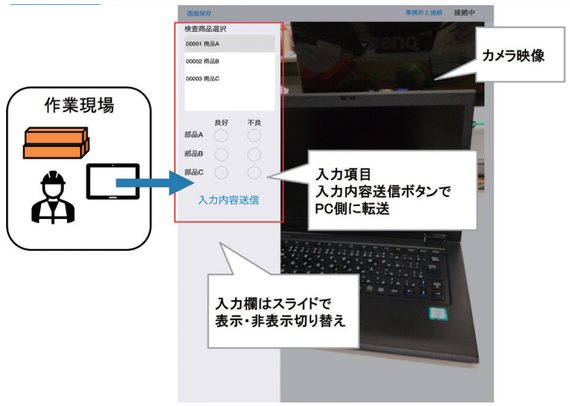


図 8 モバイル端末→PC端末へのデータ共有

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

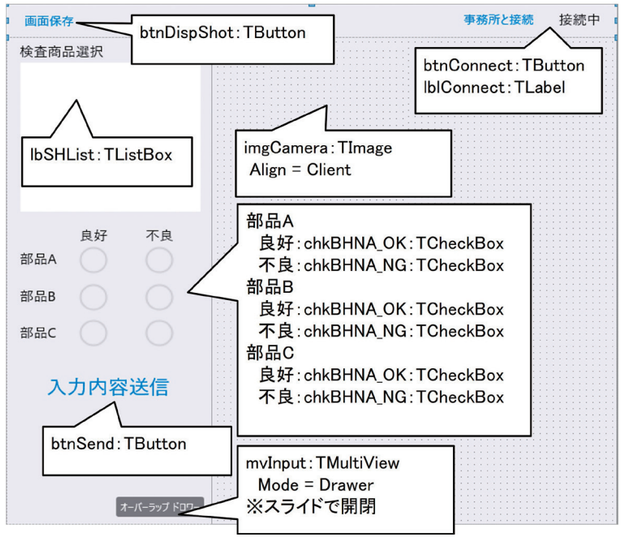
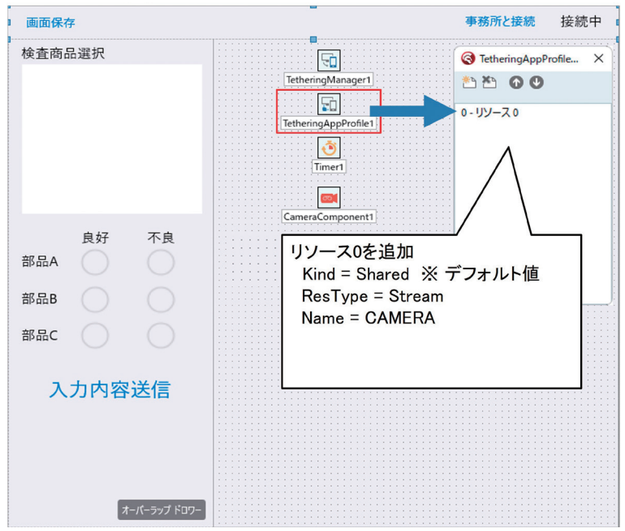


図 10　モバイル端末：非表示コンポーネント

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

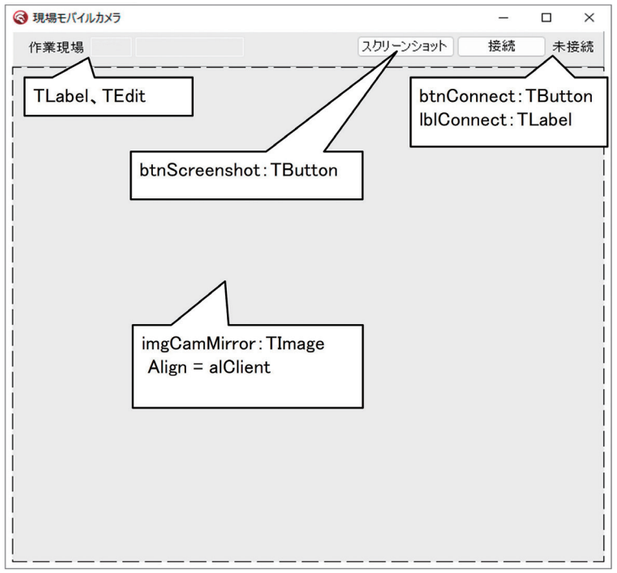


図 13　PC端末：非表示コンポーネント

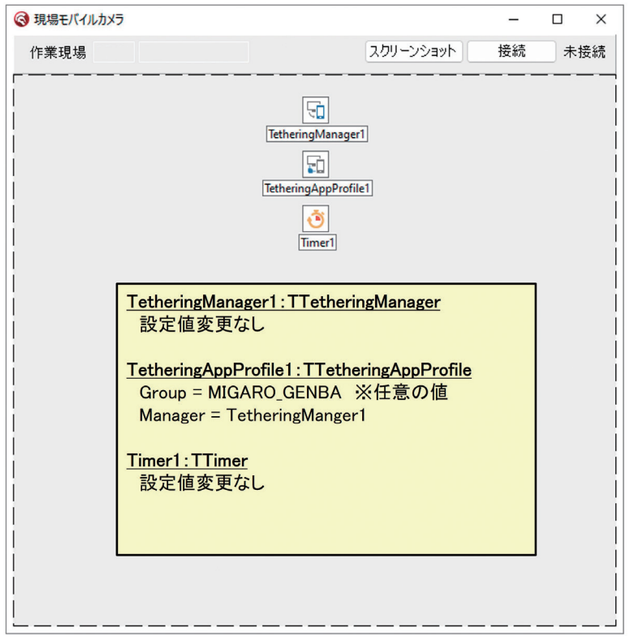


図 14　PC端末：リソースの設定

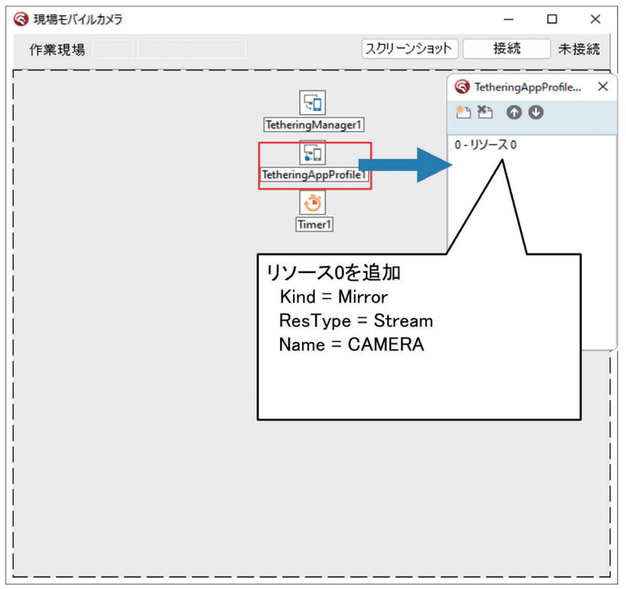
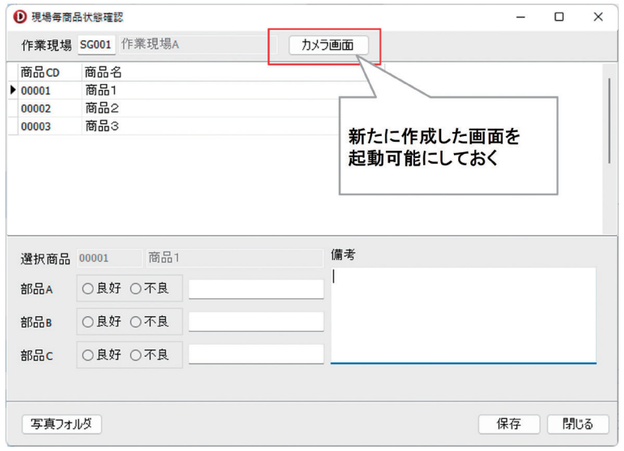


図 15　現行使用画面の改修



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Make Screenshot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

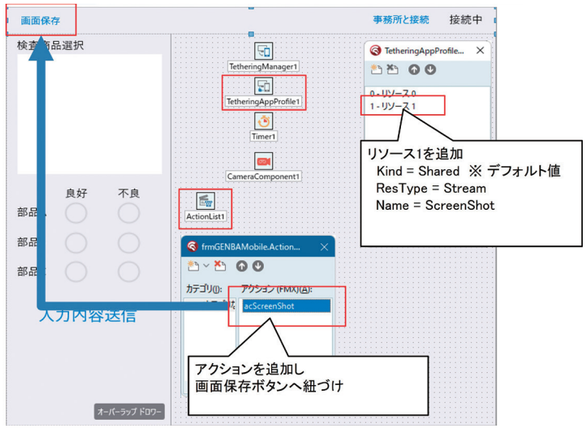
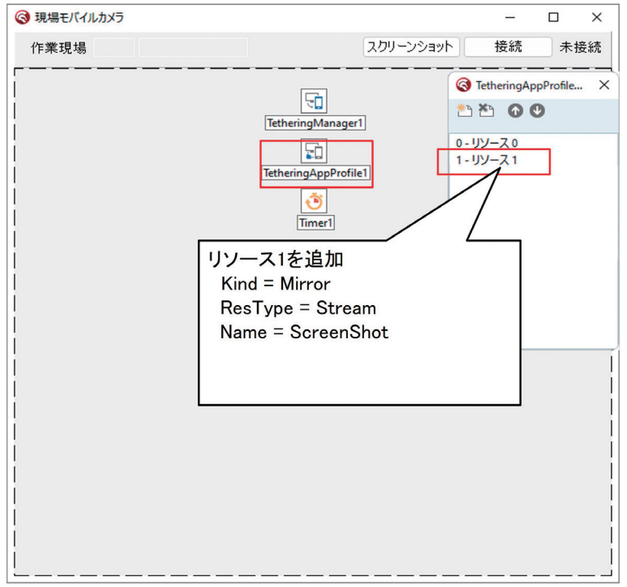


図 17　PC端末：リソースの追加



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

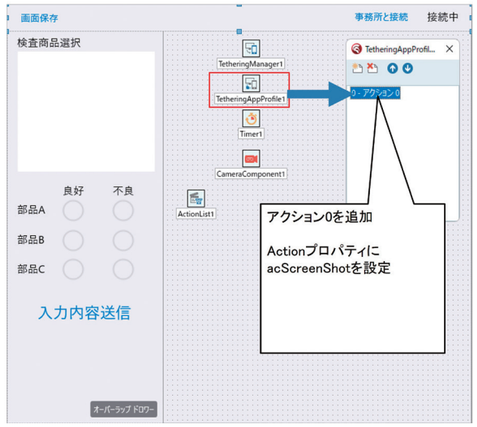
6.アクション処理の共有

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

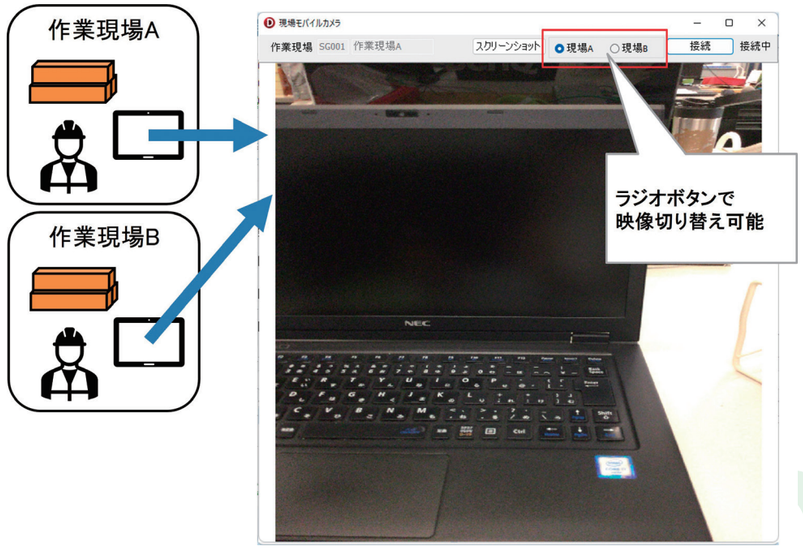
OnEndManagersDiscoveryイベント（ペアリング対象の設定）

```
{*************************************************************************
 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。