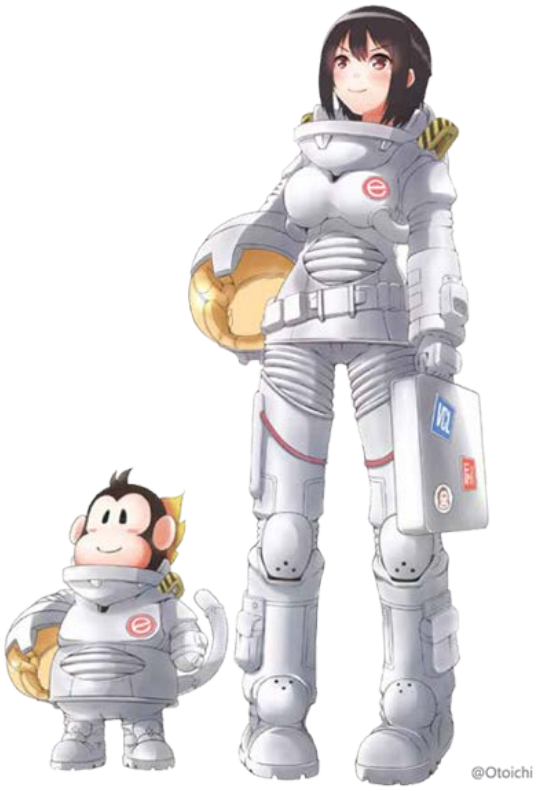
知って得する！今日から使える Delphi実践テクニック

第34回 エンバカデロ・デベロッパーキャンプ

株式会社ミガロ.
RAD事業部　技術支援課
吉原　泰介



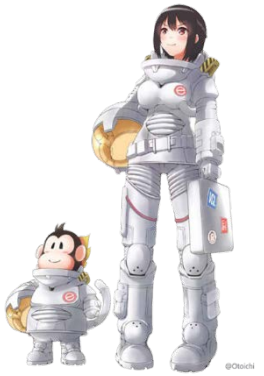
embarcadero®
DEVELOPER CAMP

▪はじめに

今回は既存プログラムに組み込んですぐに使えるプログラミングテクニックを
テーマにしています。そのため、若干VCL寄りの内容です。

MIGARO 株式会社ミガロ.

- Delphi/400 :　DelphiをIBM i (AS/400)に完全に対応させたミドルウェア
- 導入実績　:　国内約700社、全世界約6,000社



embarcadero DEVELOPER CAMP

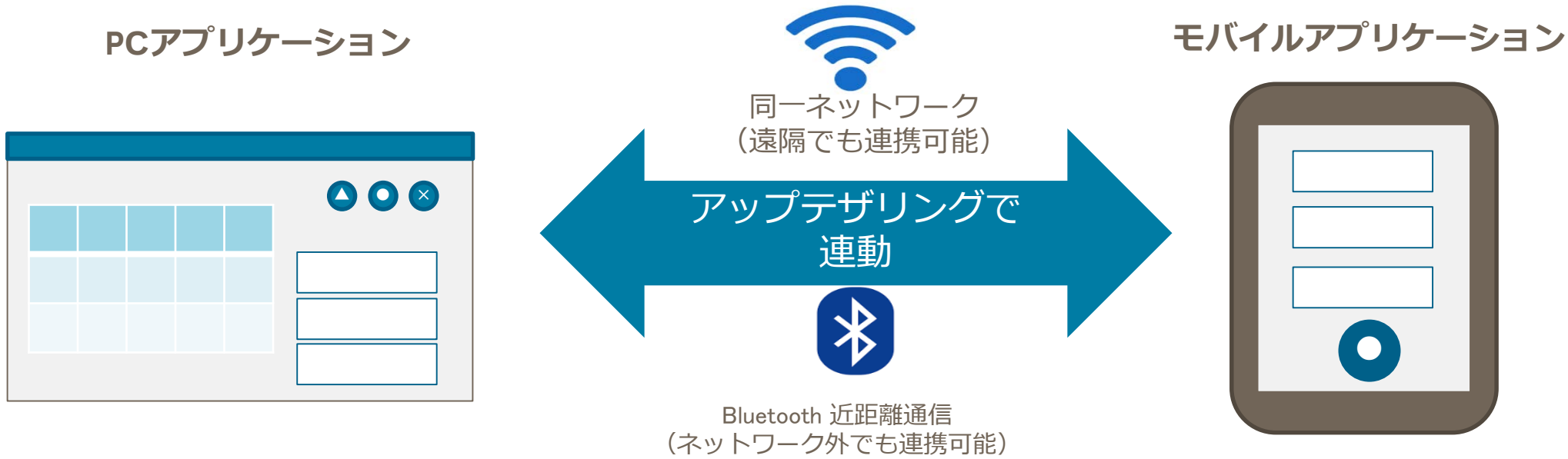
アジェンダ

- アプリケーションテザリングでモバイル端末を活用
- マルチスレッドで処理待ちの体感を改善
- DLL形式でアプリケーションを分割

1.アプリケーションテザリングでモバイル端末を活用

アプリケーションテザリングでモバイル端末を活用

- アップリケーションテザリングとは？
  同じネットワークやBluetooth上のアプリケーション間で
  データや処理を共有して連携することができるテザリング機能（以降アップテザリング）
  VCLとFireMonkey間でも連携できるので、VCL&モバイルの連携が可能！



同じネットワークに両機器がつながっているか、
または両機器がBluetoothで通信できる必要があります。

アプリケーションテザリングでモバイル端末を活用

- アップテザリング連携例
  これまで写真撮影やバーコード読取を行う業務では専用機器を用意して連携することが多かった。

商品写真を登録する

デジタルカメラで撮影して、SDカードをPCにアップして、
ようやくサーバへ登録・更新

バーコードを読み取って登録する

専用のバーコードリーダーやPOSを用意して、
PCのアプリケーションから登録・更新

アップテザリングで代用！

写真撮影やバーコード読取結果を
スマートフォンアプリからPCアプリに
送信して登録・更新
（アプリで自動化が可能）

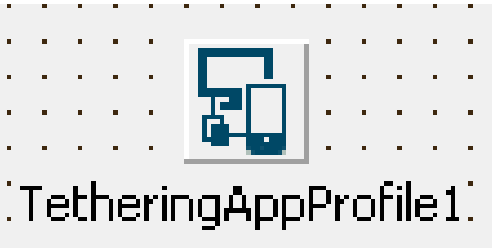
アプリケーションテザリングでモバイル端末を活用

- アップテザリング用のコンポーネント
  アップテザリングを使用する場合には、通信をする両方のアプリケーションに
  TTetheringMangerとTTetheringAppProfileコンポーネントを配置

TTetheringManagerコンポーネント

ネットワーク上でテザリング
するための接続等の管理

TTetheringAppProfileコンポーネント

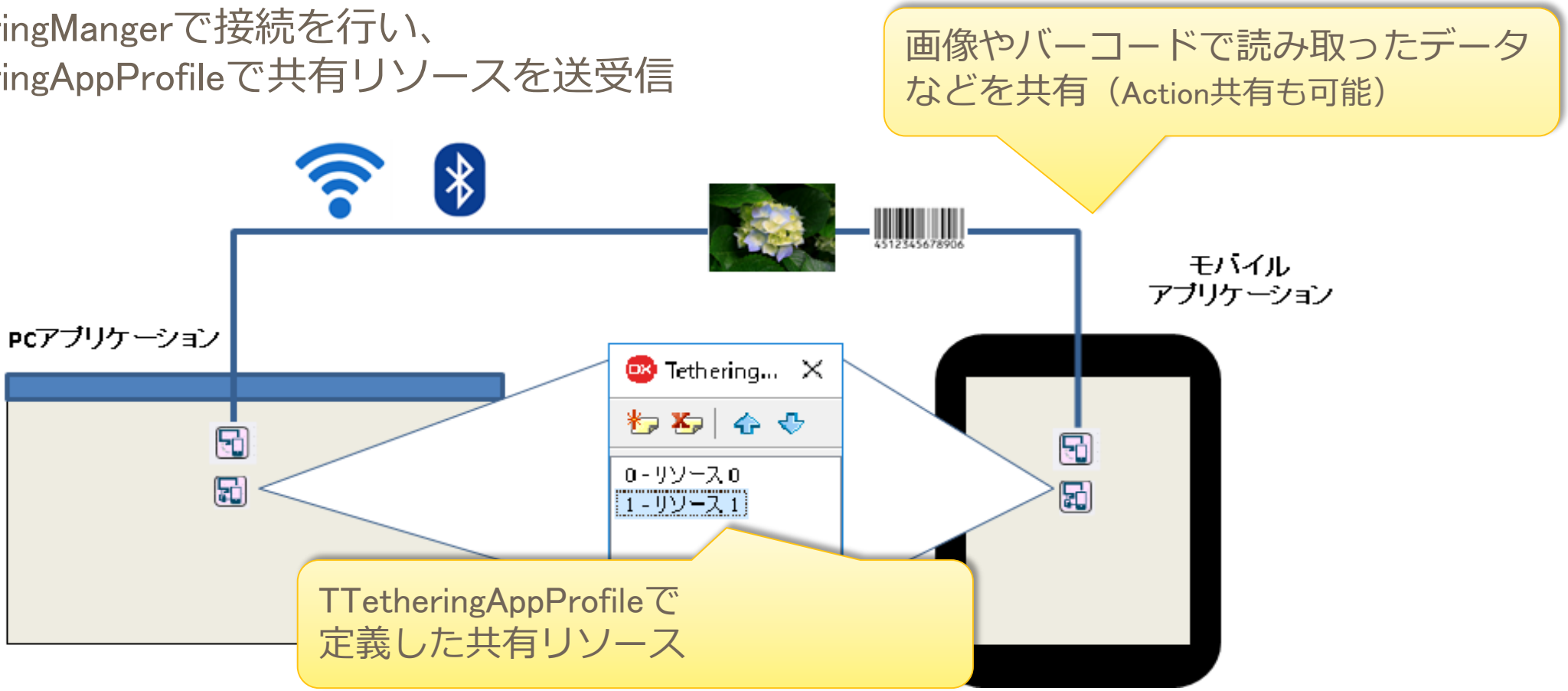


テザリングで接続した
アプリケーション間で共有する
リソースの制御

アプリケーションテザリングでモバイル端末を活用

- アップテザリング用のコンポーネントの使い方

  TTetheringMangerで接続を行い、
  TTetheringAppProfileで共有リソースを送受信



アプリケーションテザリングでモバイル端末を活用

- PCとスマートフォンのアプリケーション連携例
  拡張するアプリケーション

PCアプリケーション

モバイルアプリケーション

読取バーコード
送信

撮影写真
送信

アプリケーションテザリングでモバイル端末を活用

- PCとスマートフォンのアプリケーション連携例
  iPhoneアプリ側画面設計

TTetheringManger
（接続管理用）

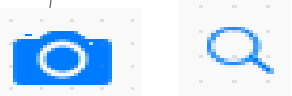
TTetheringAppProfile
（リソース共有用）

TActionList
（写真撮影用）

TMSFMXZBarReader
（バーコード読取用）
TMSのフリーコンポーネント

TImage
（撮影写真表示用）

TButton × 2　下に配置
（撮影＆バーコード読取用）



アプリケーションテザリングでモバイル端末を活用

- バーコード読み取り機能の実装に便利なコンポーネント

TMSSoftWare社のバーコード読み取りコンポーネント（無償）
【ZBarSDK】　※iOS専用
http://www.tmssoftware.com/site/freetools.asp#TTMSFMXZBarReader

ただしZBarSDKコンポーネントはiOS専用です。
Androidで使用する場合は、これをカスタマイズした
フリーソースとして公開されているTKRBarCodeSannerコンポーネン
便利です。
http://www.file-upload.net/download-8601754/TKRBarCodeSanner.zip.html

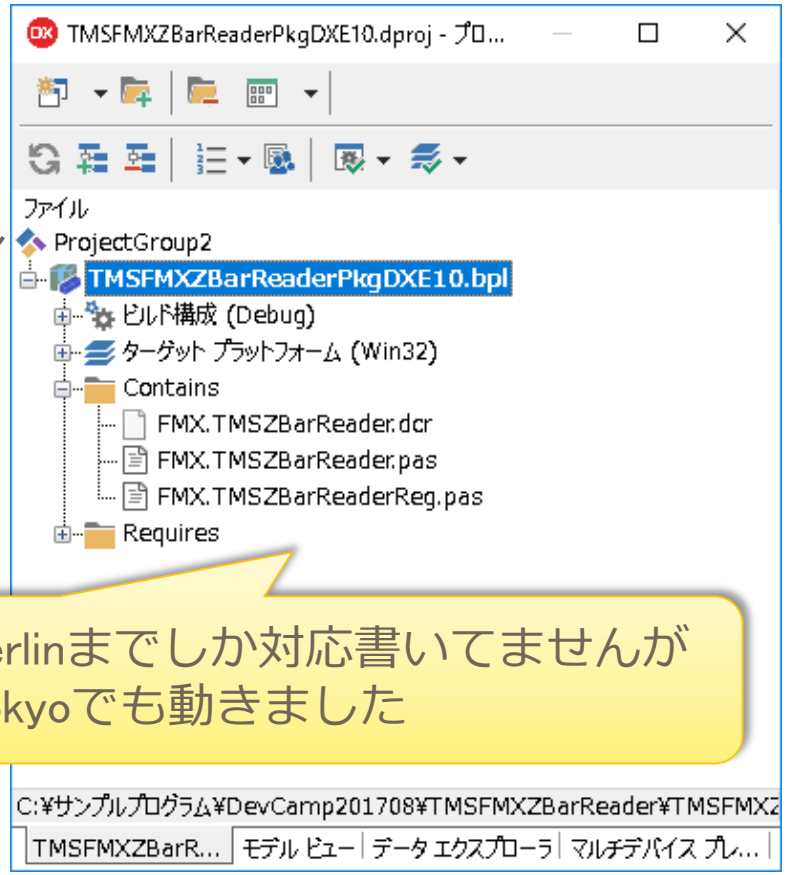
使い方はZbarSDKとほぼ同じです。
XE5当時に作られているものなのでXE7以降ではソースの修正が必要です。

（修正例）

```
interface

uses
  System.Classes
  {$IFDEF IOS}
  ,FMX.TMSZBarReader
  {$ENDIF}
  {$IFDEF ANDROID}
  ,FMX.Platform, FMX.Helpers.Android, System.Rtti, FMX.Types, System.SysUtils,
  Androidapi.JNI.GraphicsContentViewText, Androidapi.JNI.JavaTypes,
  FMX.StdCtrls, FMX.Edit, Androidapi.Helpers  // Androidapi.Helpers 追加
  {$ENDIF}
  ;
```

```
{$IFDEF ANDROID}
function TTKRBarCodeScanner.HandleAppEvent(AAppEvent: TApplicationEvent;
  AContext: TObject): Boolean;
var
  aeBecameActive : TApplicationEvent;  //----- 追加
begin
  aeBecameActive := TApplicationEvent.BecameActive;  //----- 追加
```



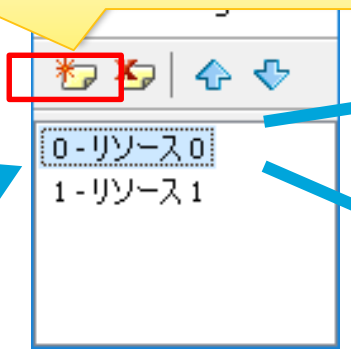
Berlinまでしか対応書いてませんが
Tokyoでも動きました

アプリケーションテザリングでモバイル端末を活用

- iPhoneアプリ側開発手順①
  TTetheringAppProfileの設定

通信グループ名

ダブルクリック

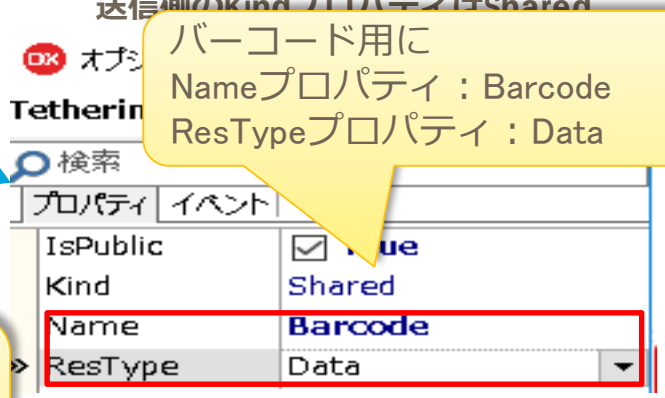


リソースを2つ追加

ResTypeプロパティはDataかStreamを選択
Data：文字などの送信
Stream：画像などの送信

ポイント！

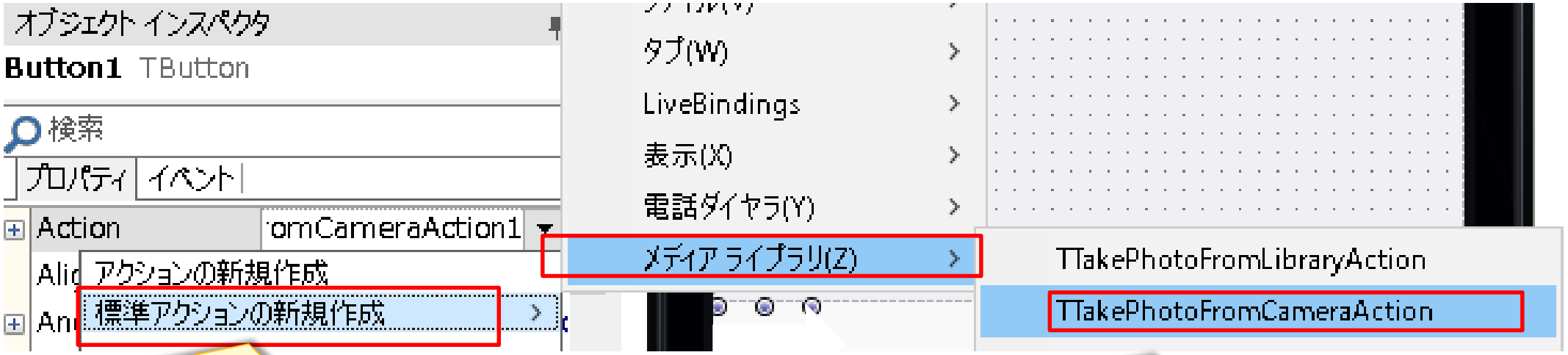
写真用に
Nameプロパティ：Camera
ResTypeプロパティ：Stream



バーコード用に
Nameプロパティ：Barcode
ResTypeプロパティ：Data

アプリケーションテザリングでモバイル端末を活用

- iPhoneアプリ側開発手順②
  Actionの設定（写真撮影ボタン）



ButtonのActionプロパティで
標準アクションの新規追加

TTakePhotoFromCameraAction
を選択

アプリケーションテザリングでモバイル端末を活用

- iPhoneアプリ側開発手順③
  Actionのイベントにプログラムを実装（写真撮影ボタン）

OnDidFinishTakingイベントを作成

OnDidFinishTakingイベント（撮影写真を送信）

```
procedure TForm1.TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
var
  FStream: TMemoryStream;
begin
  FStream := TMemoryStream.Create;  //写真用のStreamを作成
  image.SaveToStream(FStream);      //撮影写真をStreamに格納
  TetheringAppProfile1.Resources.Items[0].Value := FStream; //共有リソースに送信
  Image1.Bitmap.Assign(Image);      //画面に写真を表示
end;
```

アプリケーションテザリングでモバイル端末を活用

- iPhoneアプリ側開発手順④

バーコード撮影用のイベントにプログラムを実装（バーコード撮影ボタン）

OnClickイベント（バーコード撮影起動）

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  TMSFMXZBarReader1.Show; //バーコード撮影を起動
end;
```

バーコード撮影用のイベントにプログラムを実装（TMSFMXZBarReader）

OnGetResultイベント（取得バーコード送信）

```
procedure TForm1.TMSFMXZBarReader1GetResult(Sender: TObject; AResult: string);
Begin
 //読み取ったバーコード値を共有リソースに送信
  TetheringAppProfile1.Resources.Items[1].Value := AResult;
end;
```

アプリケーションテザリングでモバイル端末を活用

- iPhoneアプリ側開発手順⑤

画面起動時のイベントにプログラムを実装

OnCreateイベント（テザリングで接続）

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //起動時にテザリング接続を行う
  TetheringManager1.AutoConnect();
end;
```

アプリケーションテザリングでモバイル端末を活用

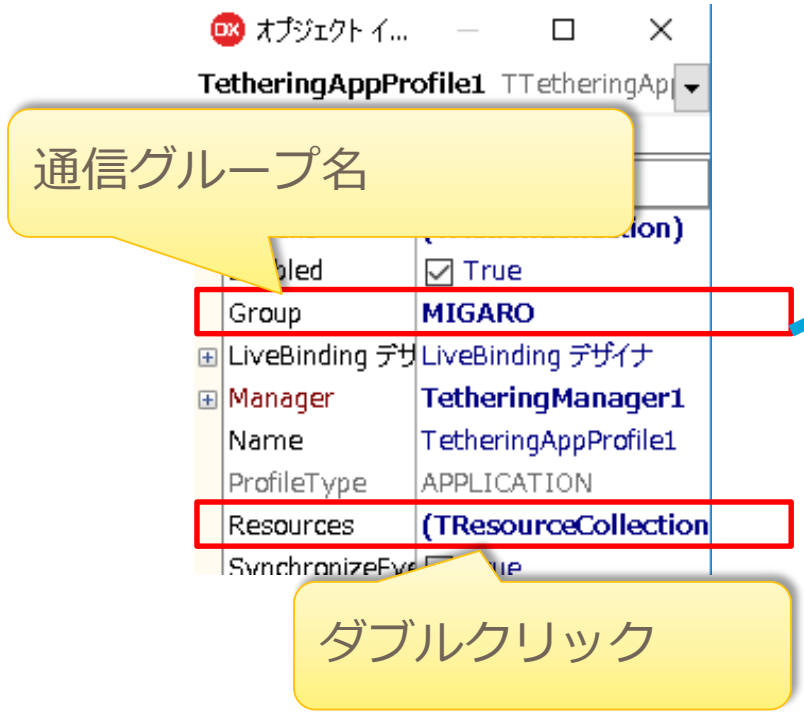
- PCとスマートフォンのアプリケーション連携例
  PCアプリ側画面設計

TTetheringManger
（接続管理用）

TTetheringAppProfile
（リソース共有用）

アプリケーションテザリングでモバイル端末を活用

- PCアプリ側開発手順①
  TTetheringAppProfileの設定
  （スマートフォン側と設定を合わせる）



アプリケーションテザリングでモバイル端末を活用

- PCアプリ側開発手順②
  写真撮影リソースのイベントにプログラム実装

OnResourceReceivedイベント（撮影写真を受信）

```
procedure TForm1.TetheringAppProfile1ResourcesOResourceReceived
  (const Sender: TObject; const AResource: TRemoteResource);
begin
  AResource.Value.AsStream.Position := 0;                    //Streamのポジション
  Image1.Bitmap.LoadFromStream(AResource.Value.AsStream);    //画面に受信画像を設定
  Image1.Repaint;                                            //再描画
end;
```

アプリケーションテザリングでモバイル端末を活用

- PCアプリ側開発手順③
  バーコードリソースのイベントにプログラム実装

OnResourceReceivedイベント（取得バーコードを受信）

```
procedure TForm1.TetheringAppProfile1Resources1ResourceReceived
  (const Sender: TObject; const AResource: TRemoteResource);
begin
  Edit1.Text := AResource.Value.AsString;  //画面に受信値を設定
end;
```

アプリケーションテザリングでモバイル端末を活用

- PCとスマートフォンのアプリケーション連携拡張例
  それぞれコンパイルを行い、アプリケーション連携が完成！簡単に拡張が可能です。

PCアプリケーション

モバイルアプリケーション

こうしたアプリ連携はAndroidはもちろん、スマートデバイス間、
PC間のアプリケーションでも活用することができます。

アプリケーションテザリングでモバイル端末を活用

- 接続状況を表示する（補足）

TLabel　（接続表示用）
TTimer　（監視用）

OnTimerイベント（接続表示）

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Label1.Visible := (TetheringManager1.RemoteProfiles.Count > 0);
end;
```

接続カウントがあれば
Labelを表示

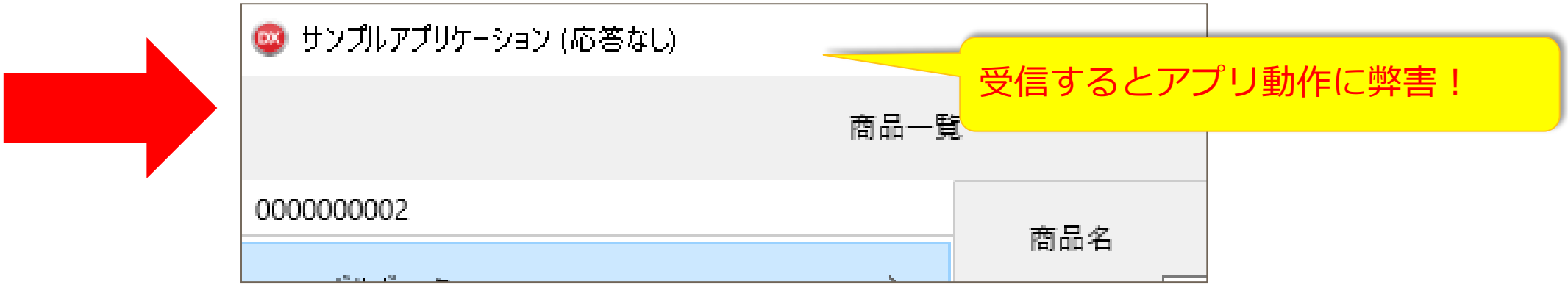
アプリケーションテザリングでモバイル端末を活用

- 受信処理が重いとどうなってしまうか？（補足）

OnResourceReceivedイベント（撮影写真を受信）

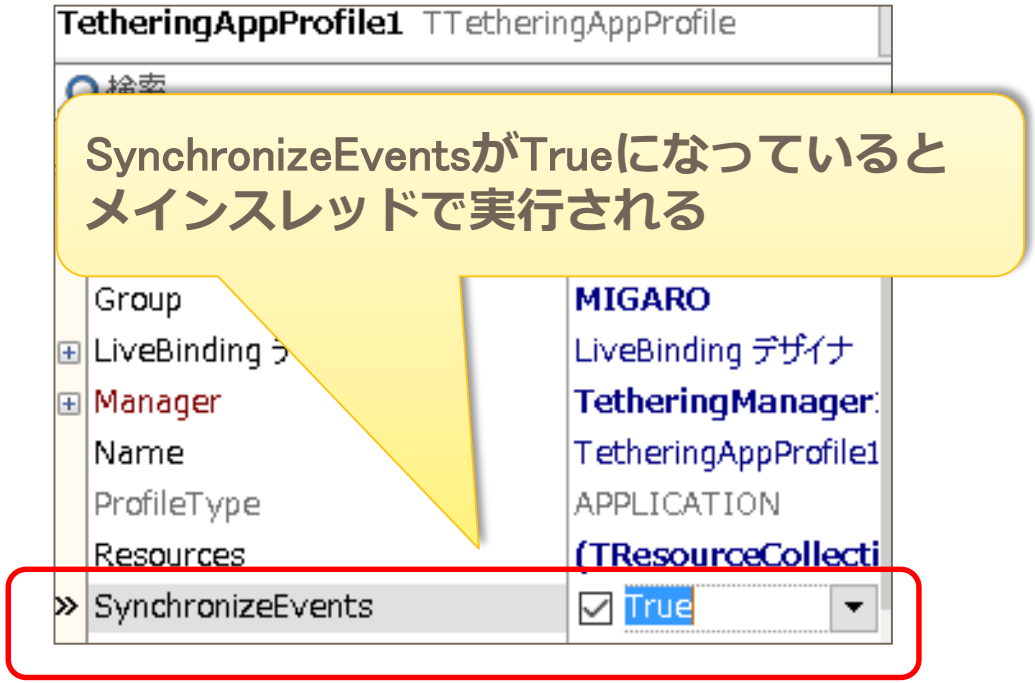
```
procedure TForm1.TetheringAppProfile1ResourcesOResourceReceived
  (const Sender: TObject; const AResource: TRemoteResource);
begin
  Sleep(10000);
end;
```

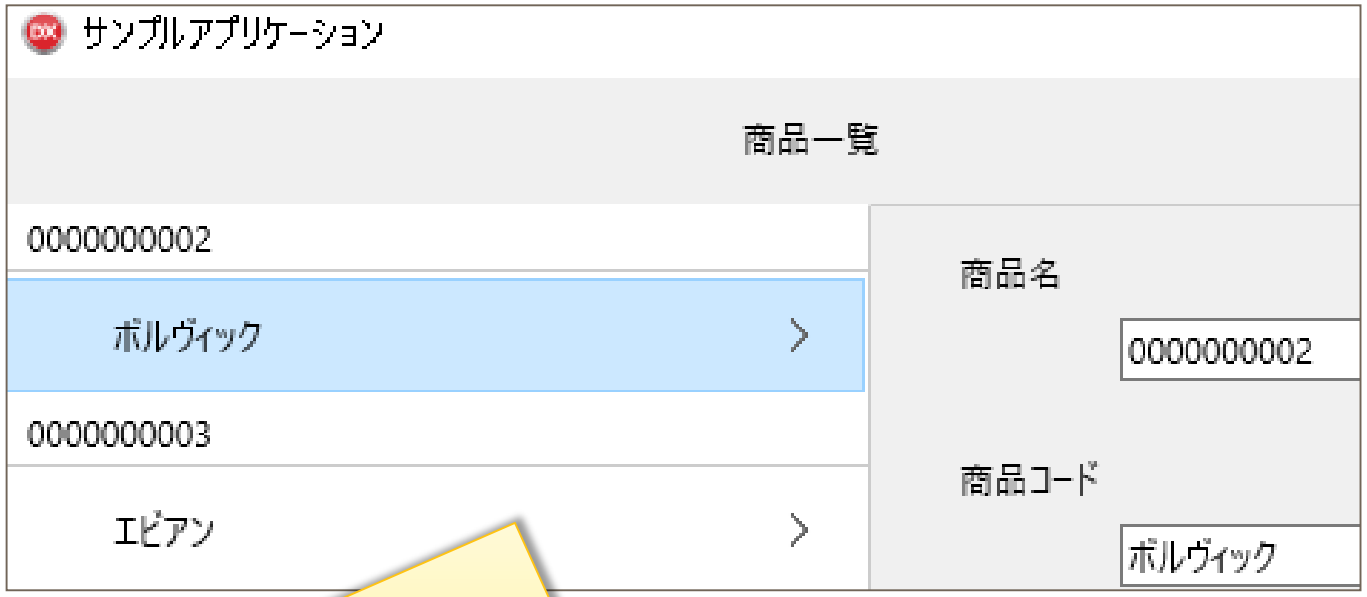
受信処理がやたら重たい場合



アプリケーションテザリングでモバイル端末を活用

- 別スレッドで実行する（補足）





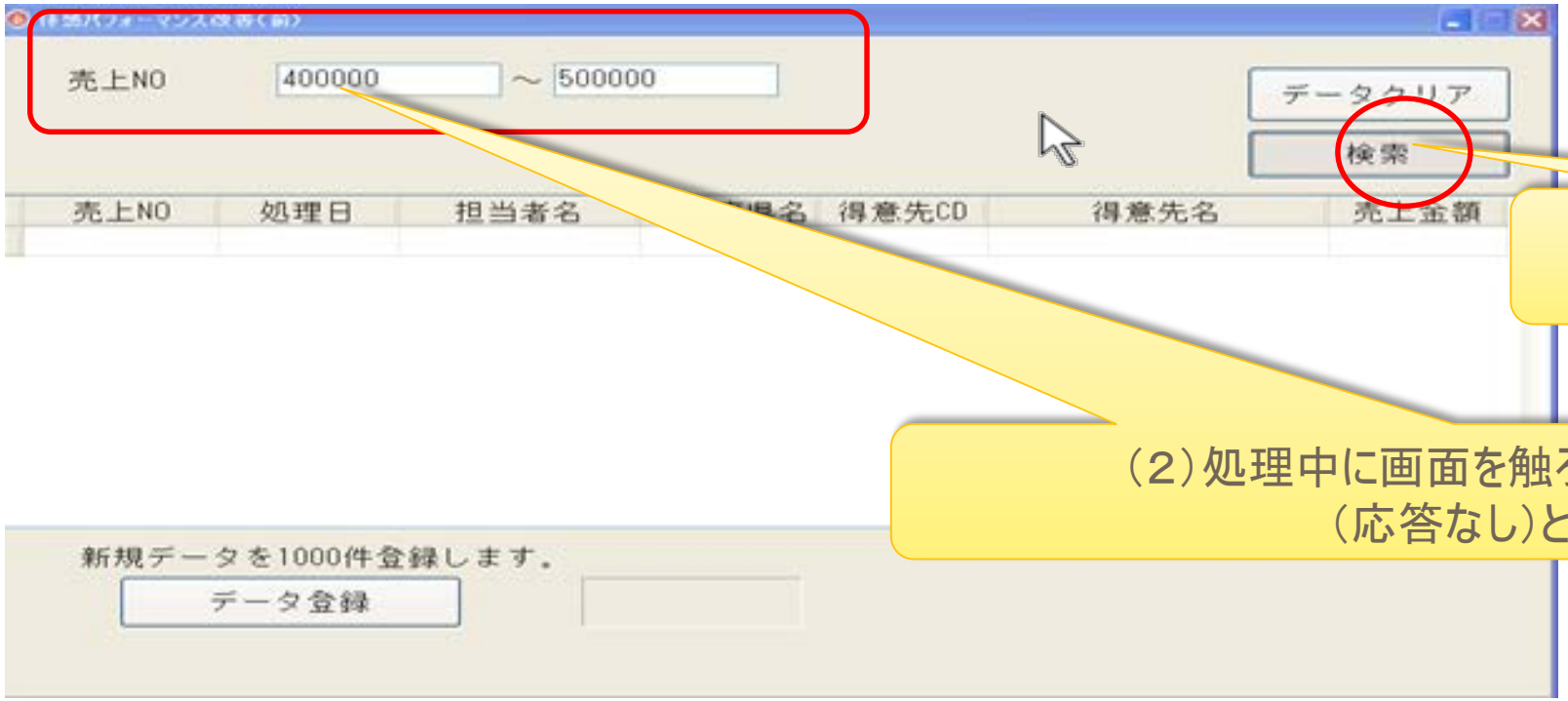
Falseにしておけば別スレッドになり、メインスレッドは止まらずに使えて便利！
10.2 Tokyo以前は、自分でスレッドを考慮する必要がある

2.マルチスレッドで処理待ちの体感を改善

（今回の内容はWindows用です）

マルチスレッドで処理待ちの体感を改善

- パフォーマンスが悪いとせっかくのアプリも評価されにくい…
  [実行]ボタンを押したとき、画面の応答がなくなると、イライラする。
  （一般的にストレスを感じない応答時間は、約3秒！）

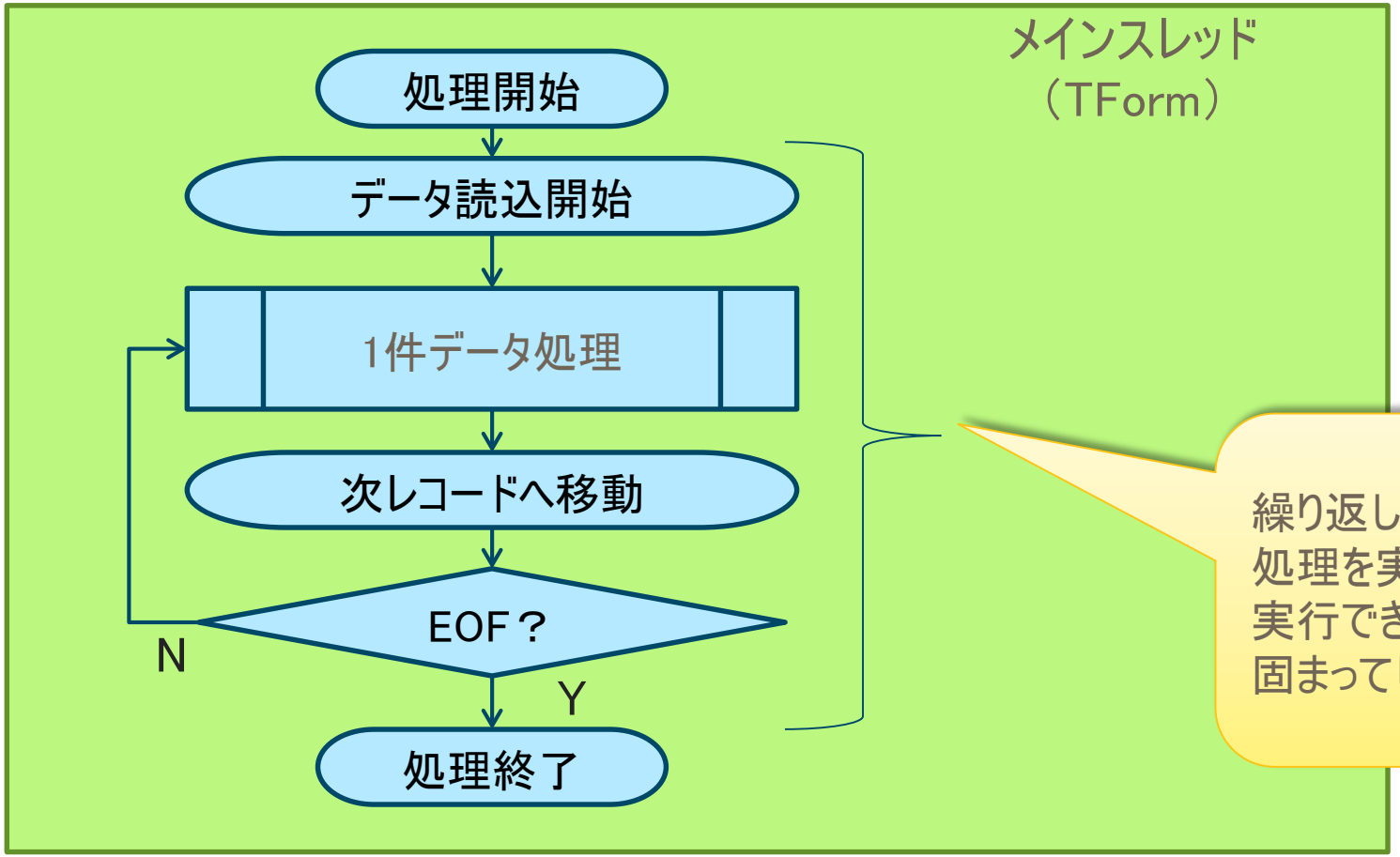


（1）ボタンを押下したか、
していないかが分からない。

（2）処理中に画面を触ろうとすると、反応がなく、
（応答なし）と表示される。

マルチスレッドで処理待ちの体感を改善

- 通常のアプリケーションは、シングルスレッド（逐次実行）である。



繰り返し処理等、時間がかかる
処理を実行すると、他の処理が
実行できないため、画面が
固まってしまう。

マルチスレッドで処理待ちの体感を改善

シングルスレッド プログラム実装例

```
procedure TForm1.btnGetDataClick(Sender: TObject);
var
  i, iRow: Integer;
begin
  iRow := 0;
  SQLQuery1.Active := True;
  try
    //繰り返し
    while (not SQLQuery1.Eof) do
    begin
      Inc(iRow);  //カウントアップ
      StringGrid1.RowCount := iRow + 1;
      for i := 0 to SQLQuery1.FieldCount - 1 do
        StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
      SQLQuery1.Next;
    end;
  finally
    SQLQuery1.Active := False;
  end;
end;
```

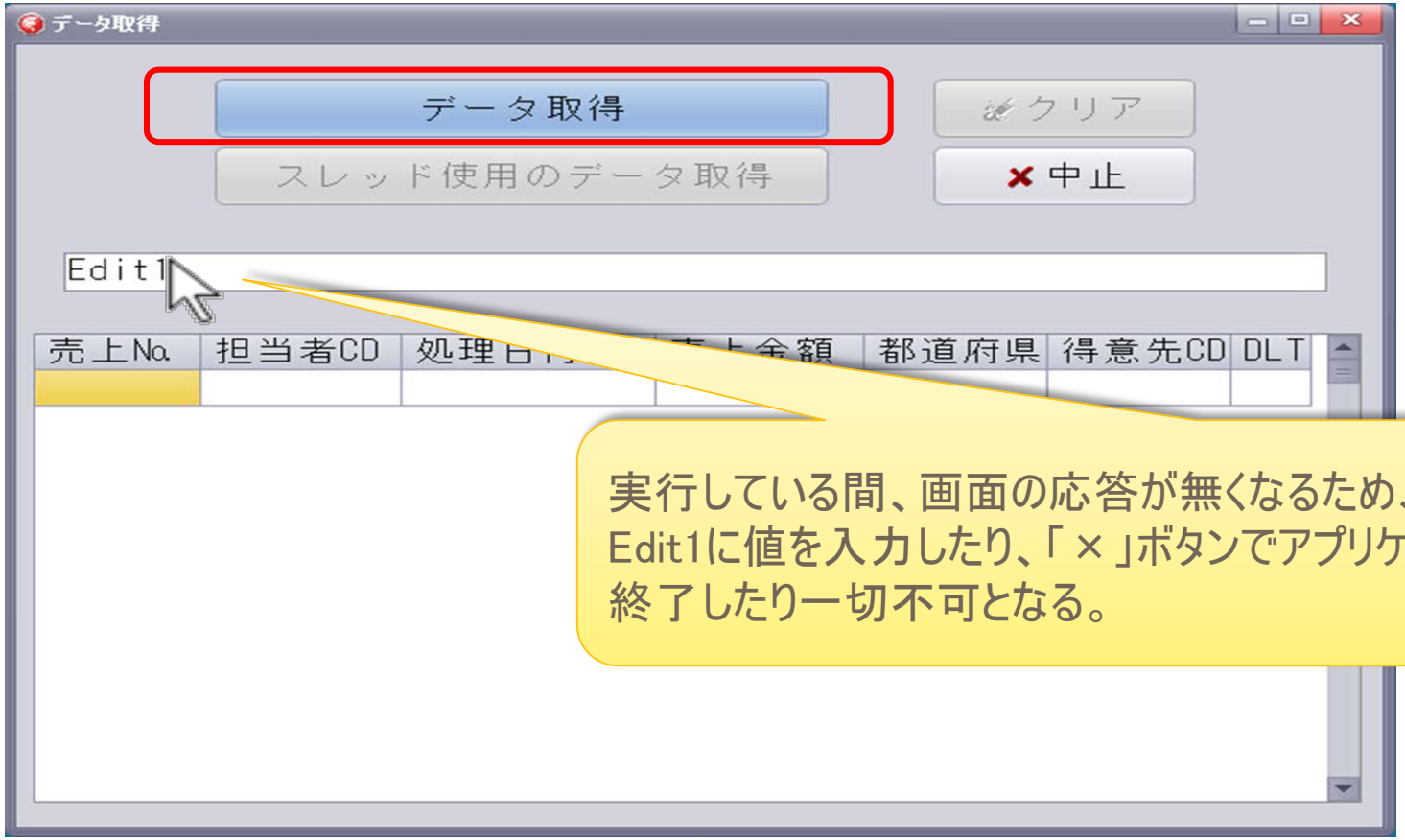
繰り返し処理

StringGirdの行数を追加

各フィールドの値を
順番にStringGridに書き出し

マルチスレッドで処理待ちの体感を改善

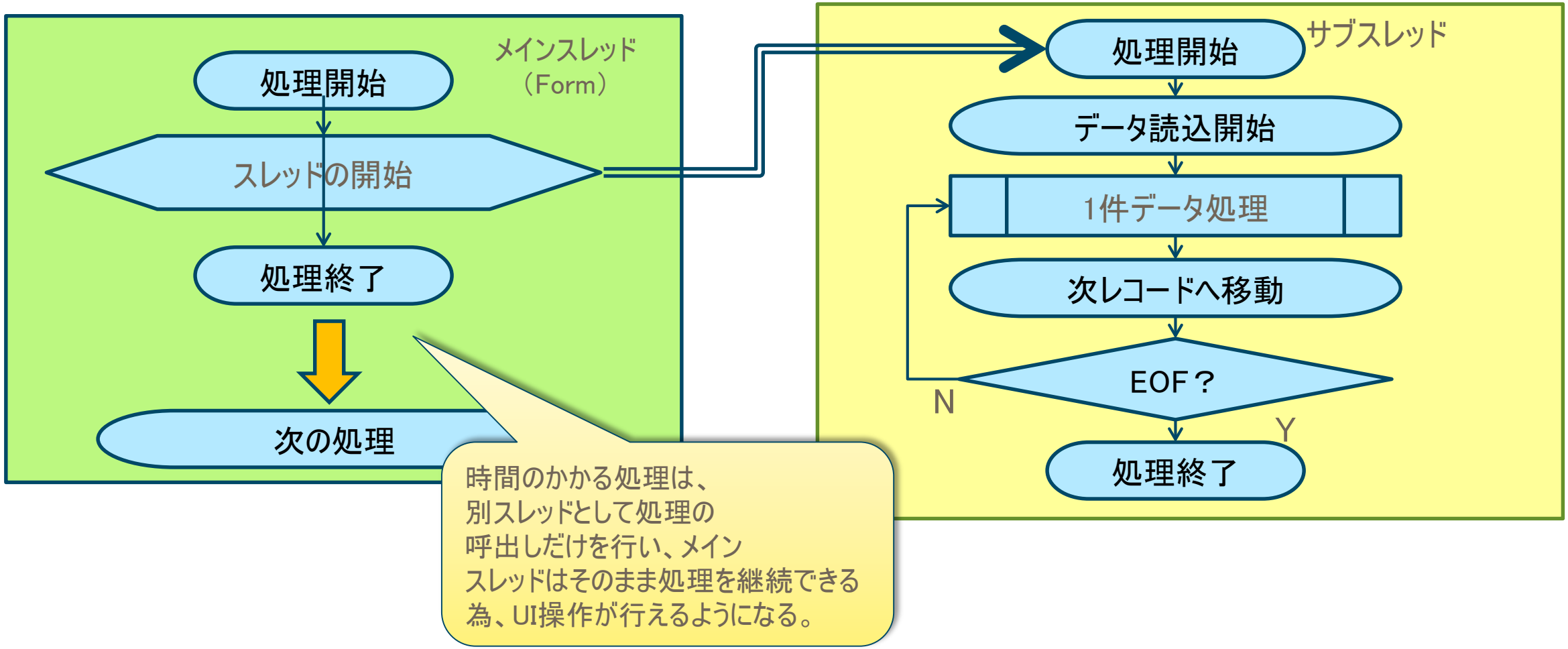
- シングルスレッド実行例



実行している間、画面の応答が無くなるため、
Edit1に値を入力したり、「×」ボタンでアプリケーションを
終了したり一切不可となる。

マルチスレッドで処理待ちの体感を改善

- マルチスレッドを使えば**レスポンスタイム（応答時間）**が向上
  時間のかかる処理をサブスレッドとすることで、メインスレッド（画面）は別の処理が実行可能



マルチスレッドで処理待ちの体感を改善

- TThread　クラスを使用して、別スレッドを記述する方法
  [ファイル]→[新規作成]→[その他]
  　新規作成ダイアログ：[Delphiファイル]→[スレッドオブジェクト]

メインスレッド

```
procedure TfrmMain.Buttton1Click(Sender: TObject);
begin
    //登録処理のスレッドを生成する
    TDataEntryThread.Create(受け渡しパラメータ);
end;
```

サブスレッド

```
type
  //データ登録用スレッド
  TDataEntryThread = class(TThread)
  private

    ((スレッド内で使用する変数や手続きを宣言))

  protected
    procedure Execute; override;
  public
    constructor Create(パラメータリスト); virtual;
  end;
```

スレッドクラスを別に定義する為、メインスレッド上では、スレッド内で
どのような処理が行われているか、一目では分かりづらい。

マルチスレッドで処理待ちの体感を改善

- CreateAnonymousThread を使ったスレッド処理

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //ボタンクリックの処理
      …
  //スレッド処理
  TThread.CreateAnonymousThread(
  procedure()
  begin
    //重たい処理
    Sleep(10000);

    Edit1.Text := '処理終了';

  end).Start;
end;
```

メインスレッド

名前の無いサブルーチン ： **無名メソッド** として定義

サブスレッド

シングルスレッド同様一つのサブルーチンで処理が記述可能！

マルチスレッドで処理待ちの体感を改善

マルチスレッドプログラム実装例

```
procedure TForm1.btnThreadGetDataClick(Sender: TObject);
begin
  TThread.CreateAnonymousThread(
  procedure()
  var
    i, iRow: Integer;
  begin
    iRow := 0;
    SQLQuery1.Active := True;
    try
      //繰り返し
      while (not SQLQuery1.Eof) do
      begin
        Inc(iRow); //カウントアップ
        StringGrid1.RowCount := iRow + 1;
        for i := 0 to SQLQuery1.FieldCount - 1 do
          StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
        SQLQuery1.Next;
      end;
    finally
      SQLQuery1.Active := False;
    end;
  end).Start;
end;
```

スレッドの生成

シングルスレッドプログラム
と同じコード

スレッドの開始

マルチスレッドで処理待ちの体感を改善

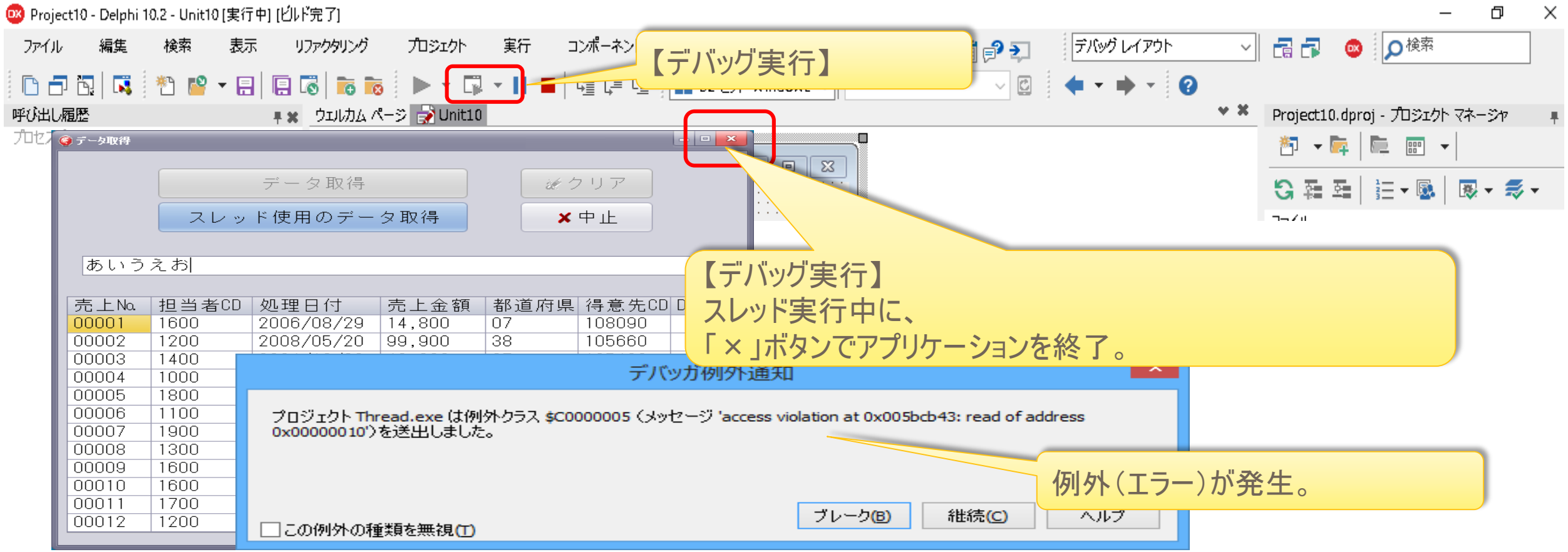
- マルチスレッド実行例

実行後、画面制御がすぐに戻る為、
Edit1への値の入力や
StringGridの内容が即座に確認可能！

レスポンスの体感が向上！

マルチスレッドで処理待ちの体感を改善

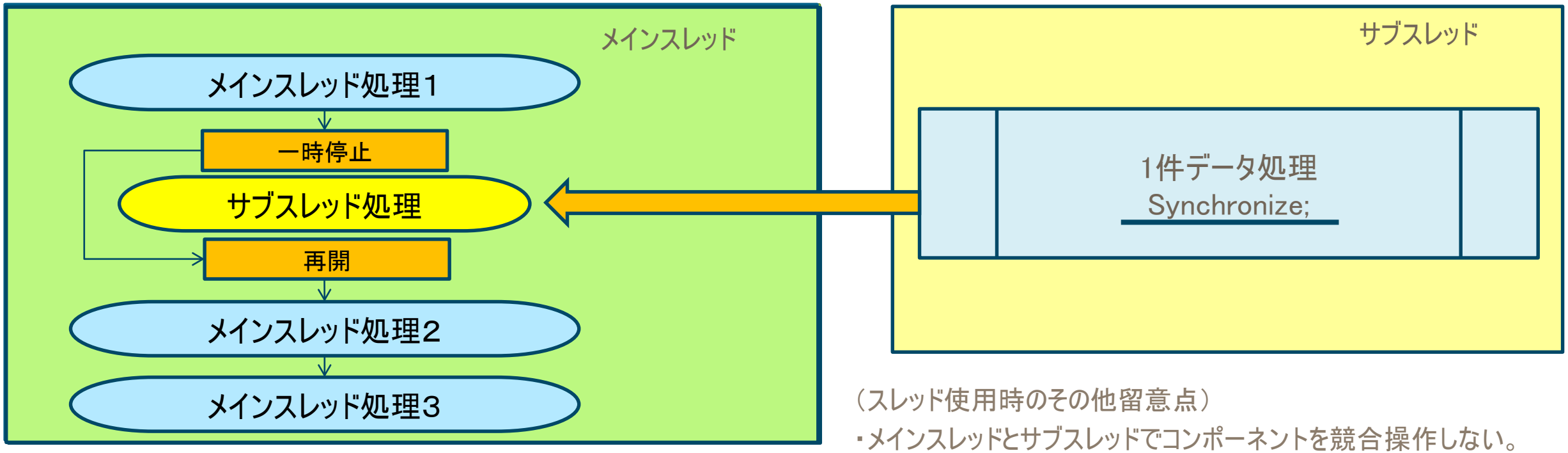
- マルチスレッドの注意点



マルチスレッドで処理待ちの体感を改善

- マルチスレッドの注意点

ビジュアルコンポーネントを使用するのは、基本メインスレッドのみである。
サブスレッド側でビジュアルコンポーネントを操作したい場合、Synchronizeメソッドを使用して、メインスレッド側を一時停止し、サブスレッド側から操作を行えるようにする必要がある。



（スレッド使用時のその他留意点）
・メインスレッドとサブスレッドでコンポーネントを競合操作しない。
・Synchronize処理に時間がかかる処理をしない。

マルチスレッドで処理待ちの体感を改善

- Synchronizeを使用した操作

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //ボタンクリックの処理
      ...
  //スレッド処理
  TThread.CreateAnonymousThread(
  procedure()
  begin
    //重たい処理
    Sleep(10000);
    TThread.Synchronize(TThread.CurrentThread,
    procedure
    begin
      Edit1.Text := '処理終了';
    end);
  end).Start;
end;
```

メインスレッド

サブスレッド

メインスレッドに割り込みして、
Edit1(ビジュアルコンポーネント)を操作。

マルチスレッドで処理待ちの体感を改善

Synchronizeを使用した改良

```
procedure TForm1.btnThreadGetDataClick(Sender: TObject);
begin
  TThread.CreateAnonymousThread(
  procedure()
  var
    i, iRow: Integer;
  begin
    iRow := 0;
    SQLQuery1.Active := True;
    try
      while (not SQLQuery1.Eof) do
      begin
        Inc(iRow); //カウントアップ
        StringGrid1.RowCount := iRow + 1;
        for i := 0 to SQLQuery1.FieldCount - 1 do
          StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
        SQLQuery1.Next;
      end;
    finally
      SQLQuery1.Active := False;
    end;
  end).Start;
end;
```

処理を書き換え

```
while (not SQLQuery1.Eof) do
begin
  Inc(iRow); //カウントアップ
  //ビジュアルコンポーネントを操作
  TThread.Synchronize(TThread.CurrentThread,
  procedure
  var
    i: Integer;
  begin
    StringGrid1.RowCount := iRow + 1;
    for i := 0 to SQLQuery1.FieldCount - 1 do
      StringGrid1.Cells[i, iRow] :=
        SQLQuery1.Fields[i].Text;
  end);
  SQLQuery1.Next;
end;
```

Synchronizeの開始

ループ変数はローカルのみ

Synchronizeの終了

サブスレッドの中で
直接StringGridに対し
書き込みを実行

マルチスレッドで処理待ちの体感を改善

- 改良したマルチスレッド実行例

【デバッグ実行】
スレッド実行中に、
「×」ボタンでアプリケーションを終了
しても、エラーとならない。

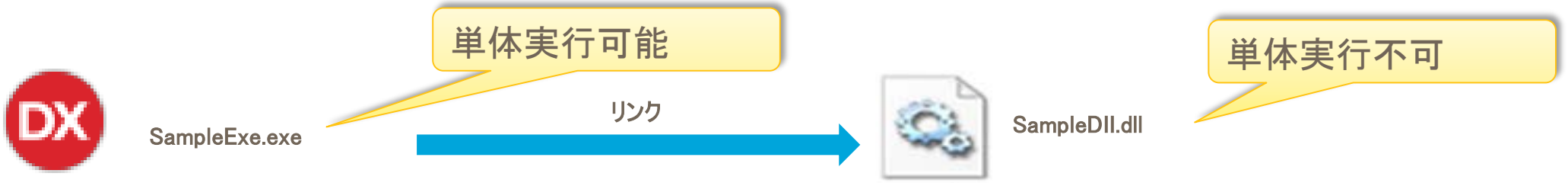
Synchronizeを使用することで、安全にスレッドを使用可能！

3.DLL形式でアプリケーションを分割

DLL形式でアプリケーションを分割

- DLLとは？

  Windowsで使用される技術の一つ。単体では実行せず、他のプログラム(EXE)から呼び出されて機能するプログラム。DLLの中にサブルーチン（手続き・関数）を定義しておき、
  EXE側からDLLをリンクすると、DLL関数を呼び出して利用できる。



DLL関数を使用

```
procedure Button1Click(Sender: TObject);
var
  C: Integer;
begin
  C := CalcAdd(3, 4, 5);
end;
```

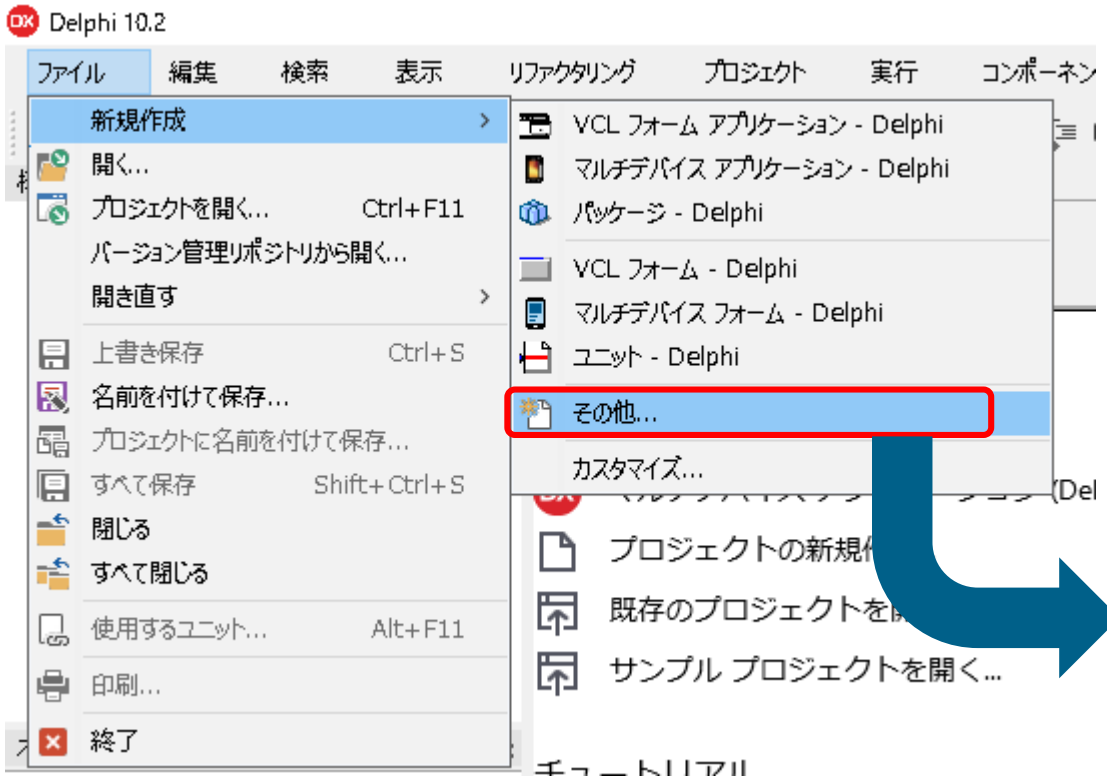
関数（サブルーチン）を定義

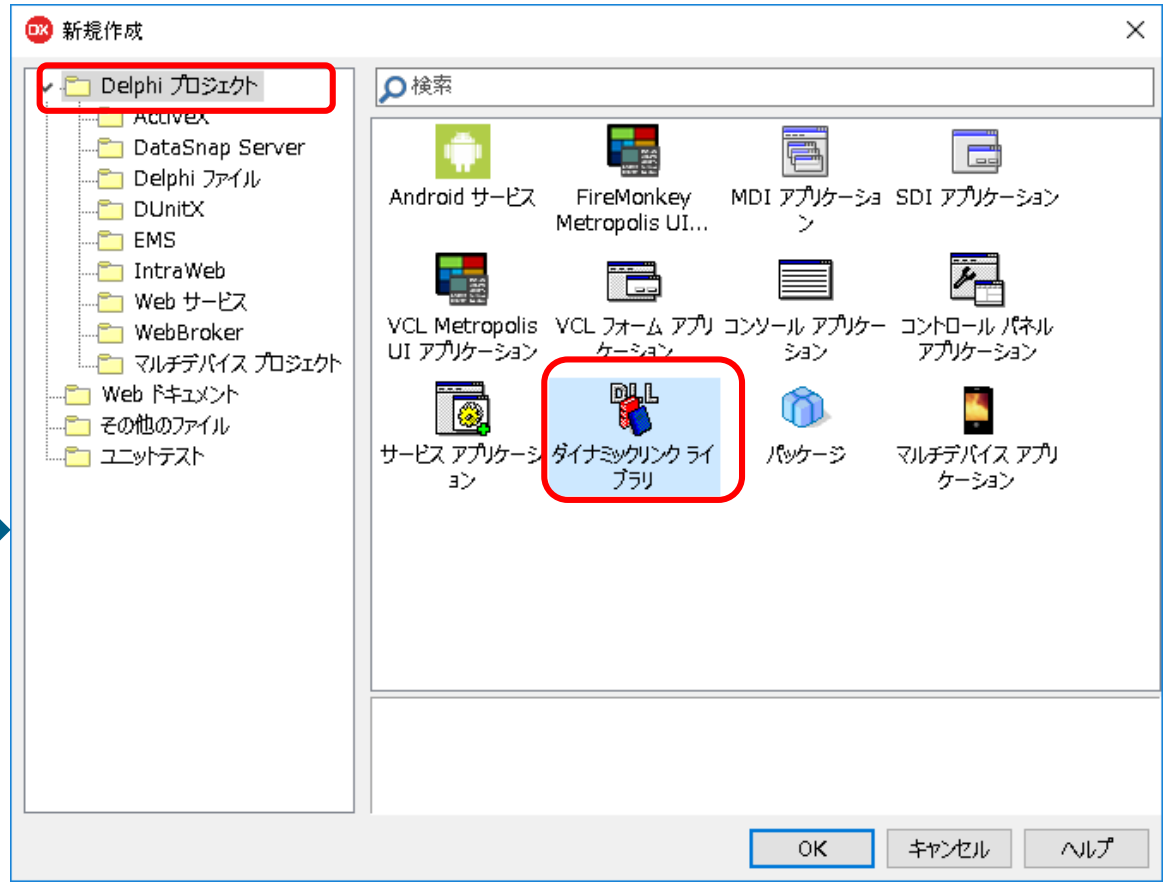
```
function  CalcAdd(A, B, C: Integer): Integer;
begin
  Result := A + B + C;
end;
```

例えばDLL側にBL(ビジネスロジック)、EXE側はUIといった分離開発にも有効！
（修正などのメンテナンスを機能で切り分けて管理できる）

DLL形式でアプリケーションを分割

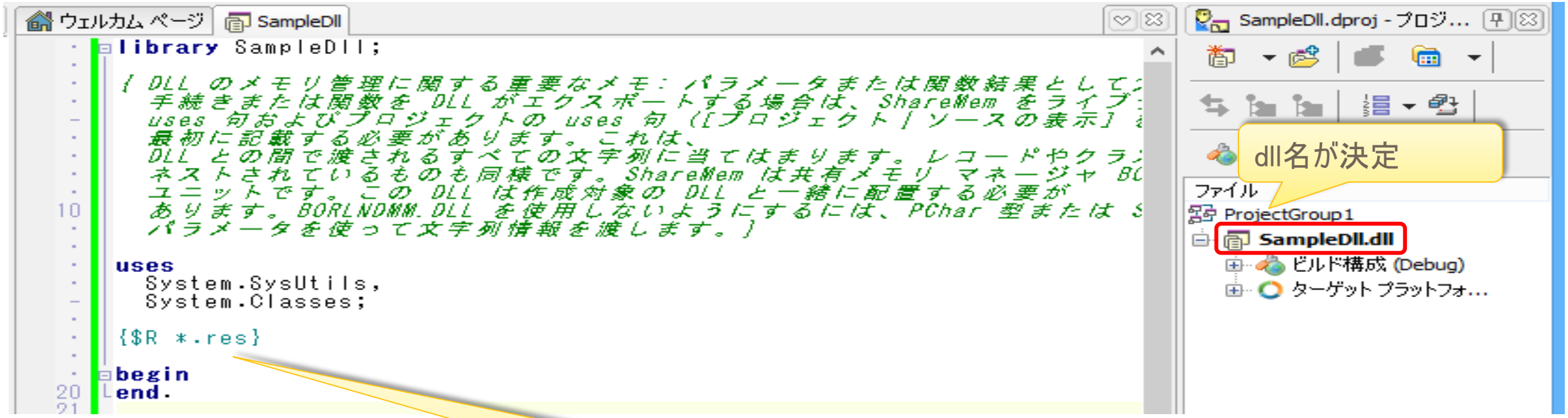
- DLLプロジェクトの新規作成
  [ファイル]→[新規作成]→[その他] より「ダイナミックリンクライブラリ」を選択





DLL形式でアプリケーションを分割

- DLLプロジェクトの作成
  [プロジェクトに名前を付けて保存]でファイルを保存



この中に、外部から呼び出される
手続き(procedure)や**関数(function)** を記述。

DLL形式でアプリケーションを分割

DLLプログラム　記述例

```
library SampleDll;
  ...
uses
  System.SysUtils,
  System.Classes;

{$R *.res}

function CalcAdd(A, B, C: Integer): Integer; stdcall;
begin
  Result := A + B + C;
end;

exports
  CalcAdd;

begin
end.
```

実行したい手続き/関数

呼出規約：stdcallを追加
（Delphi以外からdllが使用可能）

外部から呼び出したい
手続き/関数名を
exports節に追加

DLL形式でアプリケーションを分割

EXE

- VCLフォームアプリケーションよりDLL呼出し

DLL側の手続き/関数を宣言
external句 に参照するDLLを
指定

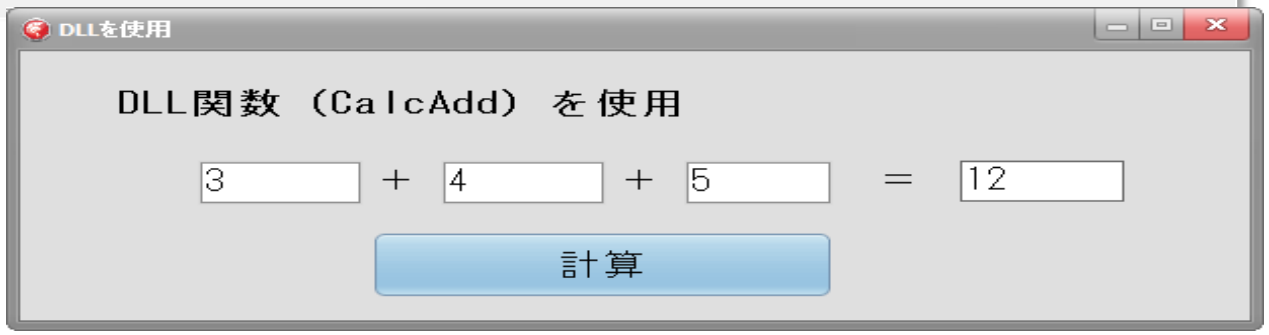
btnCalc: TButton

```
//----- DLL関数を宣言
function CalcAdd(A, B, C: Integer): Integer; stdcall; external 'SampleDll.dll';

procedure TfrmSample.btnCalcClick(Sender: TObject);
begin
  edtAns.Text := IntToStr(CalcAdd(StrToInt(edtA.Text),
                                  StrToInt(edtB.Text),
                                  StrToInt(edtC.Text)));
end;
```

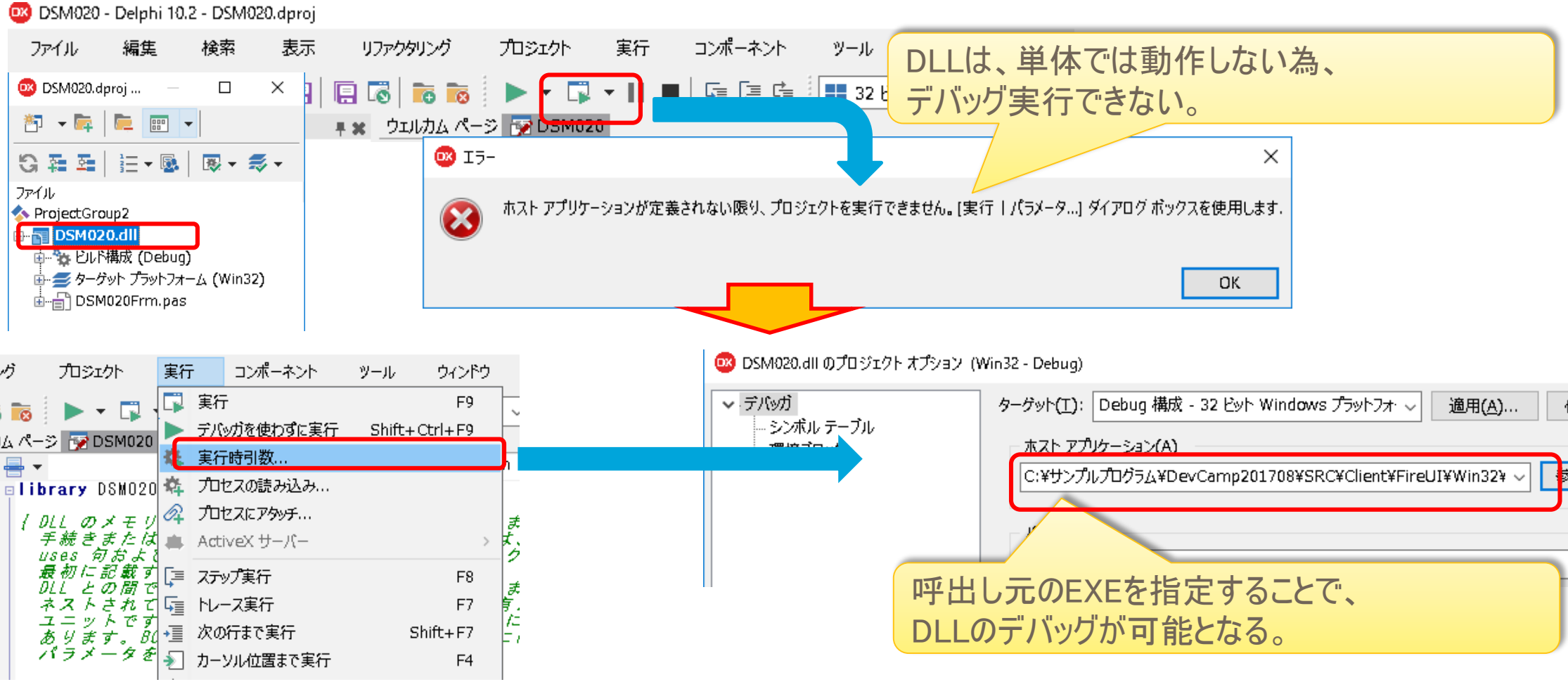
通常の手続き/関数と同様
EXE側からDLL関数が使用可能

実行



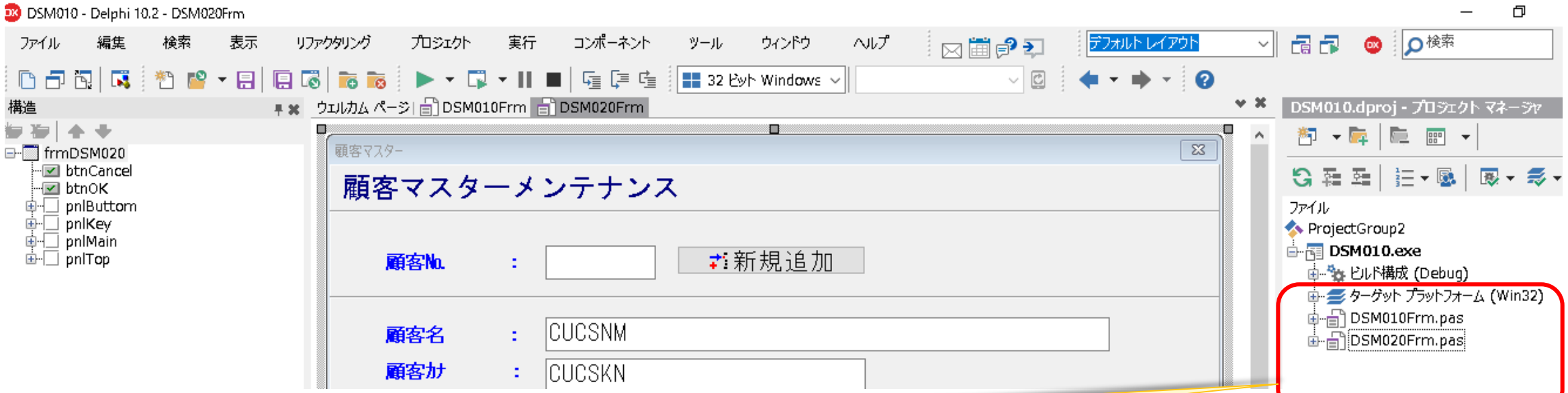
DLL形式でアプリケーションを分割

- (補足)DLLプロジェクト デバッグ方法
  [実行]→[実行時引数] より「ホストアプリケーションを指定」



DLL形式でアプリケーションを分割

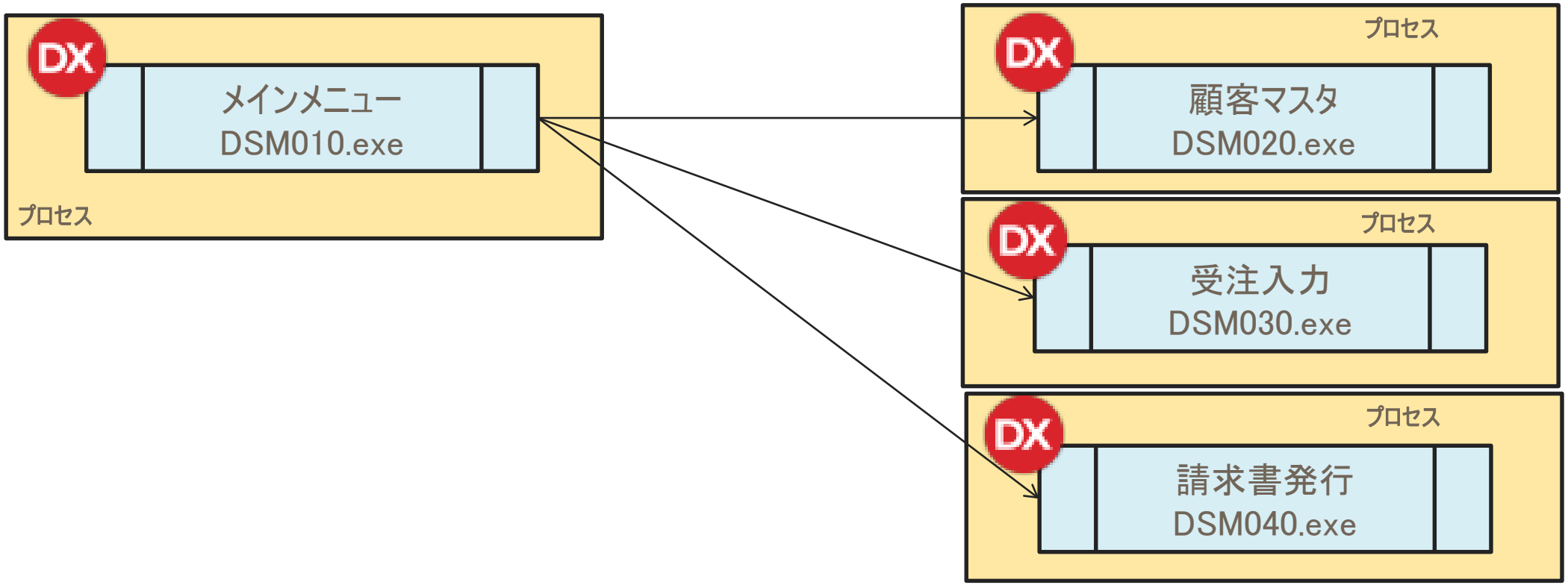
- 一つのプロジェクト(EXE)で、複数フォーム（機能）を構成する場合
  - グローバル変数等により、画面間の値の受け渡しが容易
  - EXEファイル一つでシステムが完結する
  - 画面（機能）数が多くなると、実行ファイルサイズが拡大
  - 仕様変更の都度、プロジェクト全体のEXE再配布が必要



一つのプロジェクト（EXE）に複数画面を配置。

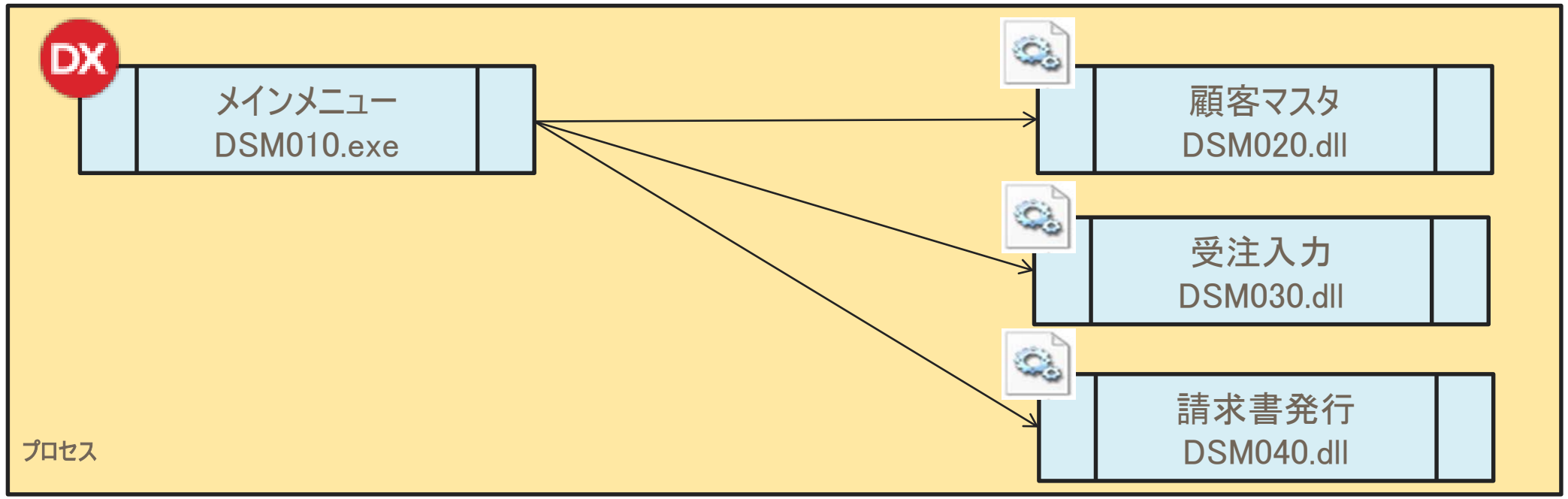
DLL形式でアプリケーションを分割

- メニュー用のEXEと各機能ごとにプロジェクト(EXE)を分割する場合
  - 機能ごとに個別開発、単体テストが行える
  - 個別機能の仕様変更が発生しても、当該EXEのみ置き換えで良い
  - 実行されるEXE分だけ、プロセスが生成され、個別データベース接続が行われる
  - EXE間の値の受け渡し方法が必要 （実行時引数など）



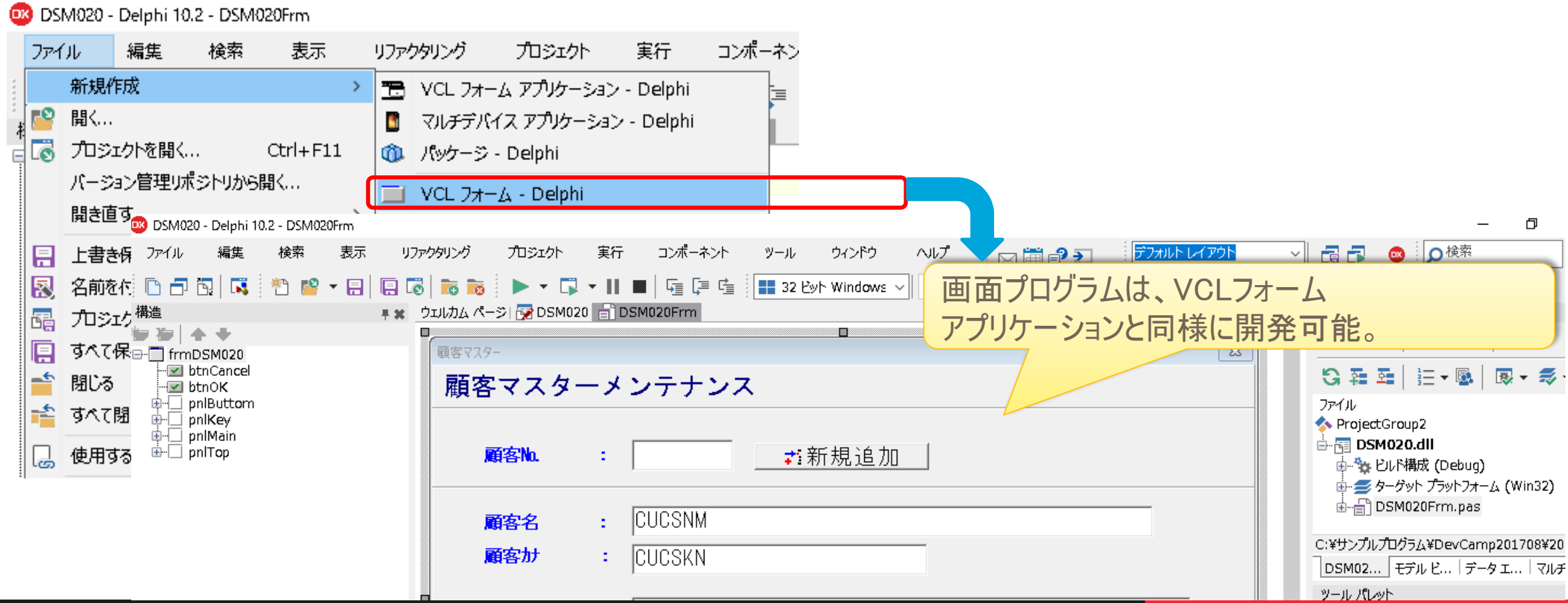
DLL形式でアプリケーションを分割

- メニュー用のEXEと各機能ごとにプロジェクト(DLL)を分割する場合
  - 機能ごとに個別開発、単体テストが行える
  - 個別機能の仕様変更が発生しても、当該DLLのみ置き換えで良い(BLとUIで分けても有効)
  - 単体EXEプロジェクト同様、実行プロセスやデータベース接続が一つとなる
  - EXE-DLL間のグローバル変数等の値の受け渡しが可能



DLL形式でアプリケーションを分割

- DLLフォームの作成
  通常のVCLフォームアプリ同様、フォームを持つDLLも作成可能。
  DLLプロジェクト作成後、[ファイル]→[新規作成]→[VCLフォーム] で作成



DLL形式でアプリケーションを分割

- DLL呼び出し部分の実装
  DLLプロジェクトには、自動生成フォームはないので
  フォームを生成して表示するDLL関数を　プロジェクトファイルに作成する

```
library DSM020;
  ...
uses
  System.SysUtils,
  System.Classes,
  Winapi.Windows,
  Vcl.Forms,
  Vcl.Controls,
  DSM020Frm in 'DSM020Frm.pas'   {frmDSM020};

{$R *.res}
```

フォーム表示処理ロジックに必要な
ユニットを追加
  Windows, Forms, Controls （XE以前）

```
function ShowDSM020Form (AppHandle: HWND): TModalResult; stdcall;
begin
  Application.Handle := AppHandle;
  try
    frmDSM020 := TfrmDSM020.Create(Application);
    try
      Result := frmDSM020.ShowModal;
    finally
      frmDSM020.Release;
    end;
  finally
    Application.Handle := 0;
  end;
end;

exports
  ShowDSM020Form;

begin
end.
```

EXEアプリのウィンドウハンドルが必要

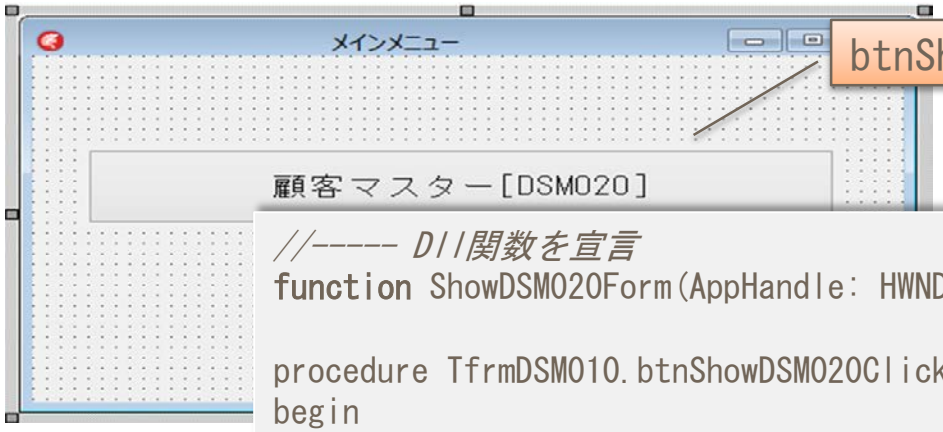
一般的なモーダルフォームの表示
と同様のロジック

処理結果(ModalResult)を呼出し元に
返却

DLL形式でアプリケーションを分割

EXE

- メインプログラムのEXEより、DLLフォームを起動



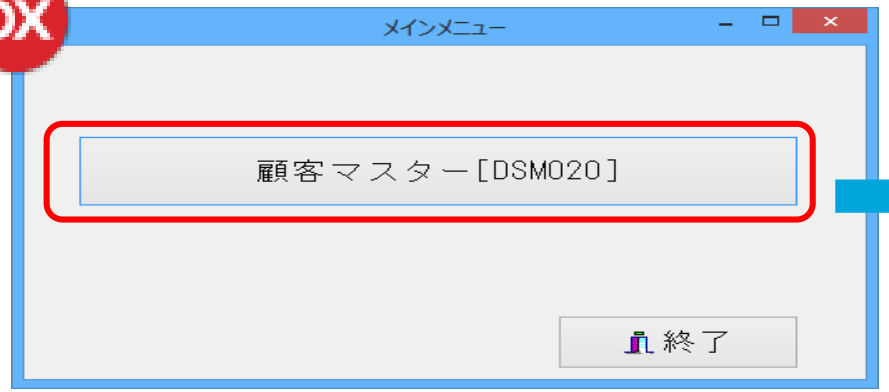
btnShowDSM020: TButton

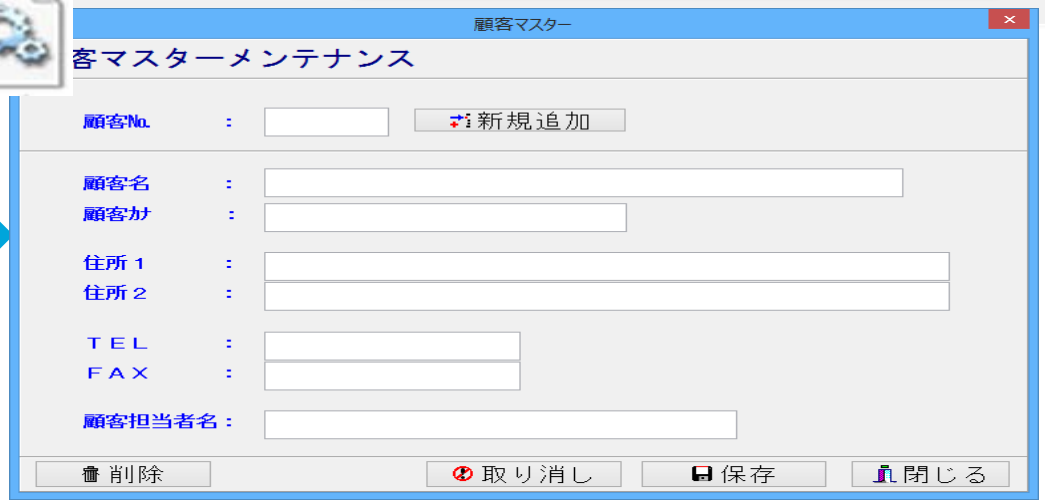
```
//----- Dll関数を宣言
function ShowDSM020Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM020.dll';

procedure TfrmDSM010.btnShowDSM020Click(Sender: TObject);
begin
  //顧客マスター呼出し
  ShowDSM020Form(Application.Handle);
end;
```

アプリケーション
メインフォームの
ウィンドウハンドルをセット

実行





DLL形式でアプリケーションを分割

- DLLが増えるごとに、DLL関数の宣言の追加が必要
  DLL関数をコード中に宣言しないと呼び出せない。

メニュープログラム

```
//----- Dll関数を宣言
function ShowDSM020Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM020.dll'; //顧客マスタ
function ShowDSM030Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM030.dll'; //受注入力
function ShowDSM040Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM040.dll'; //請求書発行
```

```
//----- Dll関数を宣言
function ShowDSM020Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM020.dll'; //顧客マスタ
function ShowDSM030Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM030.dll'; //受注入力
function ShowDSM040Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM040.dll'; //請求書発行
function ShowDSM050Form(AppHandle: HWND): TModalResult; stdcall; external 'DSM050.dll'; //入金照会
```

新しいDLL用の宣言追加が必要

EXEの置き換えが都度発生

メニュー

DSM010.exe

追加

DSM050.dll

DSM050用の宣言追加が必要、 要プログラム修正。

DLLが増えても、EXEを修正せずそのまま使用する方法はないか？

DLL形式でアプリケーションを分割

- LoadLibrary関数で、パラメータ指定したDLLを動的に読み込むことが可能
  フォームを生成して表示するDLL関数は、全て同じ関数名とする。

動的なDLLの読み込み実装

```
function TfrmDSM010.ShowForm(ADllName: String): TModalResult;
var
  hDll: Integer;
  ShowDllForm: function(AppHandle: HWND): TModalResult; stdcall;
begin
  //Dllの読み込み
  hDll := LoadLibrary(PWideChar(ADllName));
  try
    if hDll = 0 then
      raise Exception.Create(ADllName + ' を読み込むことができません');
    //Dll関数の読み込み
    @ShowDllForm := GetProcAddress(hDll, PWideChar('ShowDllForm'));
    if @ShowDllForm = nil then
      raise Exception.Create('ShowDllForm関数を読み込めません');
    //Dll関数の実行
    Result := ShowDllForm(Application.Handle);
  finally
    //Dllの解放
    FreeLibrary(hDll);
  end;
end;
```

DLLファイル名

DLL関数を表す変数
関数の定義と一致させる

DLLファイルの読込

DLLファイル内の
DLL関数の読込

読み込んだDLL関数の実行

DLLファイルの解放

DLL形式でアプリケーションを分割

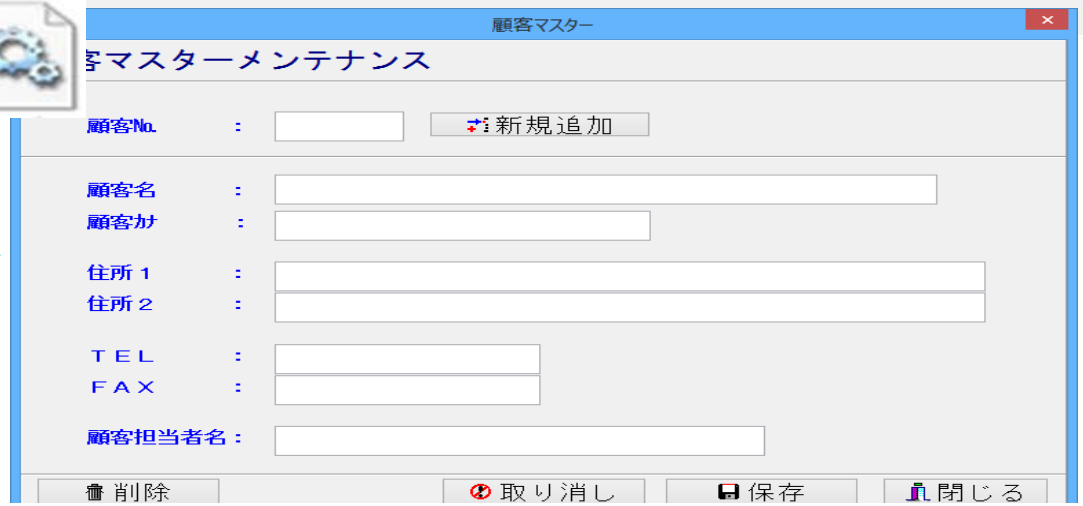
EXE

- EXE側で、DLL名を指定して実行
  DLL関数の宣言なしに、実行時にDLLを読み込むことが可能。


```
//------ DLL 宣言不要

procedure TfrmDSM010.btnDllExecClick(Sender: TObject);
var
  sDllName: String;
begin
  sDllName := edtDllName.Text;
  ShowForm(sDllName);
end;
```

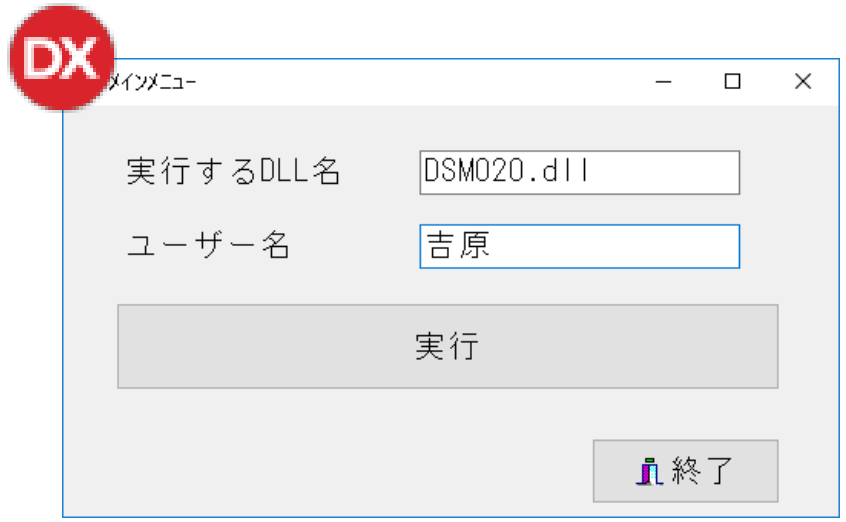
前ページで作成した
サブルーチンを使用
（引数:DLL名）

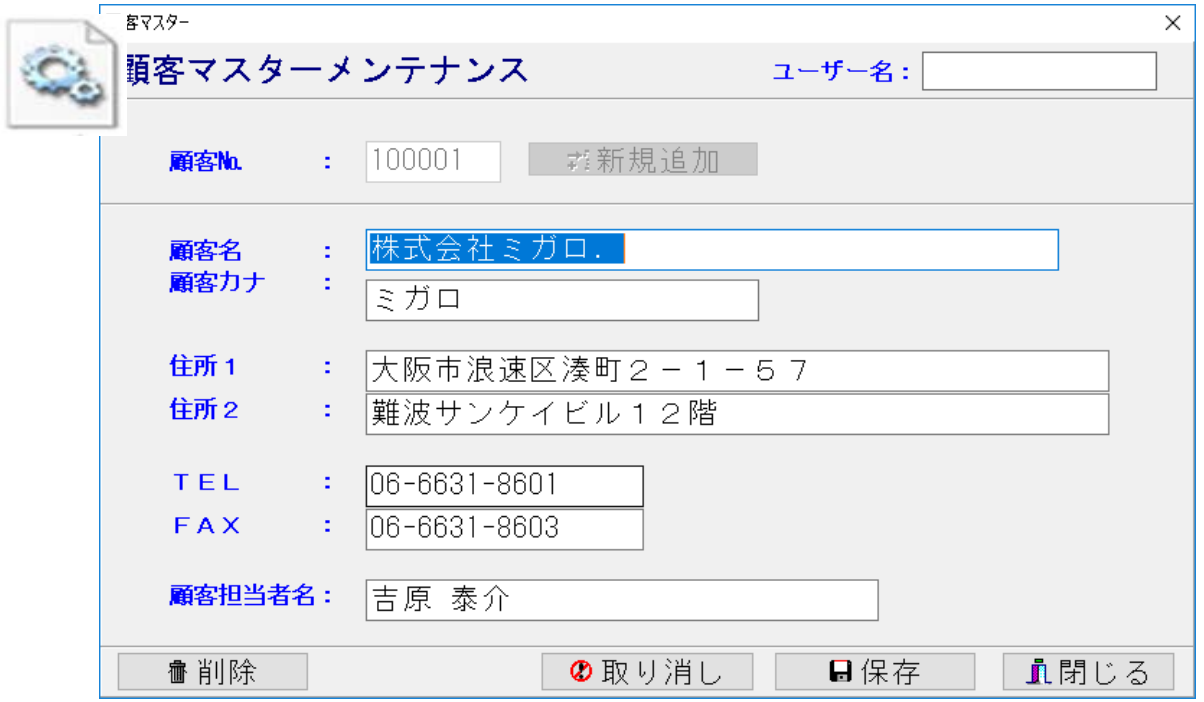


例えばメニュー項目をマスター化すれば、メインプログラムは修正不要！

DLL形式でアプリケーションを分割

- EXEとDLLでデータベース接続を共有
  EXE側でデータベース接続したものをDLL側でも使用できれば、
  データベース接続の共有化が可能。





DLL形式でアプリケーションを分割

EXE

- FireDACでの共有例

DLL呼出し時にEXE側のFDConnectionのハンドルを渡す

```
function TfrmDSM010.ShowForm(ADllName: String): TModalResult;
var
  hDll: Integer;
  ShowDllForm: function(AppHandle: HWND; ACliHandle: Pointer): TModalResult; stdcall;
begin
  //Dllの読み込み
  hDll := LoadLibrary(PWideChar(ADllName));
  try
    if hDll = 0 then
      raise Exception.Create(ADllName + ' を読み込むことができません');
    //Dll関数の読み込み
    @ShowDllForm := GetProcAddress(hDll, PWideChar('ShowDllForm'));
    if @ShowDllForm = nil then
      raise Exception.Create('ShowDllForm関数を読み込めません');
    //Dll関数の実行
    Result := ShowDllForm(Application.Handle, FDConnection1.CliHandle);
  finally
    //Dllの解放
    FreeLibrary(hDll);
  end;
end;
```

DLL関数にFDConnectionのハンドルを渡すパラメータを追加

FDConnectionのChiハンドルをセット

DLL形式でアプリケーションを分割

- FireDACでの共有例

DLL側で、FDConenctionのハンドルを受け取る

```
library DSM020;
  ...
function ShowDllForm(AppHandle: HWND; ACliHandle: Pointer): TModalResult; stdcall;
begin
  Application.Handle := AppHandle;
  try
    frmDSM020 := TfrmDSM020.Create(Application);
    frmDSM020.FDConnection1.SharedCliHandle := ACliHandle;    //受け取ったFDConnectionのハンドルをセット
    try
      Result := frmDSM020.ShowModal;
    finally
      frmDSM020.Free;
    end;
  finally
    Application.Handle := 0;
  end;
end;

exports
  ShowDllForm;
```

DLL関数にデータモジュール
を渡すパラメータを追加

EXE側で生成されたFDConenctionのハンドル
をDLL側のFDConnectionのSharedChiHandleにセット

THANKS!

www.embarcadero.com/jp

第34回 エンバカデロ・デベロッパーキャンプ