【セッションNo. 4】

プログラミングテクニックセッション2

開発者が知りたい実践プログラミングテクニック！　明日から使えるテクニック集

株式会社ミガロ.
RAD事業部　営業・営業推進課
尾崎　浩司

■ 多くの開発者から聞く共通の悩み

- アプリケーションのレスポンスを改善したい。
  - 処理に時間がかかると、画面の応答がなくなってしまう。
- プロジェクトを効率よくメンテナンスしたい。
  - 画面や機能が多くなってくると、プロジェクトの管理が煩雑になる。
- プログラムの入れ替えをシンプルに行いたい。
  - 都度ユーザーにプログラムの置き換えを依頼しないといけない。

- 課題を解決する為のヒントをテーマとします！

【アジェンダ】

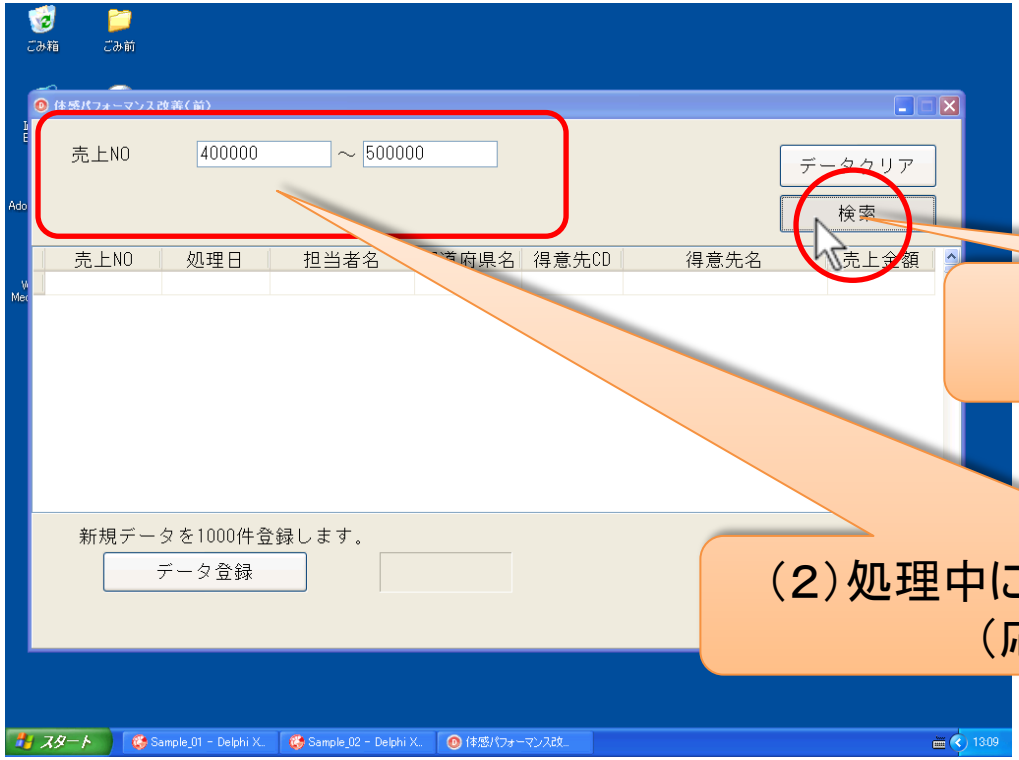
- 課題を解決する為に工夫したプログラミングテクニックを厳選してご紹介！

1. スレッドを使用した実用レスポンス向上
2. DLLを使用したプロジェクト分割手法
3. 実行ファイルバージョンアップテクニック

1．スレッドを使用した実用レスポンス向上

■ アプリケーションのパフォーマンス

- 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、イライラしてしまう。
    （一般的にストレスを感じない応答時間は、約3秒！）



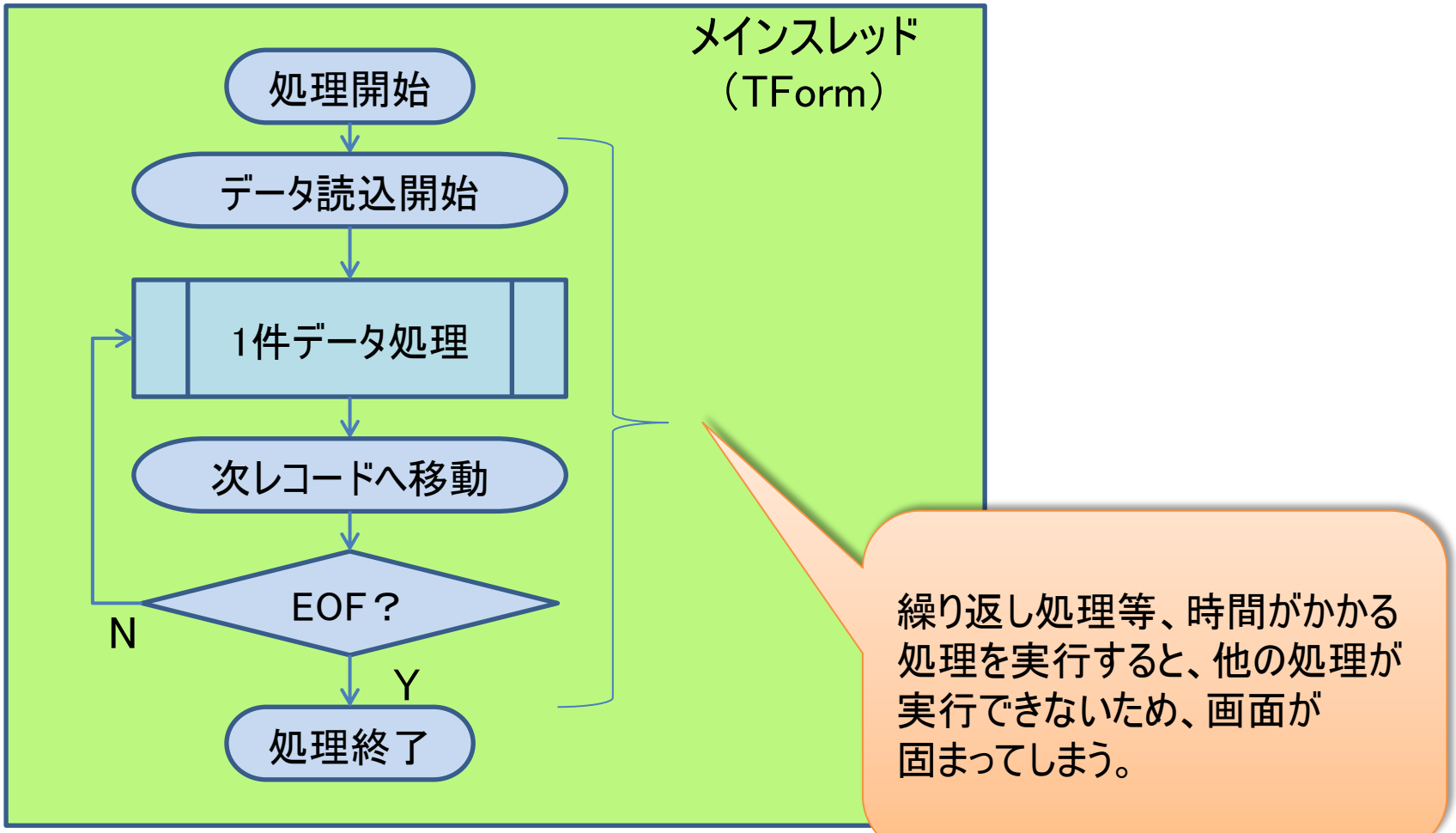
（1）ボタンを押下したか、
していないかが分からない。

（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。

- なぜ応答がなくなってしまうのか？

■ アプリケーションが固まる理由

- 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。



■ シングルスレッドプログラム

- 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;
  //データをグリッドに表示
  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;
```

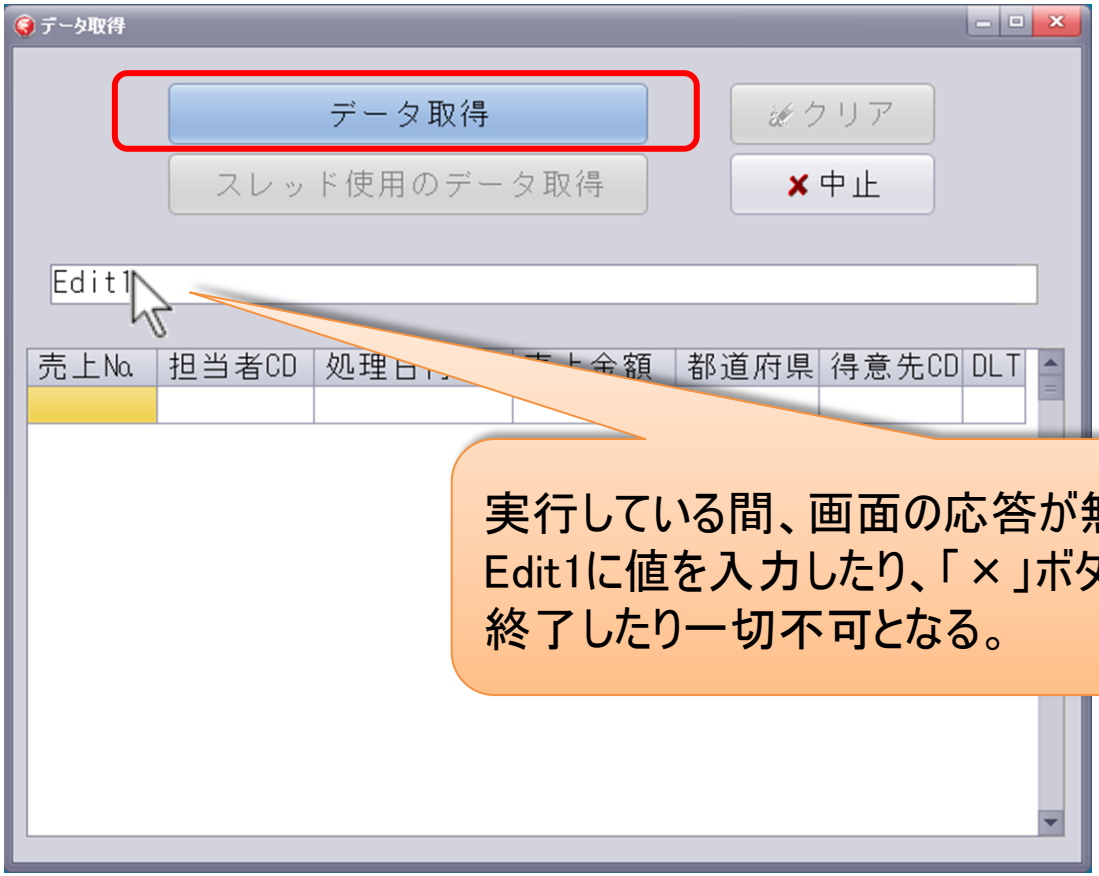
繰り返し処理

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

■ シングルスレッドプログラムの実行

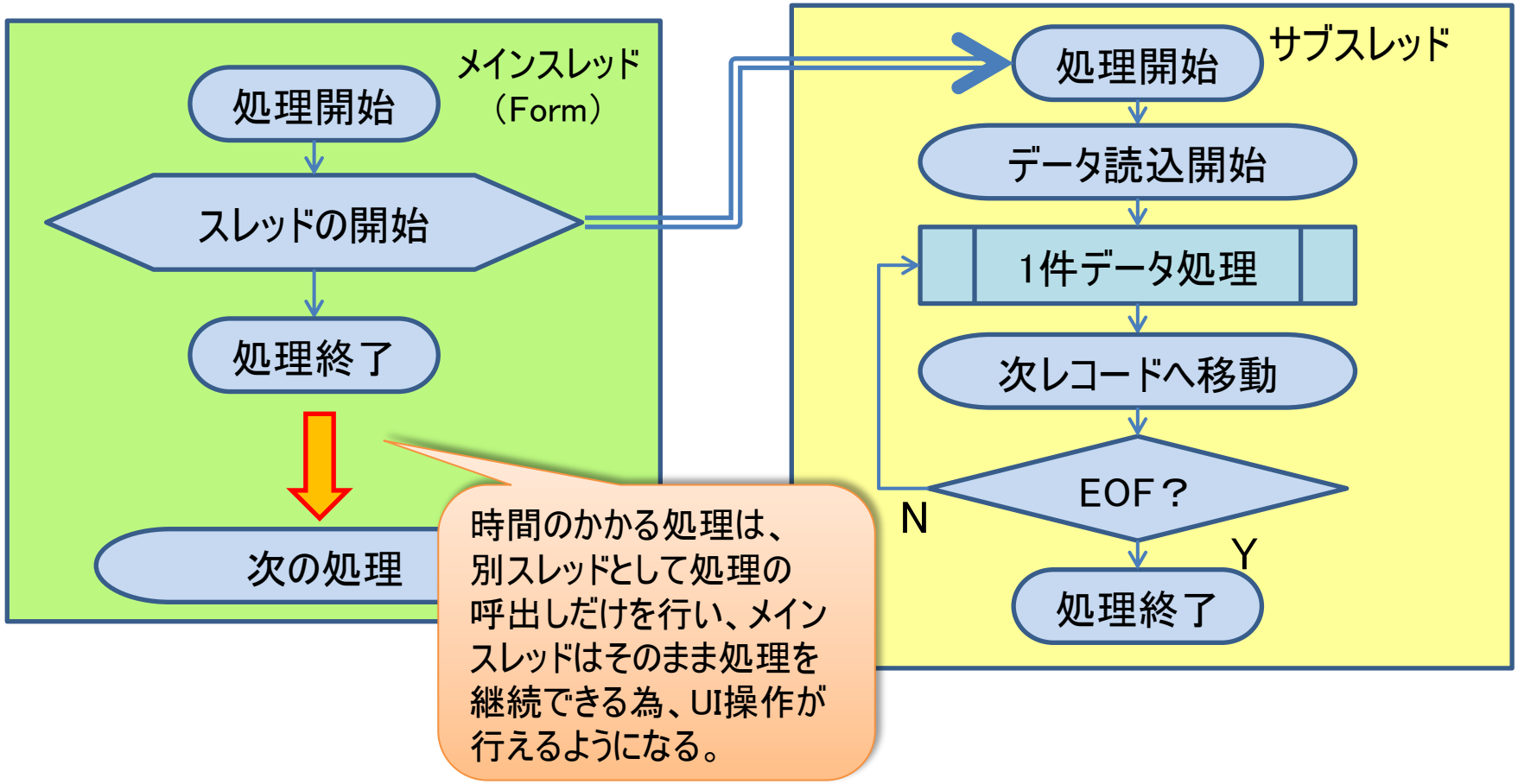
- シングルスレッド実行例



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。

- 固まってしまうのを防ぐことはできないか？

■ マルチスレッドプログラム



- マルチスレッドにより**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になる。

Delphi/400による従来からのマルチスレッド

- TThread　クラスを使用して、別スレッドを記述。
  - 使用方法についての詳細は、下記にてご紹介済み
    第12回テクニカルセミナー
    「Delphi/400開発～パフォーマンス向上テクニック～」

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。

- もっとシンプルに書けないか？

CreateAnonymousThread を使ったスレッド処理

- メインスレッドの中に直接サブスレッドを記述可能

```
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！

■ マルチスレッドプログラム

```
procedure TForm1.btnThreadGetDataClick(Sender: TObject);
begin
  TThread.CreateAnonymousThread(
  procedure()
  var
    i, iRow: Integer;
  begin
    iRow := 0;
    //データをグリッドに表示
    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;
```

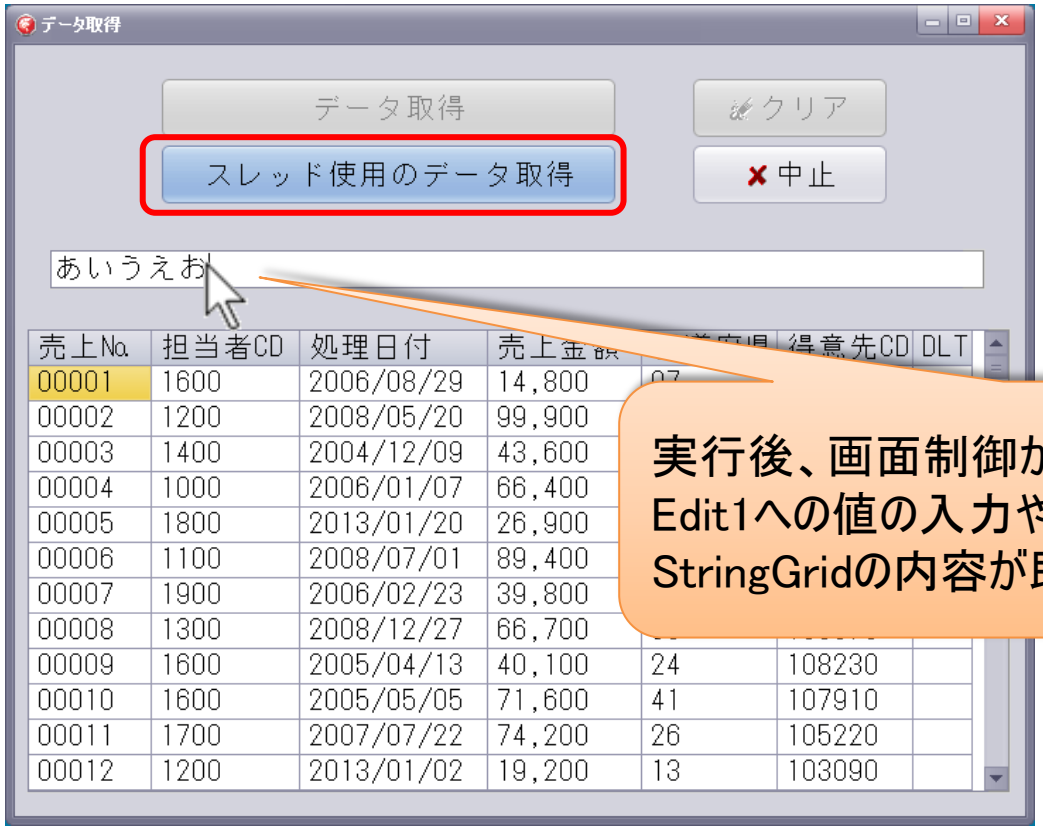
スレッドの生成

P.7のシングル
スレッドプログラム
と同じコード

スレッドの開始

■ マルチスレッドプログラムの実行

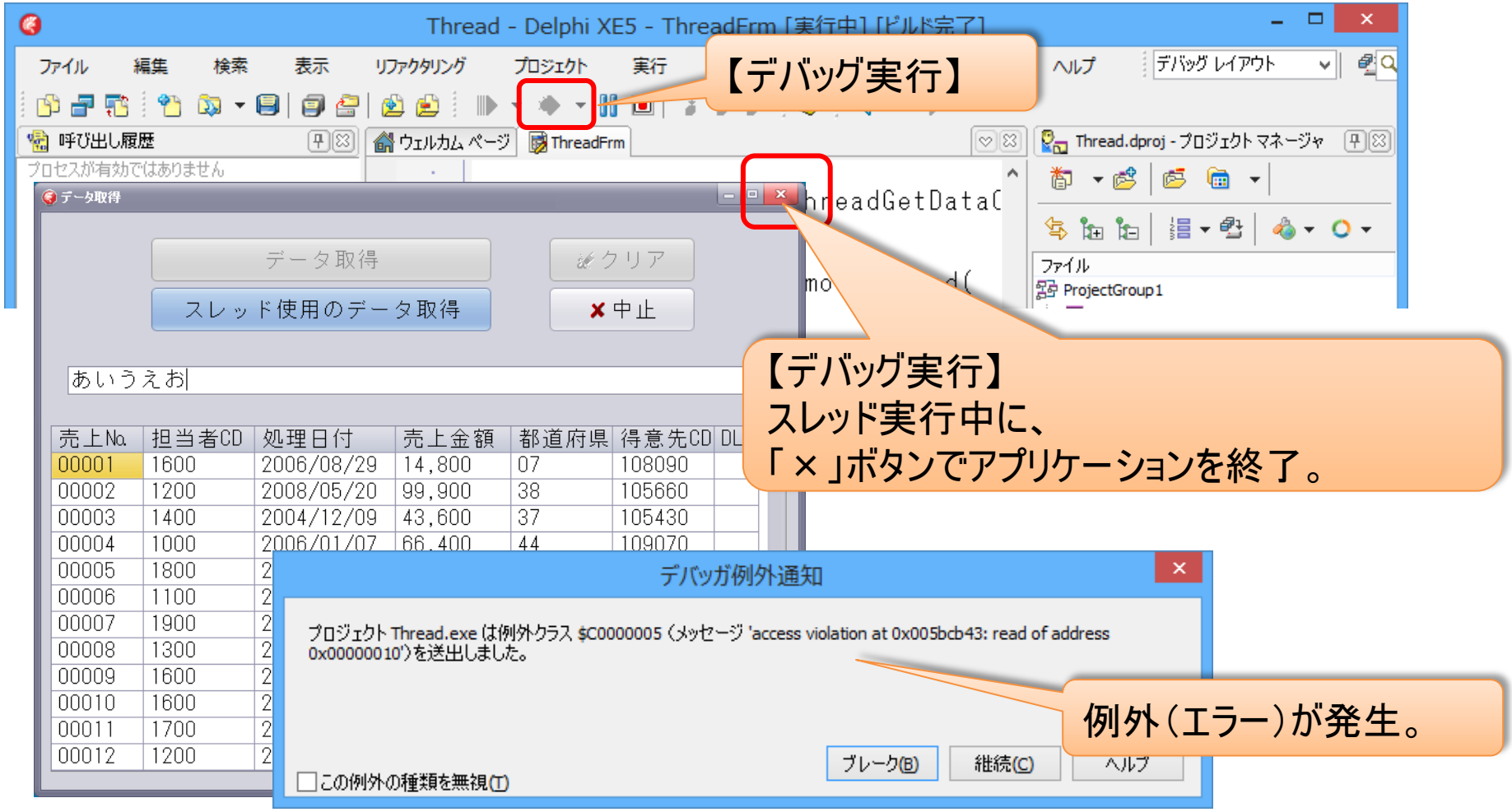
- マルチスレッド実行例



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！

- レスポンスタイムが大幅に向上！

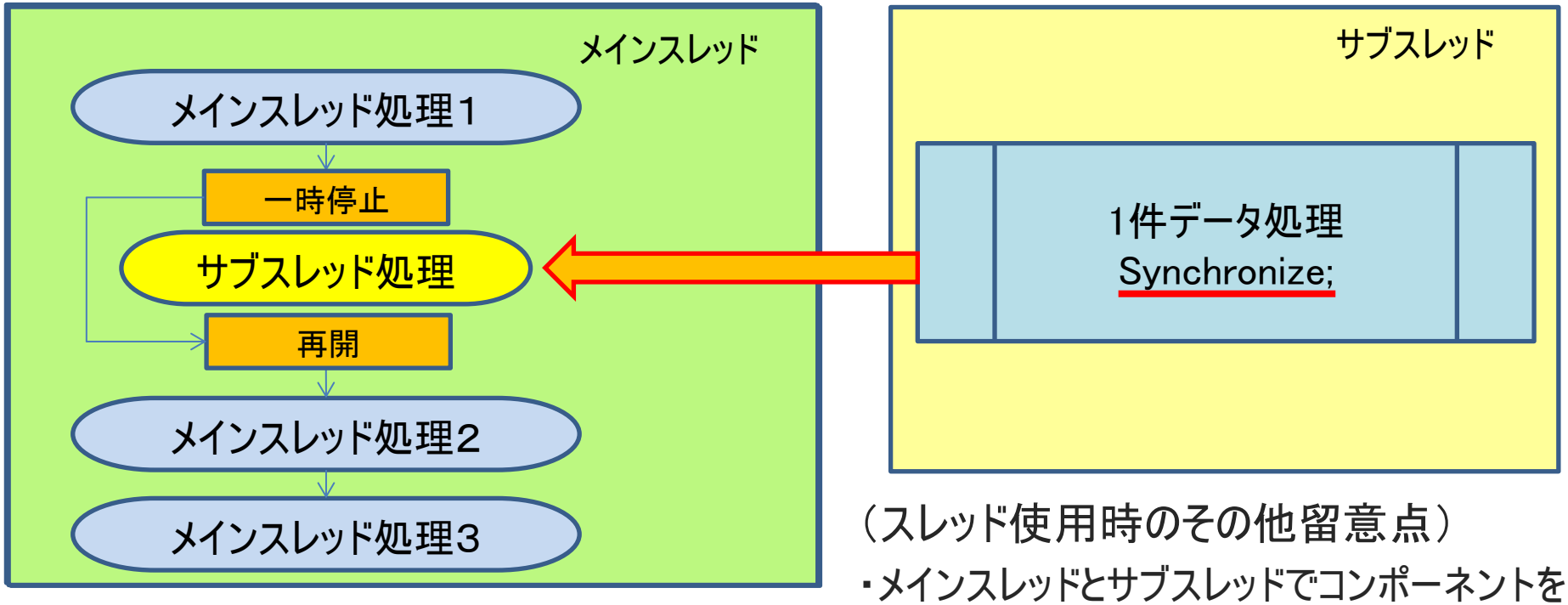
■ マルチスレッドの考慮点



- なぜ例外が発生するか？

■ マルチスレッドの考慮点

- **VCL（コンポーネント）が使用できるのは、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記載しない。

Synchronizeを使用したVCL操作

- サブスレッドの中に直接Syncronizeを追加できる。

```
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。

Synchronizeを使用した改良

```
procedure TForm1.btnThreadGetDataClick(Sender: TObje
begin
  TThread.CreateAnonymousThread(
  procedure()
  var
    i, iRow: Integer;
  begin
    iRow := 0;
    //データをグリッドに表示
    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;
```

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

Synchronizeプログラムの実行

- 改良したマルチスレッド実行例

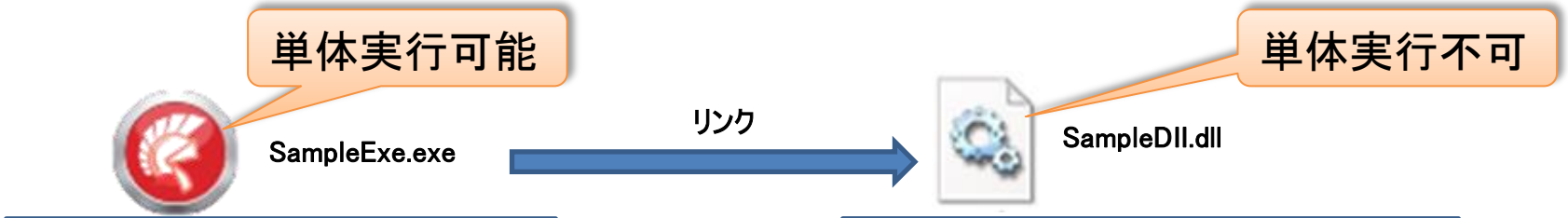
【デバッグ実行】
スレッド実行中に、
「×」ボタンでアプリケーションを終了しても、エラーとならない。

- Synchronizeを使用することで、安全にスレッドを使用可能！

２．DLLを使用した
プロジェクト分割手法

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;
```

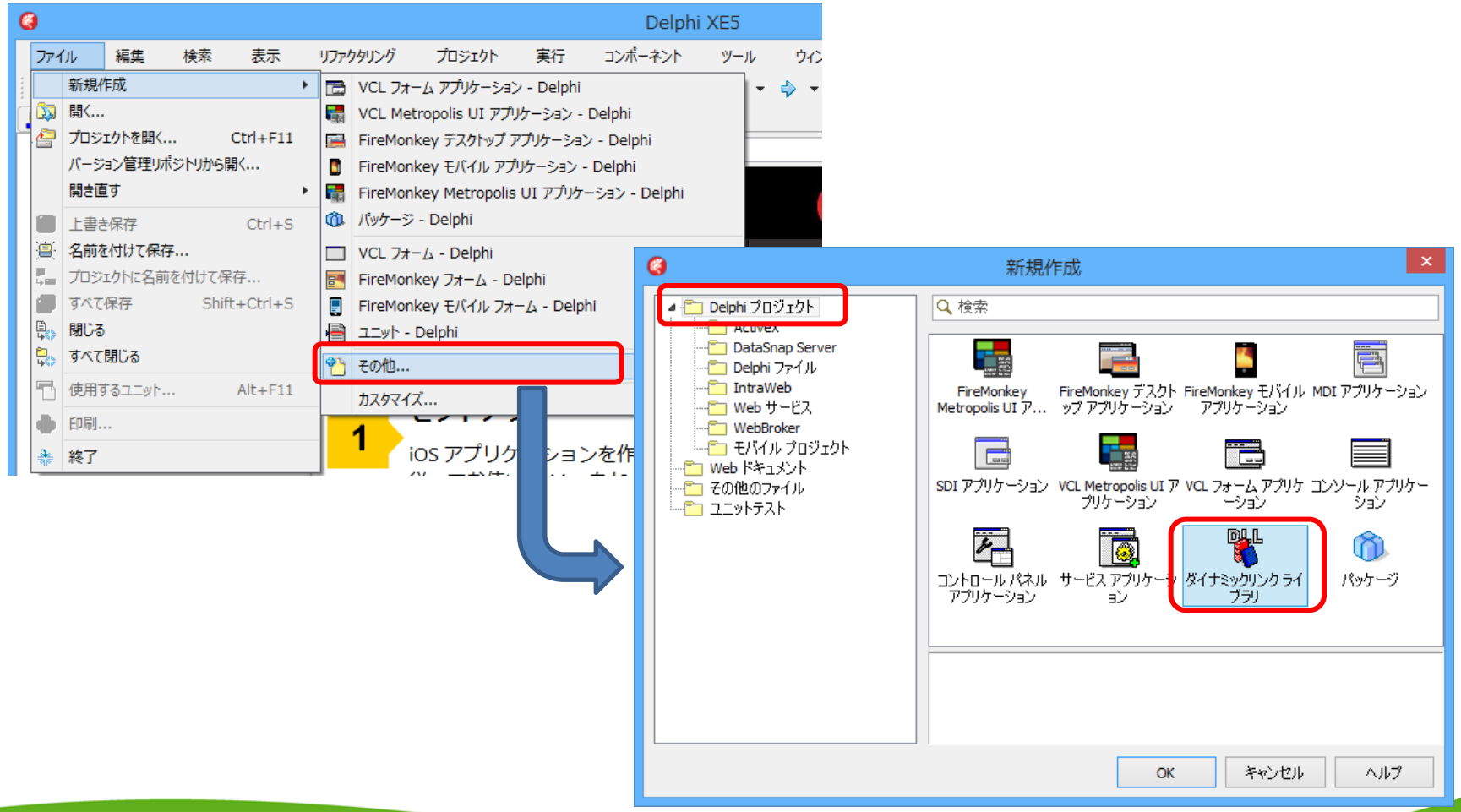
関数（サブルーチン）を定義

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

- DLL化により、色々なプログラムからサブルーチンが利用可能となる！

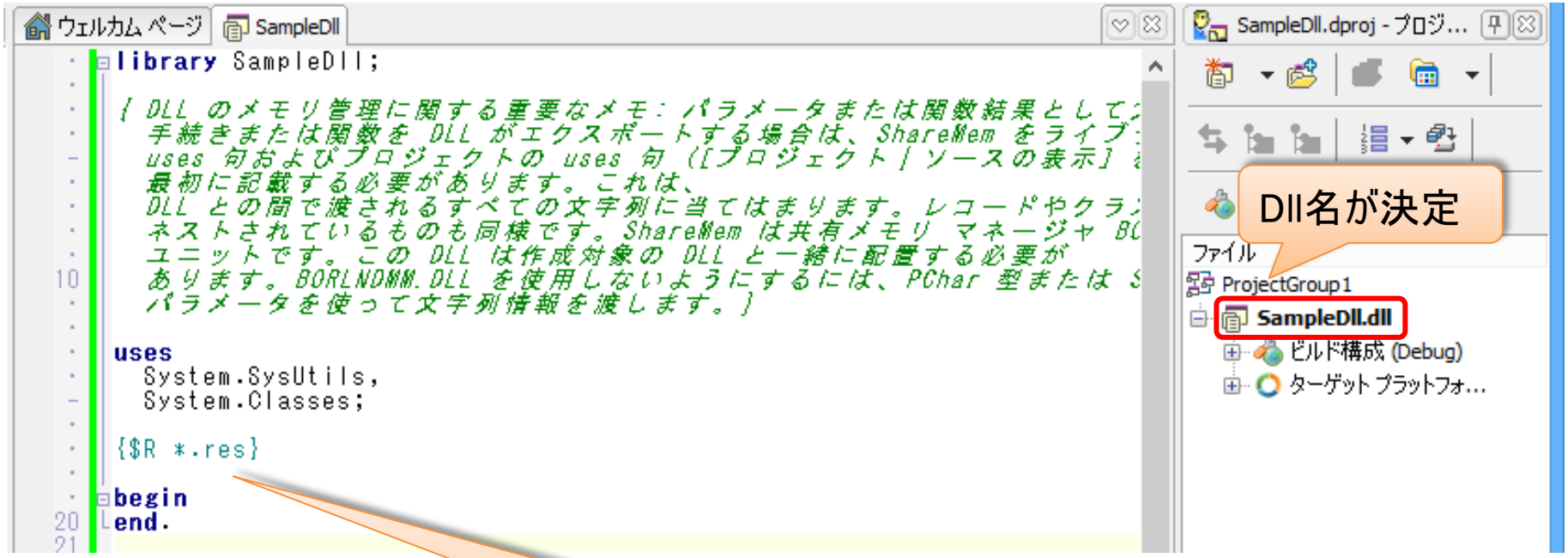
DLL作成方法

- 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作成方法

- 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.
```

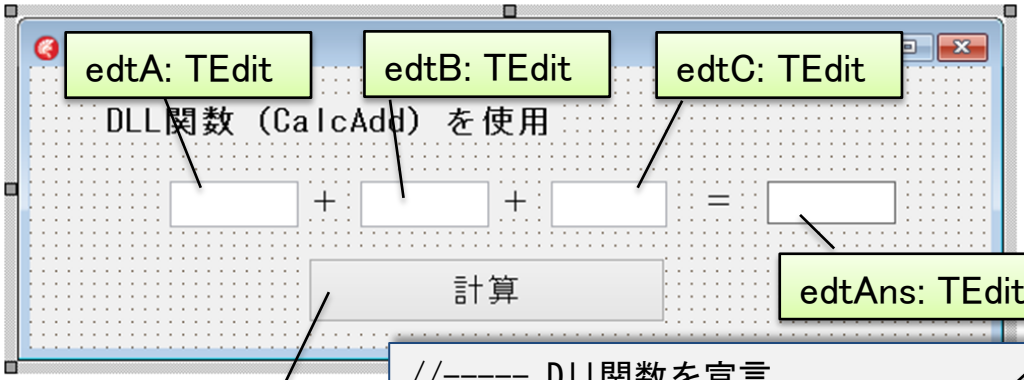
実行したい手続き/関数

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を呼び出すExeプログラム

- VCLフォームアプリケーションよりDLL呼出し



btnCalc: TButton

DLL側の手続き/関数を宣言
external句 に参照するDLLを指定

```
//----- 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

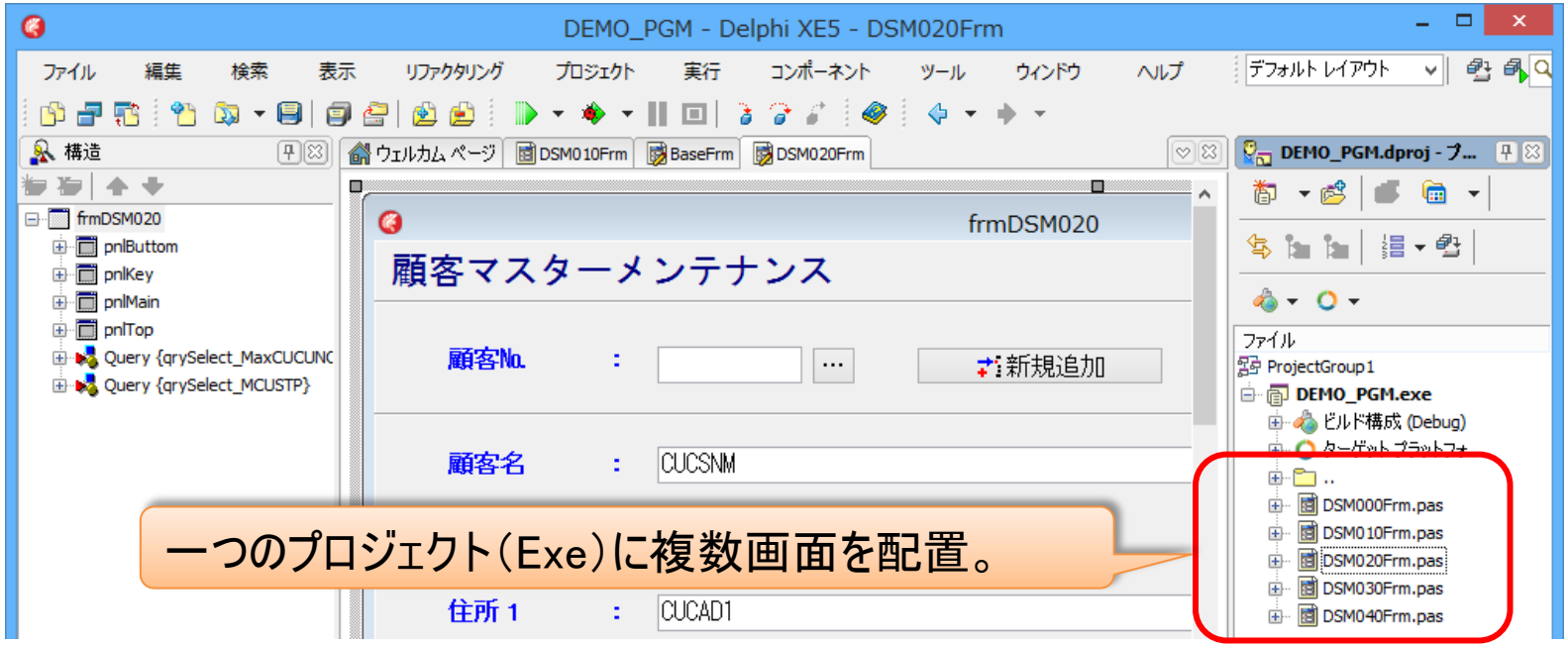
■（補足）DLLプロジェクト デバッグ方法

DLL

- 呼出し元のExeプログラムを定義することでデバッグが可能
  - [実行]→[実行時引数] より「ホストアプリケーションを指定」

■ 一般的な単体Exeプロジェクト構成

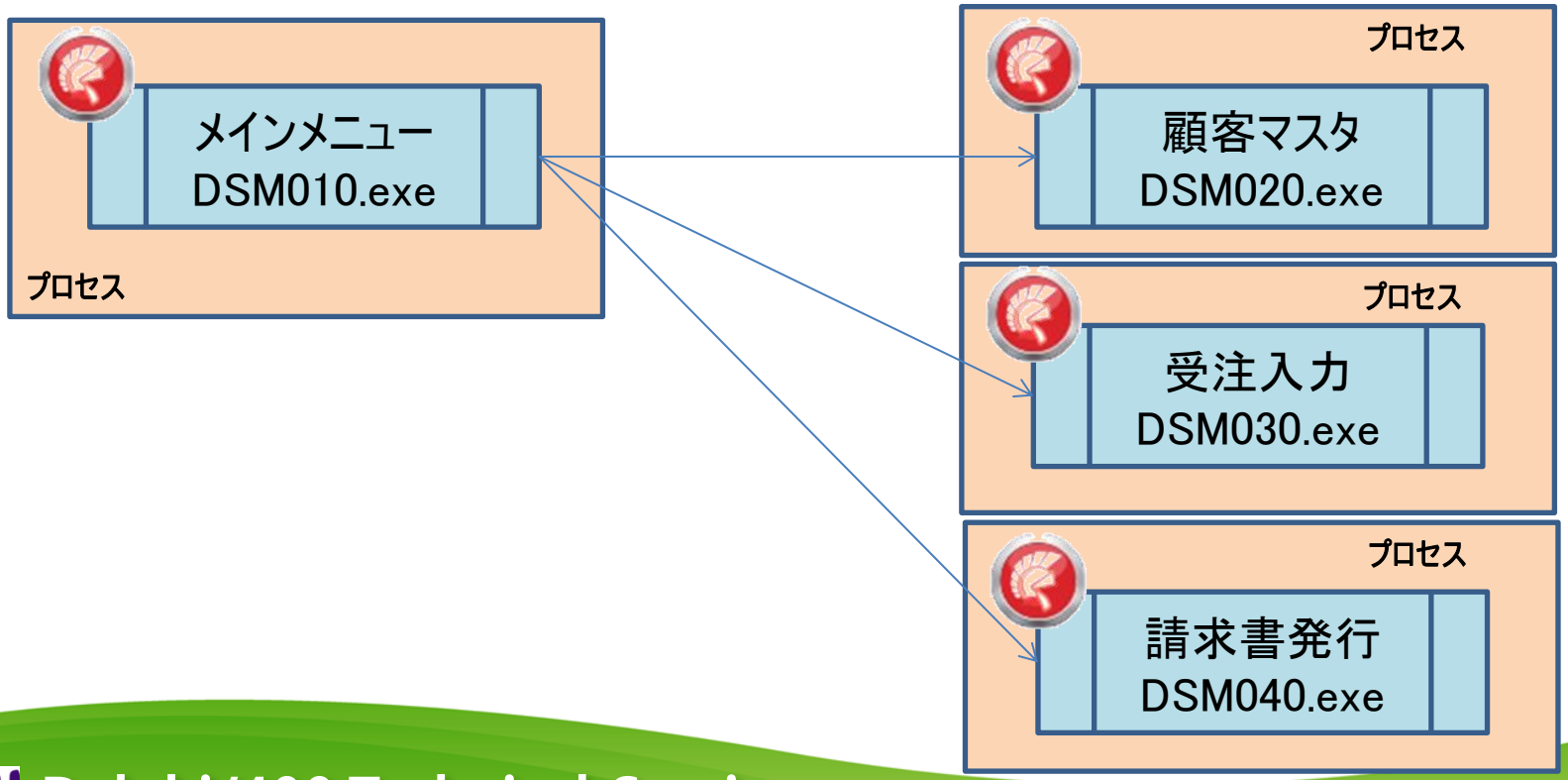
- 一つのプロジェクト(Exe)で、複数フォーム（機能）を統合
  - グローバル変数等により、画面間の値の受け渡しが容易
  - Exeファイル一つでシステムが完結する
  - 画面（機能）数が多くなると、実行ファイルサイズが拡大
  - 仕様変更の都度、プロジェクト全体のExe再配布が必要



- 各フォーム（機能）を分割することはできないか？

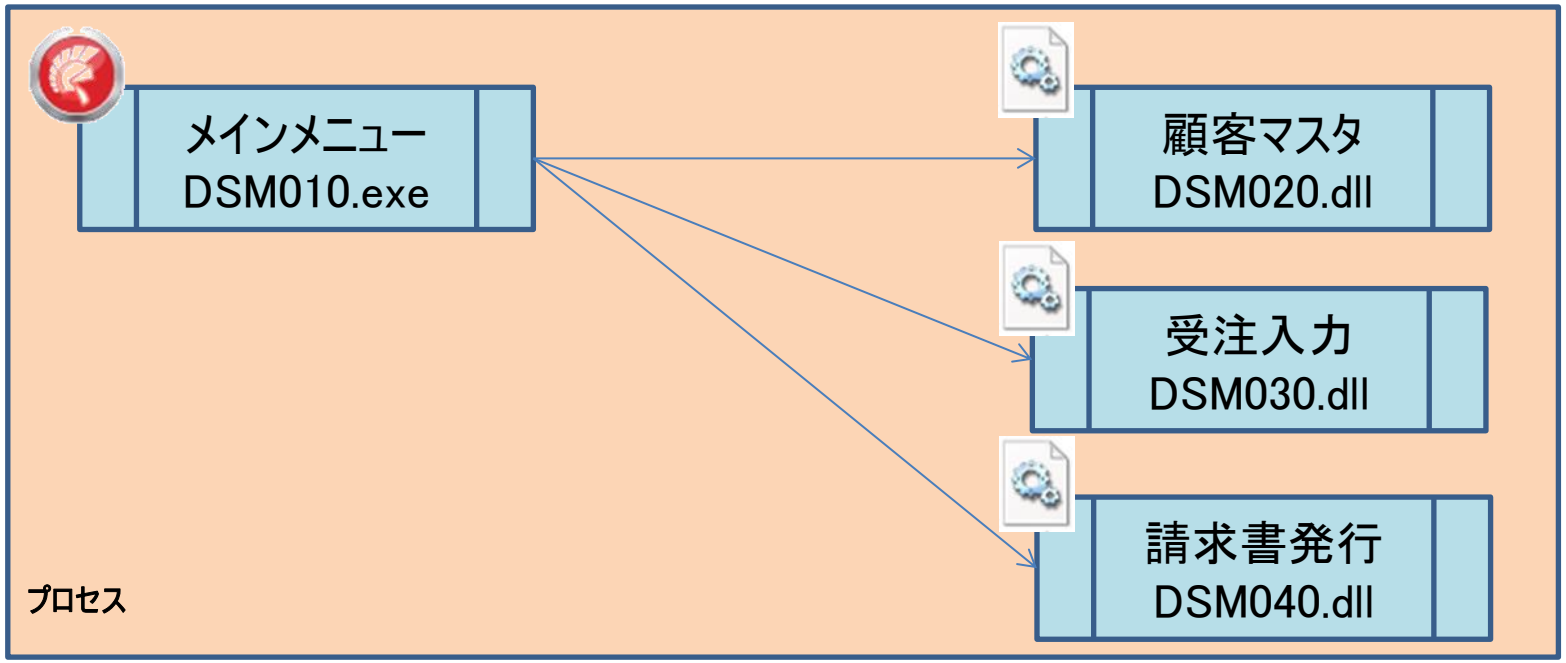
■ 機能ごとにプロジェクト(Exe)で分割

- メニュー用のExeと各機能ごとにプロジェクト(Exe)を分割
  - 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、IBMiへの接続ジョブが生成される
  - Exe間の値の受け渡し方法が必要 （実行時引数など）



■ 機能ごとにプロジェクト(DLL)で分割

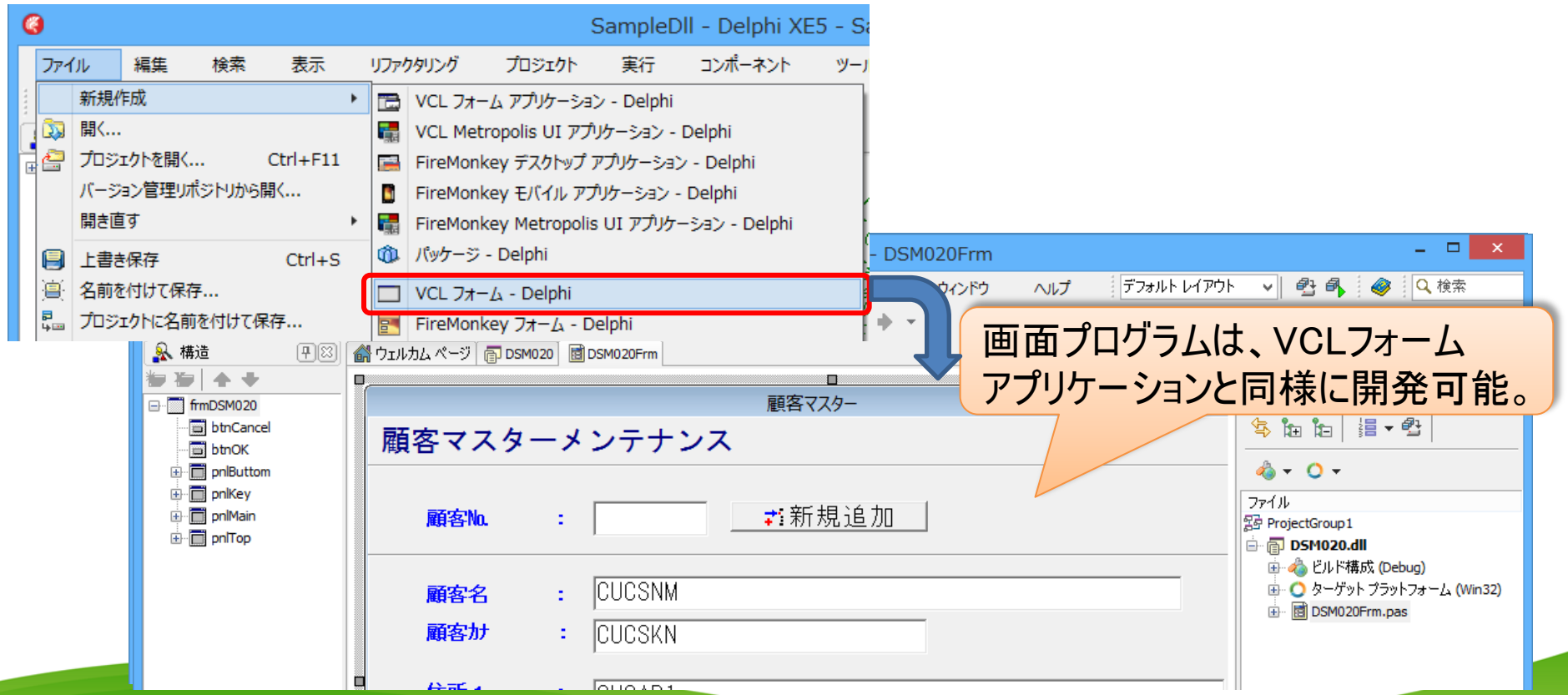
- メニュー用のExeと各機能ごとにプロジェクト(DLL)を分割
  - 機能ごとに個別開発、単体テストが行える
  - 個別機能の仕様変更が発生しても、当該DLLのみ置き換えで良い
  - 単体Exeプロジェクト同様、実行プロセスやIBMiへの接続ジョブが一つとなる
  - Exe-DLL間のグローバル変数等の値の受け渡しが可能



- 今回は、DLLによるプロジェクト分割方法を紹介！

■ 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、VCLフォームをプロジェクトに追加
    通常Exe同様、[ファイル]→[新規作成]→[VCLフォーム] で作成可能



■ DLLフォーム呼出し部の作成

- DLLプロジェクトには、自動生成フォームがない
  - 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'   {frmDS

{$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）

Exeアプリのウィンドウハンドルが必要

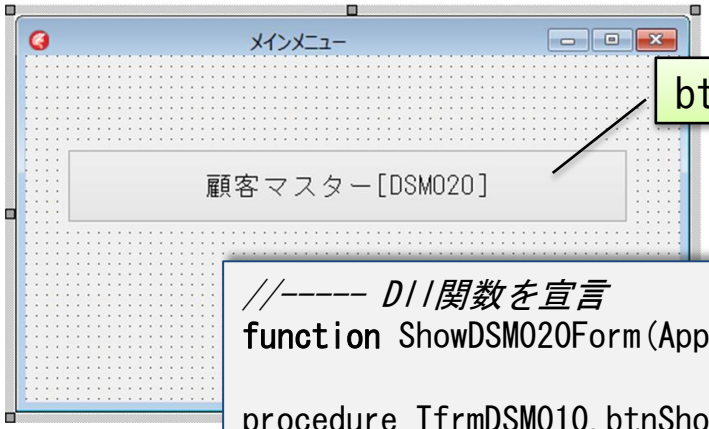
```
function ShowDSM020Form (AppHandle: HWND): TModalResult; stdcall;
begin
  Application.Handle := AppHandle;
  try
    frmDSM020 := TfrmDSM020.Create(Application);
    try
      Result := frmDSM020.ShowModal;
    finally
      frmDSM020.Release;
    end;
  finally
    Application.Handle := 0;
  end;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

exports
  ShowDSM020Form;

begin
end.
```

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

■ メインプログラム(Exe)の作成

- メニューフォームより、DLLフォームを起動

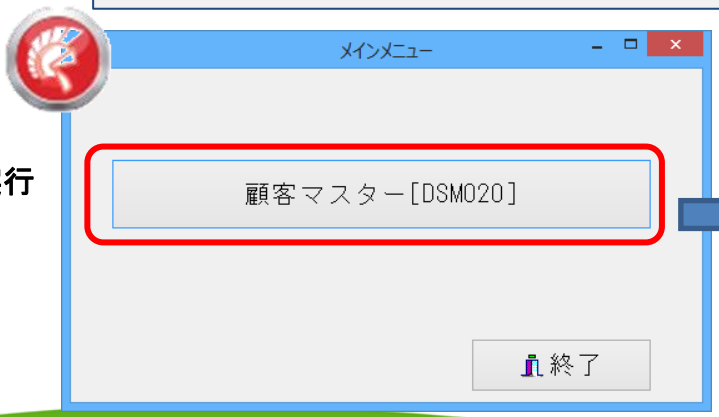


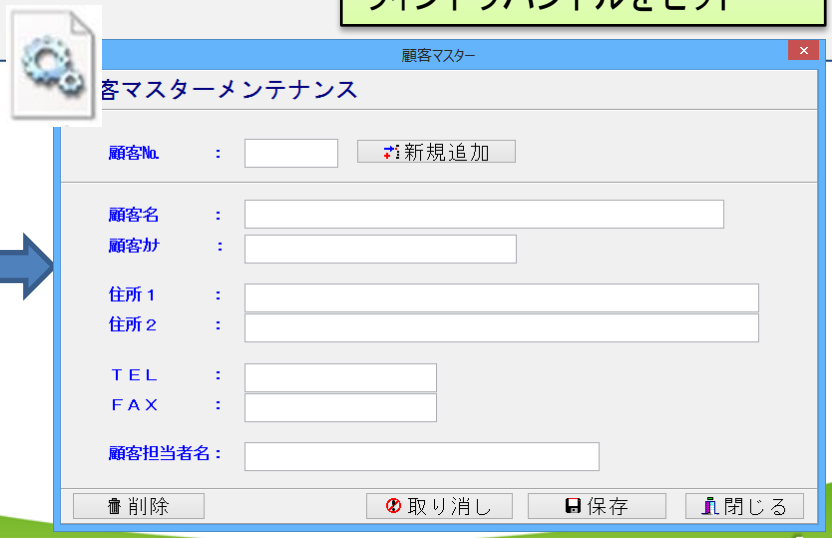
btnShowDSM020: TButton

```
//----- 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





■ メインプログラム(Exe)の課題

- 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
```

もし、DSM050.dll　を追加しようとすると

```
//----- 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？

■ 動的DLLリンクを使用したメインプログラム(Exe)

- LoadLibrary関数で、実行時にパラメータ指定された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。(例:"ShowDllForm")

```
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

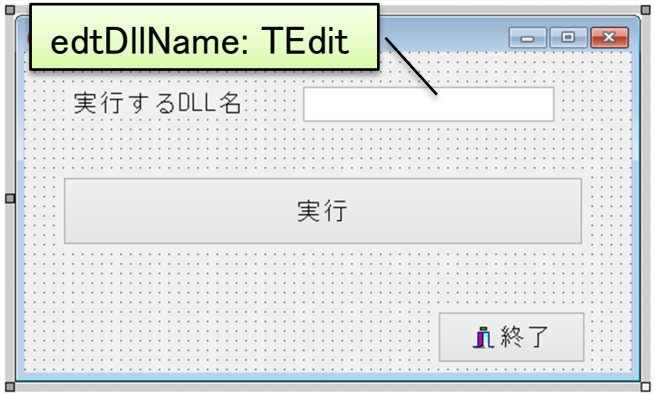
DLLファイル内の
DLL関数の読込

読み込んだDLL関数の実行

DLLファイルの解放

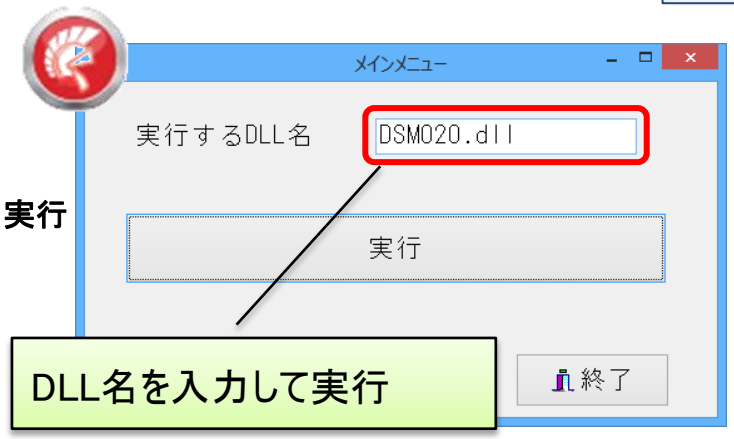
■ 動的DLLリンクを使用したメインプログラム(Exe)

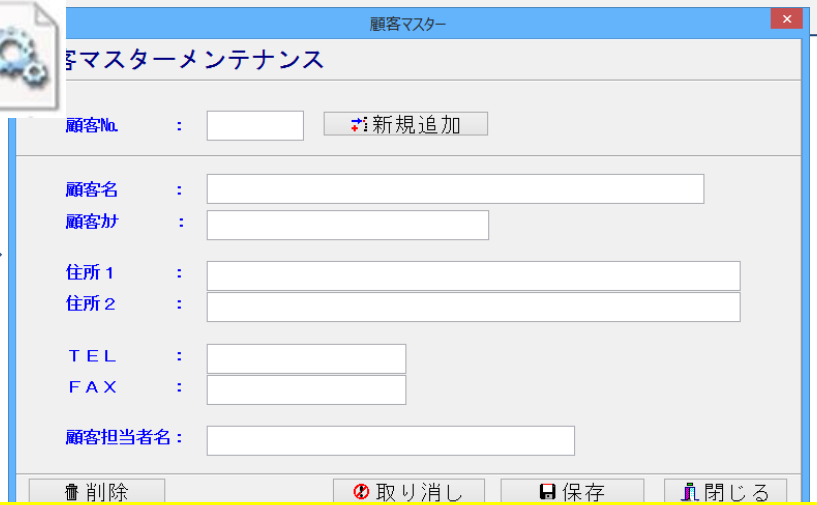
- Exe側で、DLL名を指定して実行
  - DLL関数の宣言なしに、実行時にDLLを読み込むことが可能。




```
//------ 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）

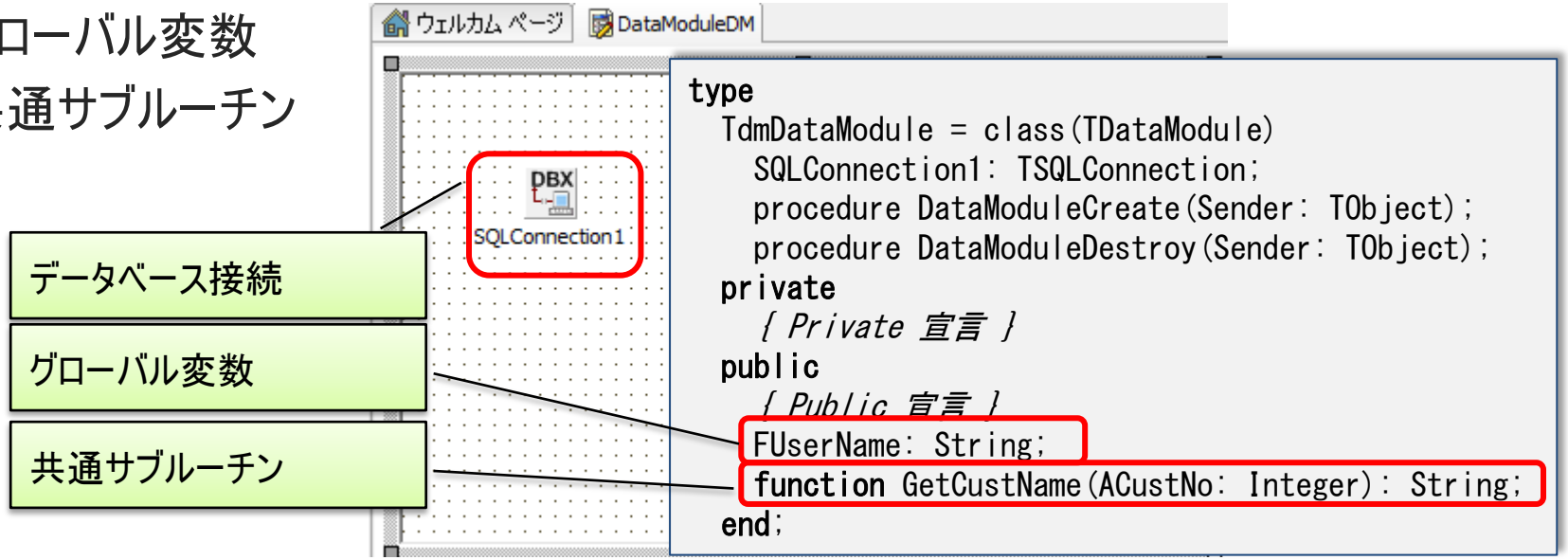




- メニュー項目をマスター化すれば、メインプログラムは修正不要！

■ データモジュールの活用

- アプリケーション共通部分の一元管理に便利
  - データベースの接続ロジック
  - グローバル変数
  - 共通サブルーチン

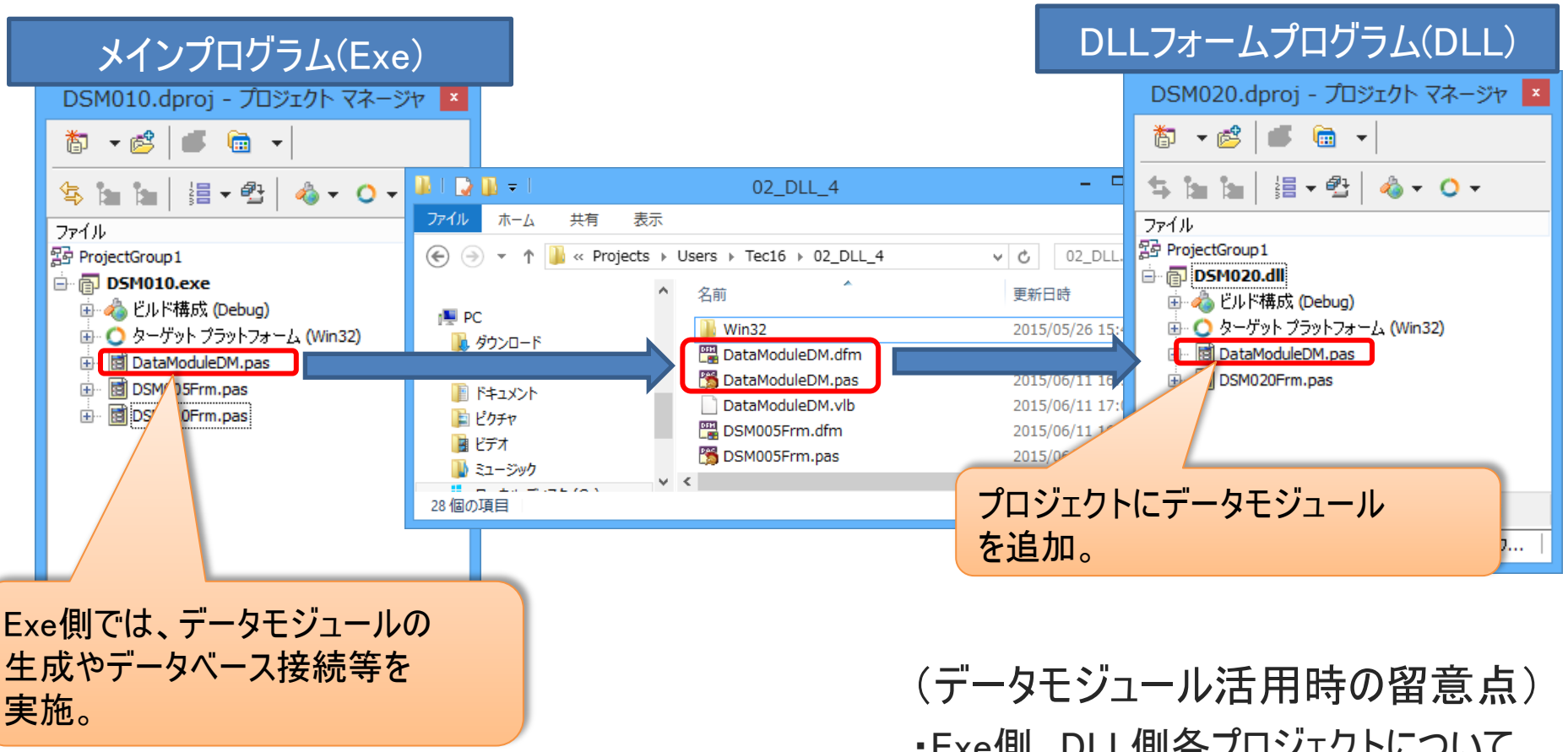


- ExeとDLLのデータモジュール共有
  - Exe側でデータベースの接続したものをDLL側でも使用できれば、IBMiへのジョブ一本化が可能。（つまり、共通でQTEMPも使える）

- Exe側で生成したデータモジュールをDLL側でも使用できないか？

■ データモジュールの活用

- Exe側プロジェクトで作成したデータモジュールユニットをDLL側プロジェクトに追加



（データモジュール活用時の留意点）

・Exe側、DLL側各プロジェクトについて、実行時パッケージを有効にする。

■ メインプログラム(Exe) DLL呼出し部の改良

- DLL呼出し時にExe側のデータモジュールを渡せるように変更

```
function TfrmDSM010.ShowForm(ADllName: String): TModalResult;
var
  hDll: Integer;
  ShowDllForm: function(AppHandle: HWND; DataMod: TDataModule): 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, dmDataModule);
  finally
    //Dllの解放
    FreeLibrary(hDll);
  end;
end;
```

DLL関数にデータモジュールを渡すパラメータを追加

データモジュール変数をセット

DLLフォーム呼出し部の改良

- DLL側で、データモジュールの受け取り部を追加

```
library DSM020;
  ...
function ShowDllForm(AppHandle: HWND; DataMod: TDataModule): TModalResult; stdcall;
begin
  Application.Handle := AppHandle;
  dmDataModule := TdmDataModule(DataMod);  //受け取ったデータモジュールをセット
  try
    frmDSM020 := TfrmDSM020.Create(Application);
    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

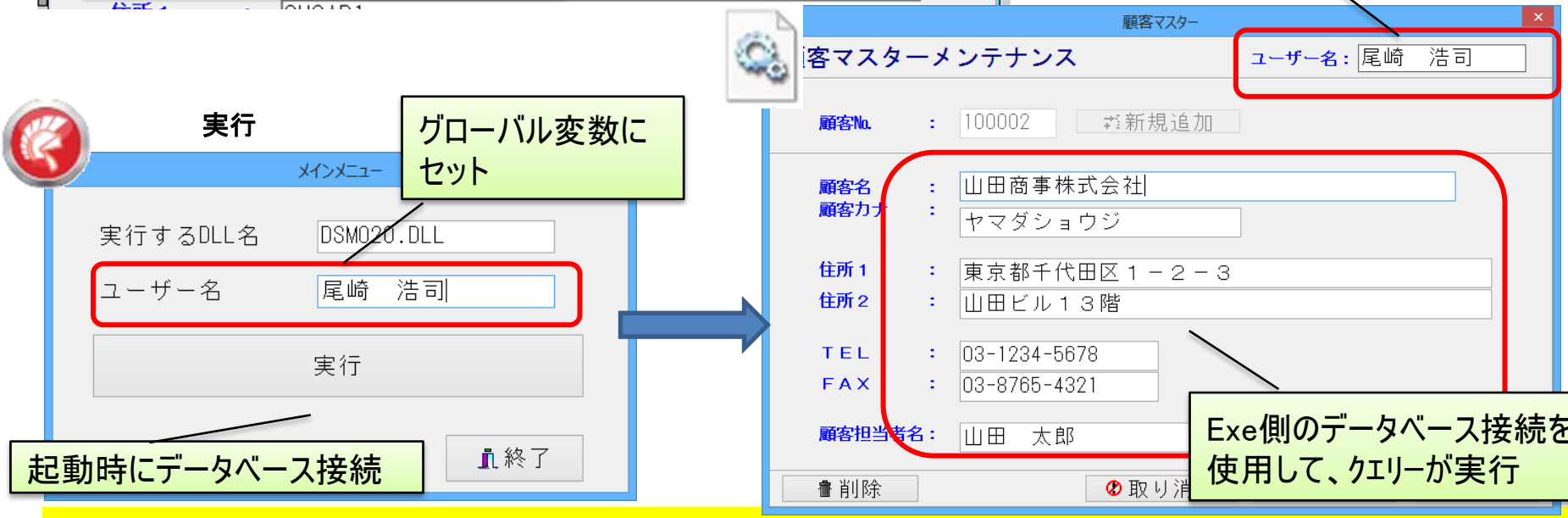
Exe側で生成されたデータモジュール変数をDLL側変数にセット

■ DLL側プログラムの実行

DLL

- データモジュールを使用するDLLフォーム

グローバル変数の値
が表示

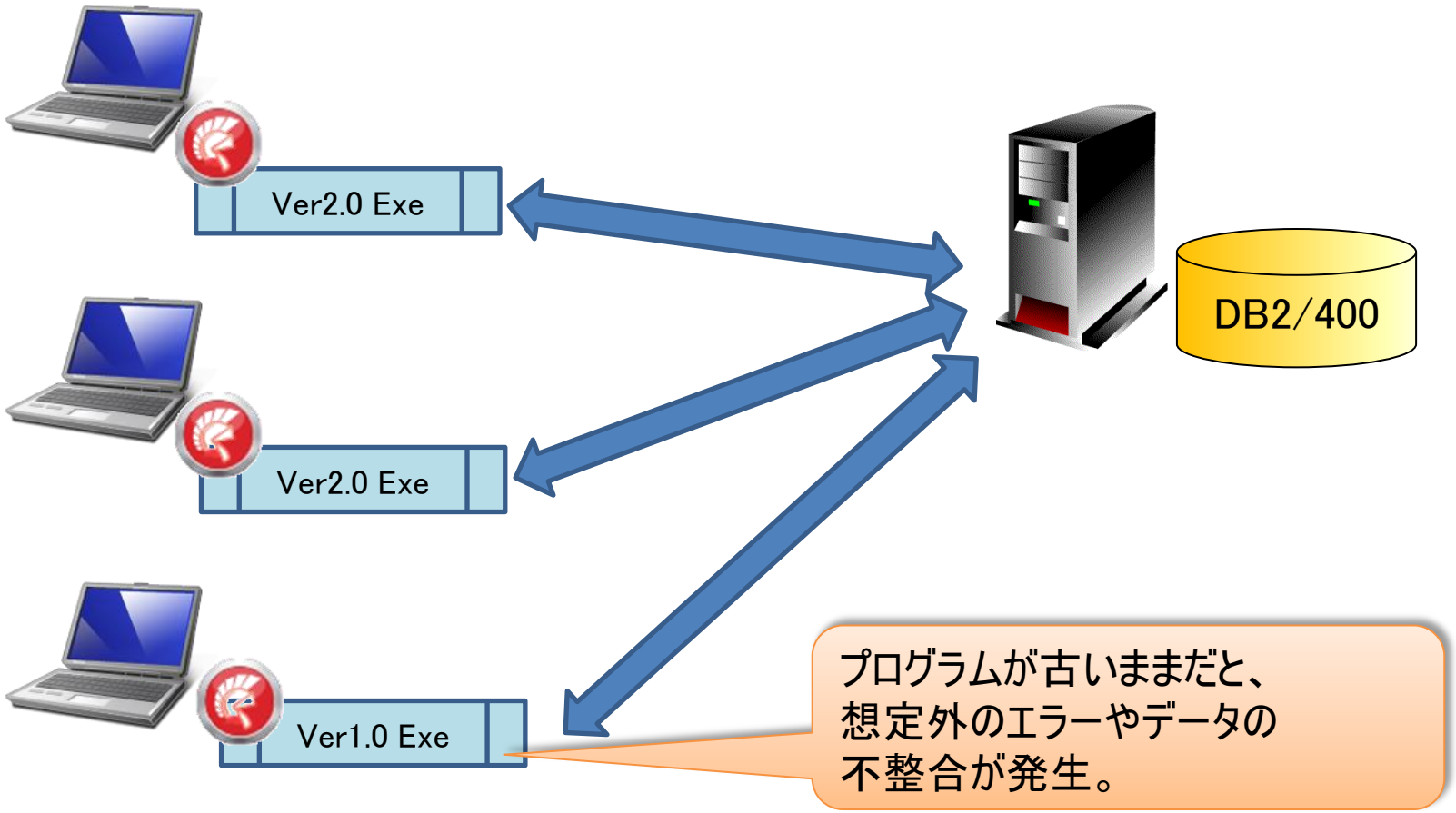


- DLLからもExeと同じデータモジュールが使用可能！

３．実行ファイル
バージョンアップテクニック

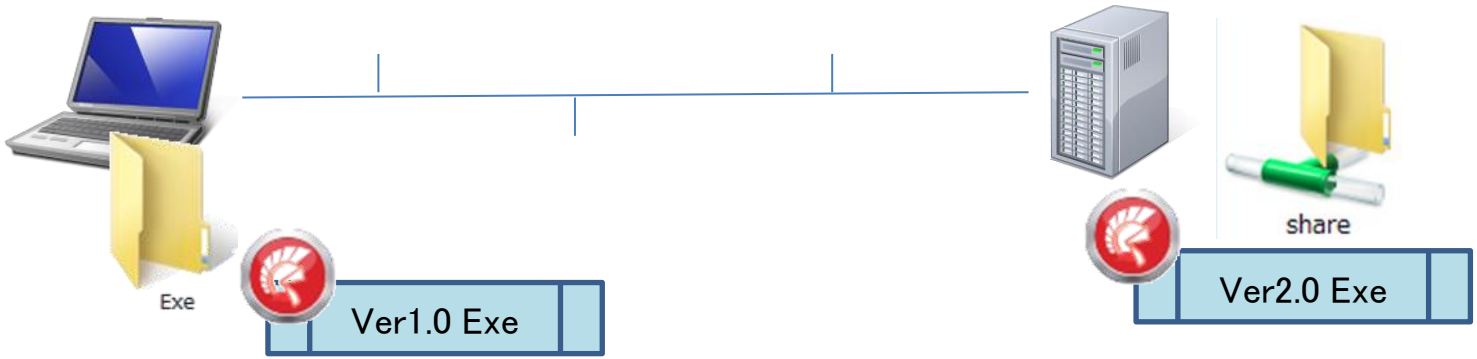
アプリケーションのバージョンアップ

- プログラムは常に最新版で稼働させたい。



■ バージョン管理方法の検討

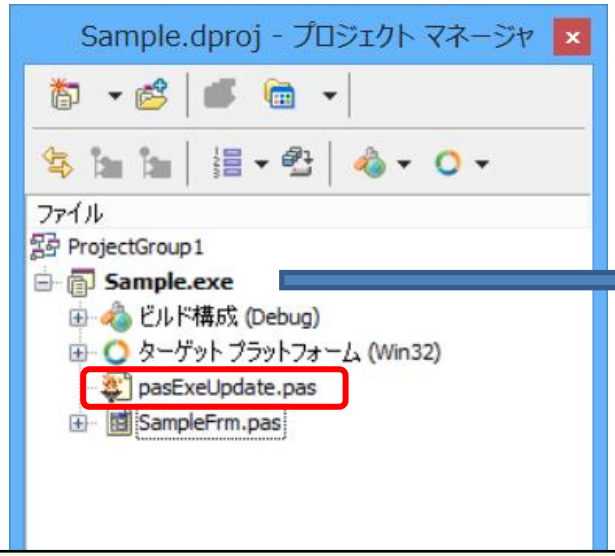
- ファイルサーバーを使用したバージョン管理を検討
  - ユーザーに告知し、ユーザー自身が直接ファイルをコピー
    → 作業漏れの可能性がある。
  - ログオン時にバッチファイルを実行し、ファイルをコピー
    → ログオン時しか入れ替えられない。
  - プログラム開始時にバージョン比較して、ファイルをコピー
    → Exe実行中は、自分自身のExeファイルを置き換えられない。



- Exeファイルをスムーズに置き換える方法はないか？

■ Exe自動バージョンアップユニット

- 使用方法
  - プロジェクトファイルに[pasExeUpdate.pas] を追加
  - プロジェクトメインルーチン(.dpr)
    に**PgmUpdate関数**を追加



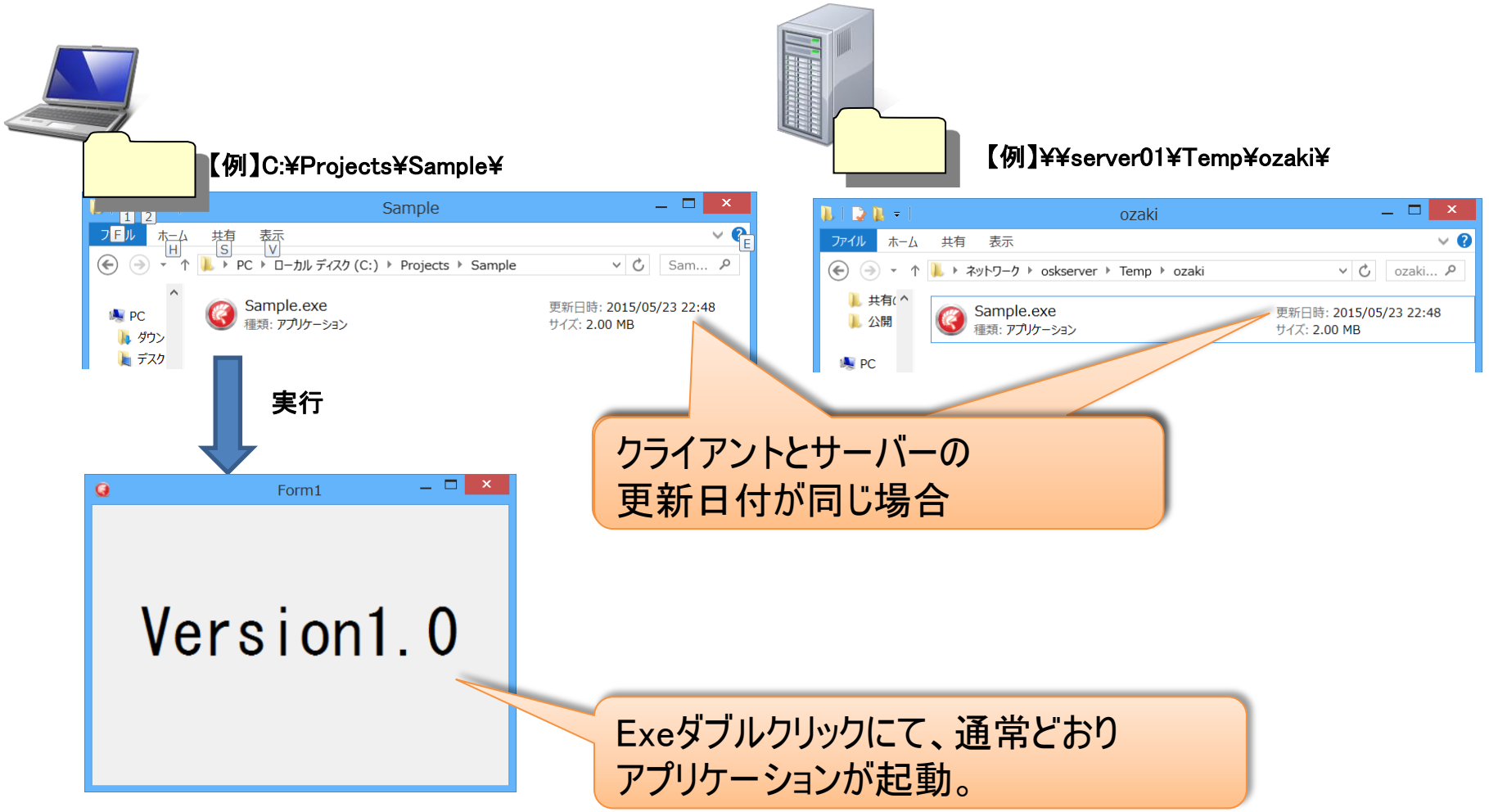

```
program Sample;

uses
  Vcl.Forms,
  SampleFrm in 'SampleFrm.pas' {Form1},
  pasExeUpdate in 'pasExeUpdate.pas';
{$R *.res}

begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
//---↓ ここにバージョンチェックロジックを追加
{$IFNDEF DEBUG}  //---- デバッグ時は実行しない
  if PgmUpdate('¥¥[FileServer]¥[Dir]¥') then
    Application.Terminate;
{$ENDIF}
//---↑ ここにバージョンチェックロジックを追加
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

ファイルサーバー上のExe格納
共有フォルダを指定

（例えば、Iniファイル等にPath情報を
設定しておくと利便性向上）

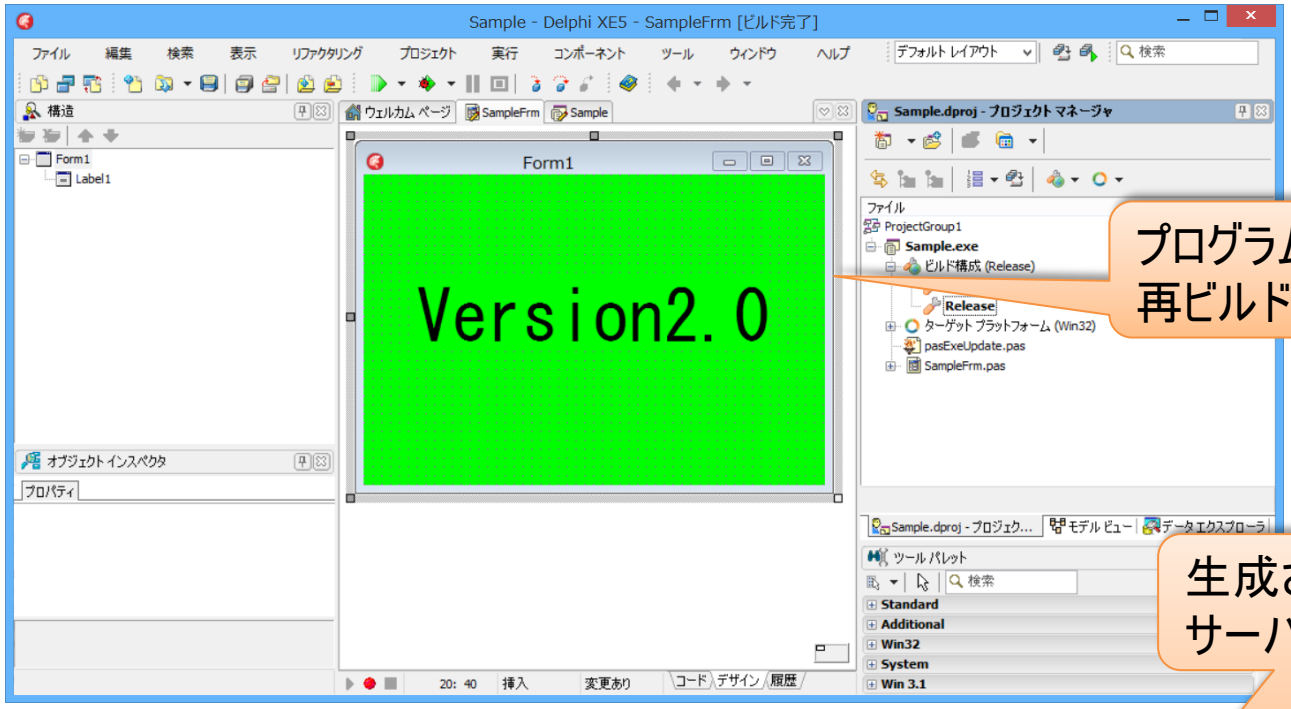
■ 動作デモ

- サーバーとクライアントのExe更新日付が同じ場合



■ 動作デモ

- プログラムを修正して、修正版Exeをサーバーへアップ

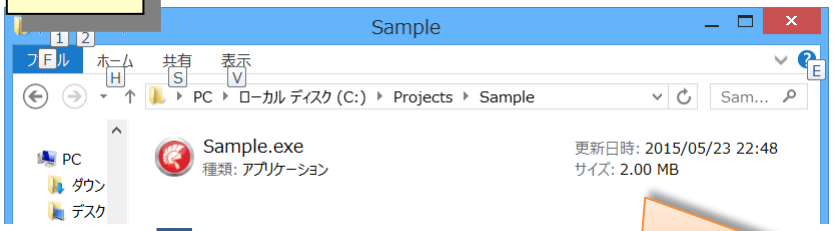


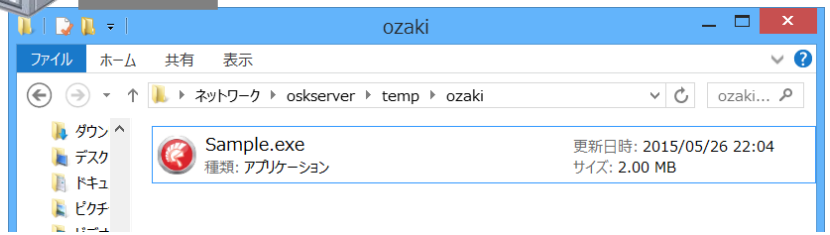
■ 動作デモ

- サーバーとクライアントの更新日付が異なる場合

【例】C:¥Projects¥Sample¥

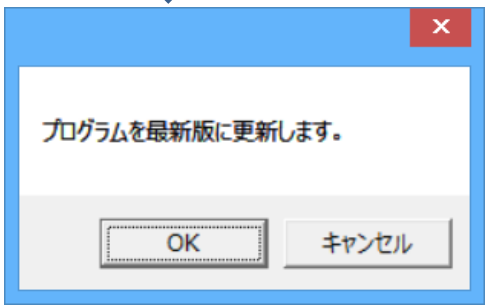
【例】¥¥server01¥Temp¥ozaki¥



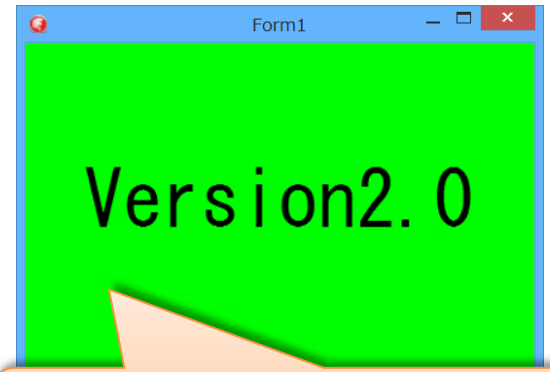


実行

クライアントの更新日付が古い。



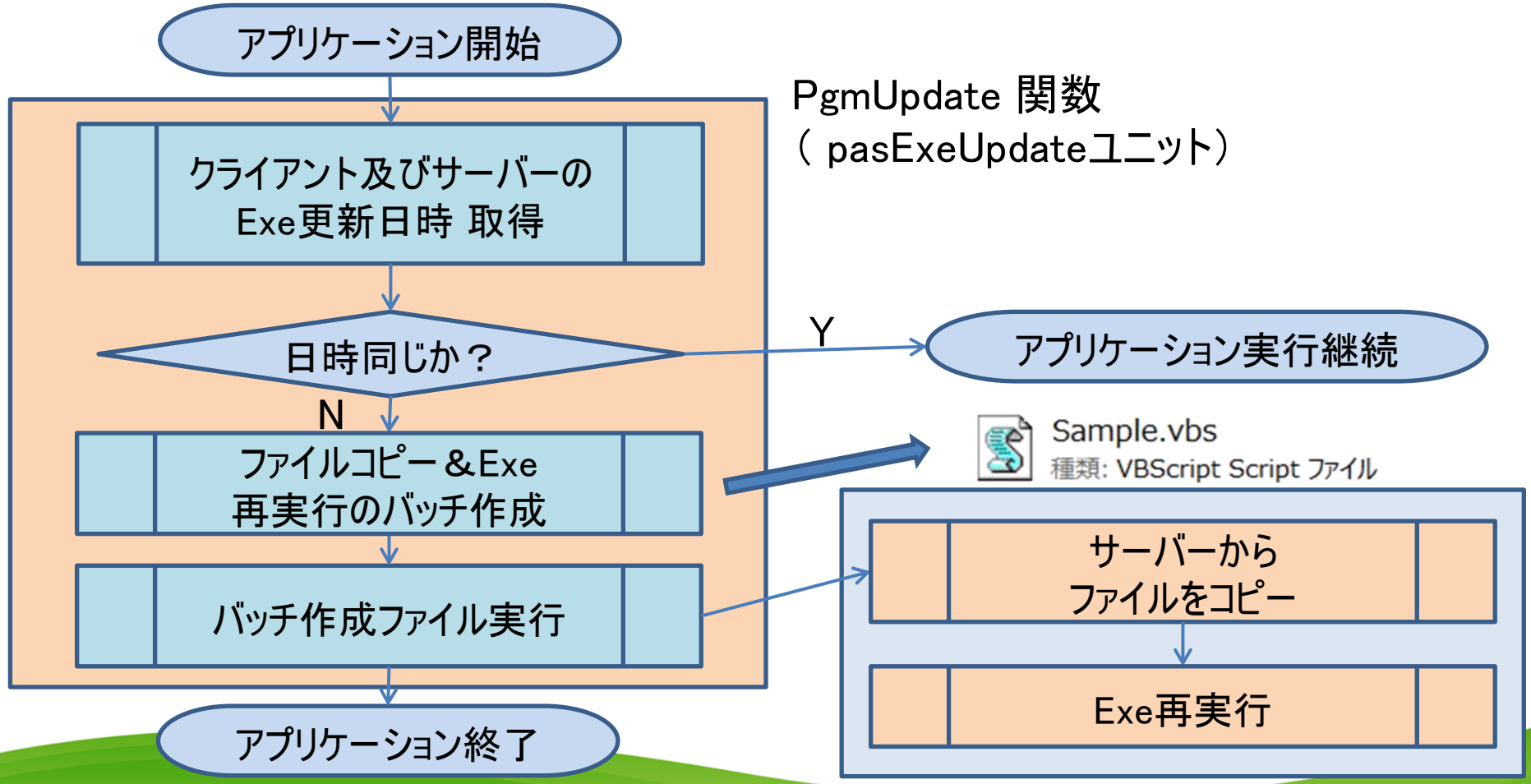
ファイルがコピー



自動的にプログラムが最新版に
置き換えられ、その後実行される。

■ Exeファイル置き換えの仕組み

- プログラム起動時に下記処理を実行
  - アプリケーション自身でファイルの置き換えができない為、置き換え用のバッチファイルをDelphiの中で自動作成して、そのバッチによりExeを置き換える。



■ メインルーチン

- Exe比較を行い、異なる場合バッチを作成し実行
  - バッチを実行した場合、Trueを返す

```
function PgmUpdate(AServerPath: String): Boolean;
var
  sExeName, sDest, sSource : String;
begin
  Result := False;
  //クライアント/サーバーのExe（フルパス）
  sExeName := ExtractFileName(ParamStr(0));
  sSource := IncludeTrailingPathDelimiter(AServerPath) + sExeName;
  sDest := ParamStr(0);
  //スクリプトファイルが既に存在する場合削除
  DeleteScript(sDest);
  //サーバーとクライアントのバージョンをチェック
  if FileCheck(sDest, sSource) then
  begin
    //スクリプトファイル作成
    MakeScript(sDest, sSource);
    //スクリプト実行
    ExecScript(sDest);
    Result := True;
  end;
end;
```

AServerPath - サーバーパス

ParasStr(0) ー
実行Exeファイル(フルパス)

①ファイルチェック

②バッチファイルの作成

③バッチファイルの実行

■ ① ファイルチェック

- FileAge関数を使用してタイムスタンプの比較を実施
  - 異なる場合Trueを返すサブルーチン

```
function FileCheck(AClientExe, AServerExe: String): Boolean;
var
  dClientDateTime: TDateTime;  // クライアントタイムスタンプ
  dServerDateTime: TDateTime;  // サーバータイムスタンプ
begin
  Result := False;

  //サーバーファイル存在チェック
  if not FileExists(AServerExe) then Exit;

  //タイムスタンプ取得
  FileAge(AClientExe, dClientDateTime);
  FileAge(AServerExe, dServerDateTime);

  //クライアント、サーバーのバージョン比較
  if dClientDateTime <> dServerDateTime then
    Result := True;
end;
```

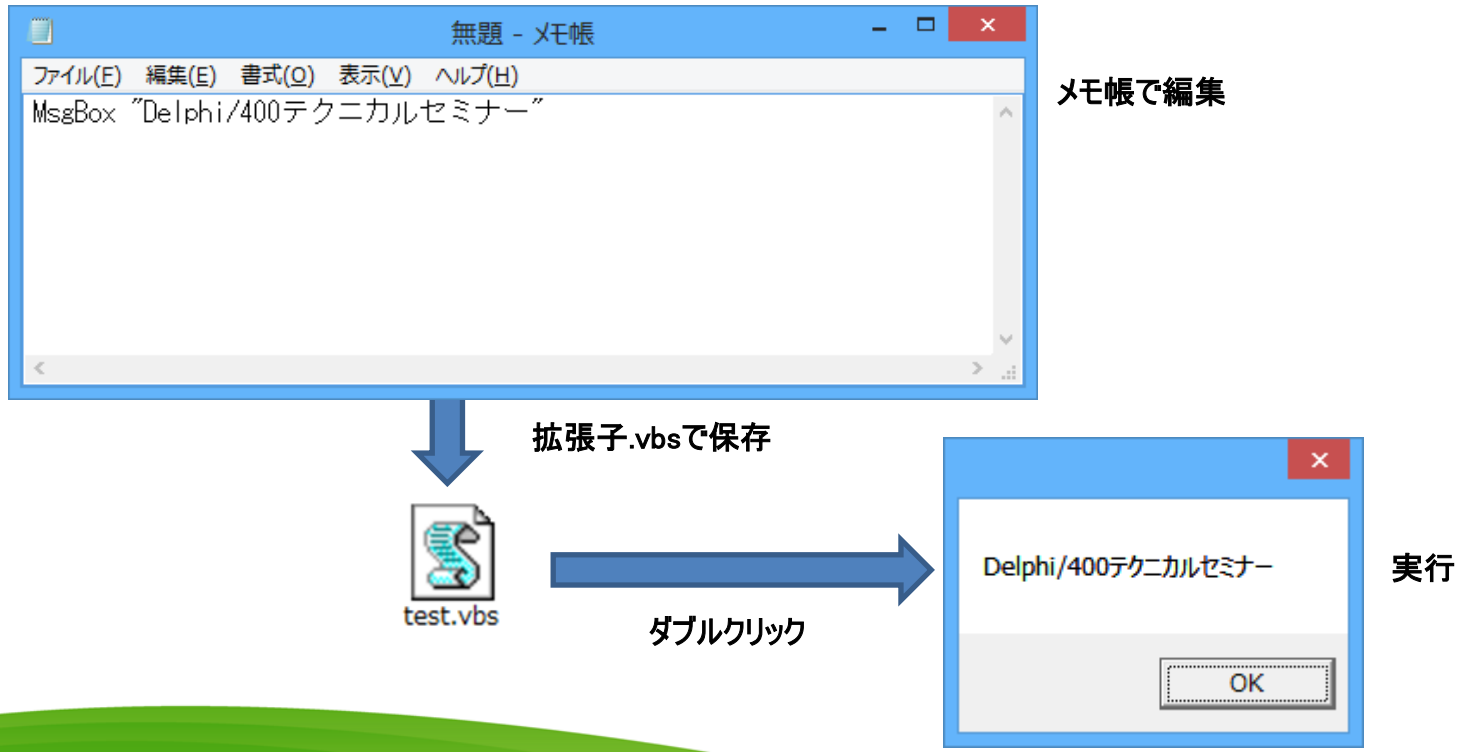
AClientExe – クライアントExe(フルパス)
AServerExe – サーバーExe(フルパス)

サーバーファイルが存在
しない場合チェック不要

FileAge関数で
タイムスタンプを取得

■ ② バッチファイルの作成

- VBScript とは？
  - バッチ処理が行えるスクリプト言語。 拡張子.vbs。
  - 従来のバッチファイル(.bat)より高機能で、メッセージの出力等も可能。
  - Windows単体で実行可能。



■ ② バッチファイルの作成

- TStringListを使用してバッチファイルを作成
  - Addメソッドで、コマンドを書込し、SaveToFileメソッドでファイルとして保存

```
procedure MakeScript(AClientExe, AServerExe: String);
var
  sScriptName: String;  //スクリプトファイル名
  sList: TStringList;
begin
  sScriptName := ChangeFileExt(AClientExe, '.vbs');
  sList := TStringList.Create;
  try
    sList.Add('Ret = MsgBox("プログラムを最新版に更新します。", vbOKCancel)');
    sList.Add('If Ret = vbOK Then');
    sList.Add('Set fso = CreateObject("Scripting.FileSystemObject")');
    sList.Add('fso.CopyFile "' + AServerExe + '", "' + AClientExe + '"');
    sList.Add('Set fso = Nothing');
    sList.Add('Set ws = CreateObject("WScript.Shell")');
    sList.Add('ws.Run "' + AClientExe+ '"');
    sList.Add('Set ws = Nothing');
    sList.Add('End If');
    sList.SaveToFile(sScriptName);
  finally
    sList.Free;
  end;
end;
```

ファイル名を
[Exeファイル名].vbs とする

メッセージ表示

ファイルコピー

Exeファイルの実行

バッチファイルの保存

■ ③ バッチファイルの実行

- ShellExecute を使用してプログラムを起動
  - 外部プログラムや、関連付けファイルのオープン、ブラウザ起動が可能

```
uses …, ShellAPI;

procedure ExecScript(AClientExe: String);
var
  sClientPath: String;  //実行パス
  sScriptName: String;  //スクリプトファイル名
begin
  sClientPath := ExtractFilePath(AClientExe);
  sScriptName := ChangeFileExt(AClientExe, '.vbs');

  //スクリプト実行
  ShellExecute(0, 'Open', PChar(sScriptName),'' , PChar(sClientPath), 0);
end;
```

ShellAPIユニットを追加

ファイル名を
[Exeファイル名].vbs とする

バッチファイルを実行

関連付けファイル起動	`ShellExecute(0, 'open', PChar('C:¥Dir¥Book1.xlsx'), nil, nil, SW_SHOWNORMAL);`
ブラウザ起動	`ShellExecute(0, 'open', PChar('www.migaro.co.jp'), nil, nil, SW_SHOW);`

■ バージョンアップユニットの利用

- 既存プログラムに組み込んで利用。
  - 付録のCD-ROMにユニットソースを添付。

- 今回サンプルソースをベースに拡張。
  - サーバーフォルダの場所
    - ➢ AS/400上のデータから取得できるようにする。
  - バッチファイルでのファイルコピー
    - ➢ Delphiで作成したExeを起動してコピーできるようにする。
      （処理状況の表示など、より細かな制御が可能。）
  - 関連ファイルの一括コピー
    - ➢ Exeファイル以外の関連ファイルを同時にコピーする。

- Exeファイルのスムーズな置き換えに是非ご活用ください！

まとめ

■ まとめ

- アプリケーションのレスポンスを改善したい。
  - シンプルなマルチスレッドによるレスポンスタイムの向上方法をご紹介
    1. CreateAnonymousThreadを使ったスレッド処理
    2. Synchronizeを使ったVCL操作
- プロジェクトを効率よくメンテナンスしたい。
  - DLLによるプロジェクト分割手法をご紹介
    1. DLL作成方法
    2. フォームを持つDLL作成方法
    3. 動的なDLLリンク方法
    4. データモジュール活用方法
- プログラムの入れ替えをシンプルに行いたい。
  - Exe自身に組み込むバージョンアップテクニックのご紹介
    1. PgmUpdate関数のご紹介

ご清聴ありがとうございました。