【セッションNo．4】

Delphi/400技術セッション

開発者が知りたい実践プログラミングテクニック！

株式会社ミガロ.
RAD事業部　営業・営業推進課
尾崎　浩司

■ 今回のテーマ

- **課題を解決する為に、工夫したテクニックを厳選してご紹介！**

- テストの品質向上や問題点の調査を効率よく行いたい。
  - デバッグをする際に、もっと効率よく実施できないか？
  - メモリーリークを防ぐことはできないか？
  - 他の担当者が作成したプログラムを意図通りに使用できるか？

『１．Delphi/400デバッグテクニック』にて効果的なデバッグ手法をご紹介！

- ローカルデータベースを使って処理の改善を検討したい。
  - オフラインでも動作するアプリが作成できないか？
  - INIファイルやレジストリを使わずにアプリの設定情報を端末保持できないか？
  - Paradoxに変わるローカルデータベースが使用できないか？

『２．ローカルデータベースの活用』にて最新のローカルDB活用法をご紹介！

1．Delphi/400 デバッグテクニック

■ デバッグとは？

- プログラムの中のバグ（誤り）を発見して、正しく動くように修正すること。
- デバッグを支援するプログラム　：　デバッガ

＜一般的なデバッガ機能＞

① **ブレークポイント**
- ソースコード中に設定し、実行時の流れを一時停止する機能。これにより、任意の位置での実行状況（変数の値等）が確認できる。

② **ステップ実行**
- 処理を一時停止したのちに、１ステップずつソースコードを実行する機能。これにより、ステップごとにソースコードを追いかけながら実行することで、ロジックに問題点がないか確認できる。

③ **変数やプロパティ値の確認**
- その時点における変数やプロパティの状態（値）を確認することで、想定どおりの状態遷移がされているか確認できる。

■ アプリケーションの開発におけるデバッグの位置づけ

- プログラム開発工程（PG）
  - デバッグは、主に単体テストにおけるホワイトボックステスト時に実施する。

画面設計

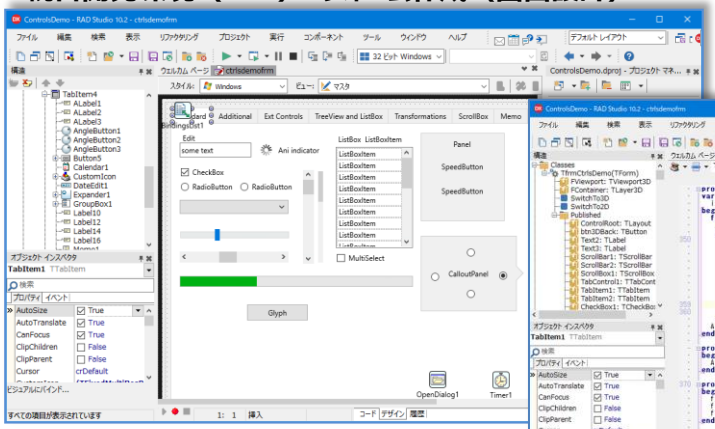
ロジックコーディング

コンパイル

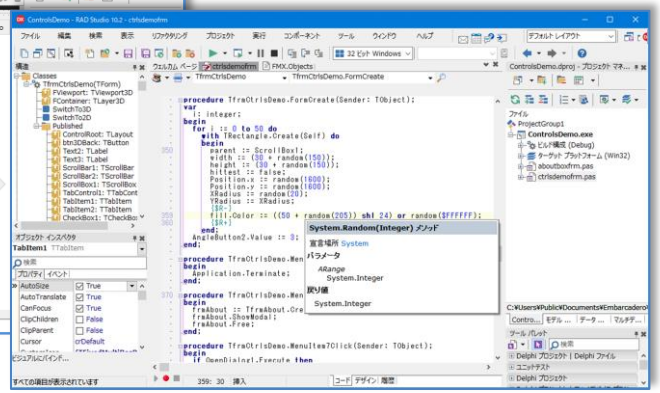
単体テスト

完成

統合開発環境（IDE） フォーム作成（画面設計）



ロジックコーディング



テストケース（仕様）にもとづくテスト実行

第18回テクニカルセミナー
『ユニットテストフレームワークの活用方法』参照

ソースレベルでの実行確認

デバッグ （今回のテーマ）

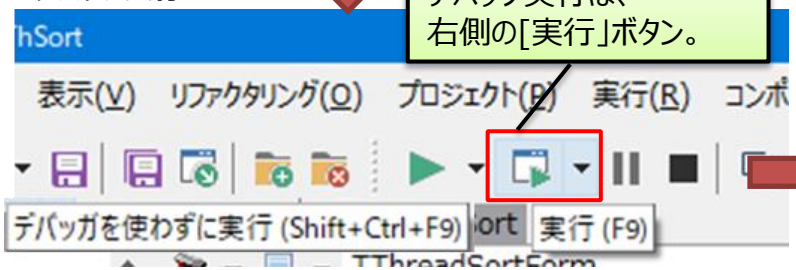
Delphi/400におけるデバッグの基本

- ①ブレークポイント
  - 設定したい行にカーソルをあわせて、[F5]キーを押下することで設定/解除。
  - ソース行の左側をマウスでクリック。

コードエディタ

```
procedure TThreadSortForm.StartBtnClick(Sender: TObject);
begin
  RandomizeArrays;
  ThreadsRunning := 3;
  with TBubbleSort.Create(BubbleSortBox, BubbleSortArray) do
    OnTerminate := ThreadDone;
  with TSelectionSort.Create(SelectionSortBox, SelectionSortArray) do
    OnTerminate := ThreadDone;
  with TQuickSort.Create(QuickSortBox, QuickSortArray) do
    OnTerminate := ThreadDone;
  StartBtn.Enabled := False;
end;

procedure TThreadSortForm.RandomizeArrays;
var
  I: Integer;
```

デバッグ実行



デバッグ実行は、
右側の[実行]ボタン。

プログラム実行時、ブレークポイントに到達すると、
該当箇所に緑色の矢印がつく。（該当箇所実行前に停止）

```
procedure TThreadSortForm.StartBtn
begin
  RandomizeArrays;
  ThreadsRunning := 3;
  with TBubbleSort.Create(BubbleS
    OnTerminate := ThreadDone;
```

■ Delphi/400におけるデバッグの基本

- ②ステップ実行
  - [F8]：次のステップまで実行。手続きや関数を呼び出した場合、その処理の実行が終わり次のステップにきて停止。　（ステップ実行）
  - [F7]：次のステップまで実行。手続きや関数を呼び出した場合、その処理の中の最初のステップにきて停止。（トレース実行）

手続き（サブルーチン）を呼び出し

```
procedure TThreadSortForm.StartBtnCl
begin
  RandomizeArrays;
  ThreadsRunning := 3;
  with TBubbleSort.Create(BubbleSortBox, BubbleS
    OnTerminate := ThreadDone;
  with TSelectionSort.Create(SelectionSortBox, S
    OnTerminate := ThreadDone;
```

F8

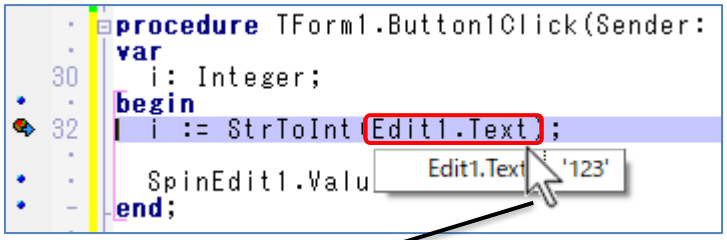
```
procedure TThreadSortForm.StartBtnClick(Sender:
begin
  RandomizeArrays;
  ThreadsRunning := 3;
  with TBubbleSort.Create(BubbleSortBox, BubbleS
    OnTerminate := ThreadDone;
  with TSelectionSort.Create(SelectionSortBox, S
    OnTerminate := ThreadDone;
```

F7

```
procedure TThreadSortForm.RandomizeArrays;
var
  I: Integer;
begin
  if not ArraysRandom then
  begin
    Randomize;
    for I := Low(BubbleSortArray) to High(Bubbl
      BubbleSortArray[I] := Random(170);
    SelectionSortArray := BubbleSortArray;
    QuickSortArray := BubbleSortArray;
    ArraysRandom := True;
    Repaint;
  end;
end;
```

■ Delphi/400におけるデバッグの基本

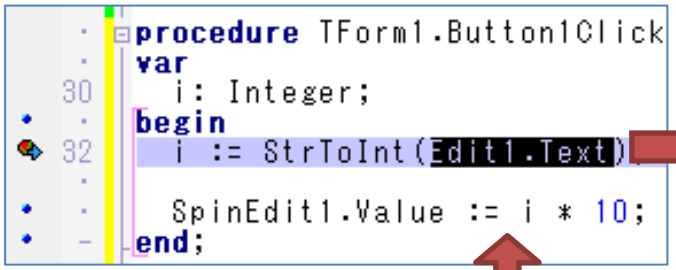
- ③変数やプロパティ値の確認
  - ツールチップ式評価：停止中に変数名やプロパティ名の上にカーソルを移動。

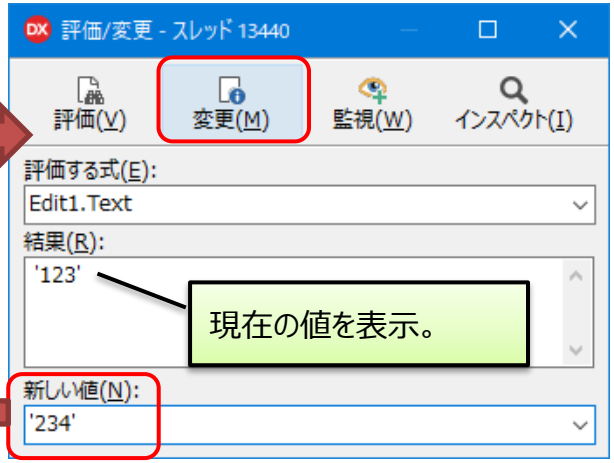


プロパティ名にカーソルを移動
現在の設定値を表示。

レコード型の変数にカーソルを移動
各要素を展開して表示。

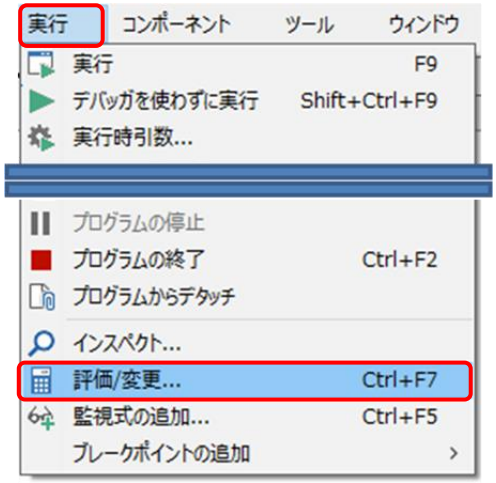
  - 評価/変更：変数や式を評価として参照。任意の値に変更することも可能。[Ctrl]+[F7]





現在の値を表示。

値を変更すると、
デバッグにおいて、仮にその値で
実行した時の結果を確認できる。

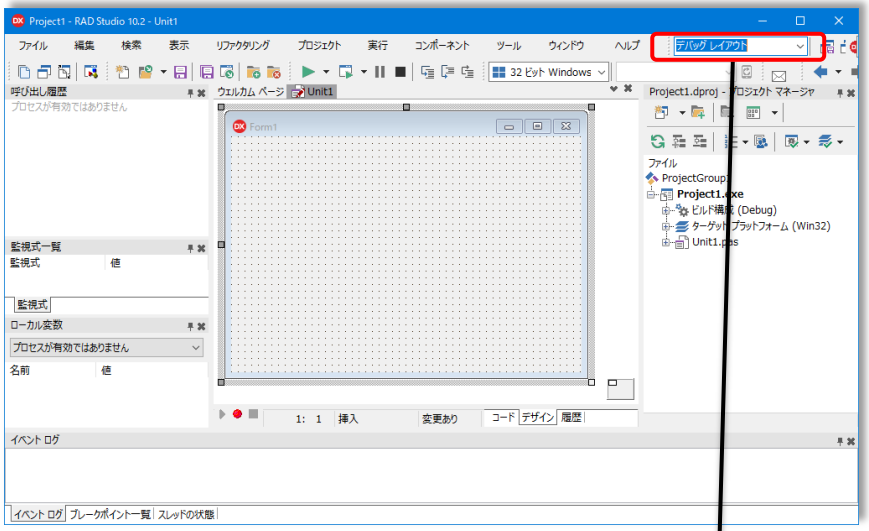


Delphi/400デバッグテクニックのポイント

- 効率的なデバッグを実現する応用テクニックをご紹介！
  - （１）デバッグ効率向上テクニック
  - （２）マルチスレッドアプリのデバッグ
  - （３）メモリーリーク調査方法
  - （４）言語レベルのデバッグ機能

■ （１）デバッグ効率向上テクニック

- 便利なデバッグ機能を活用
  - デバッグウィンドウを使用することで、より効率的なデバッグが行える。



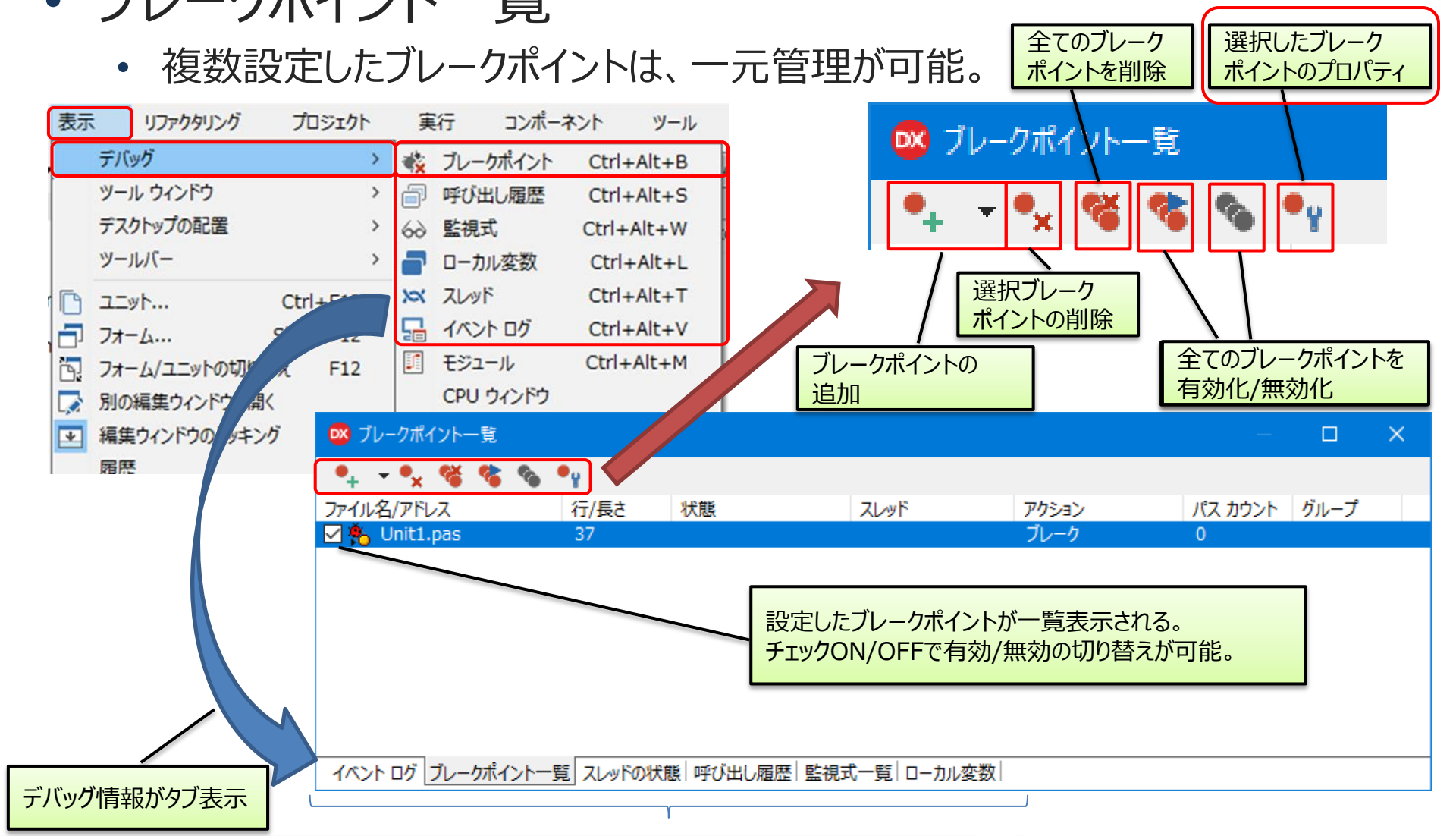
IDEのレイアウトを「デバッグレイアウト」にすると
デバッグウィンドウがデフォルトで表示される。

- 今回ご紹介する機能
  - ブレークポイントの一元管理方法
  - ブレーク条件を指定したデバッグ方法
  - 呼出履歴およびローカル変数ウィンドウを使用した実行履歴の追跡方法
  - イベントログを使用した実行状況の確認方法

効率的なデバッグを行うためのテクニックをご紹介！

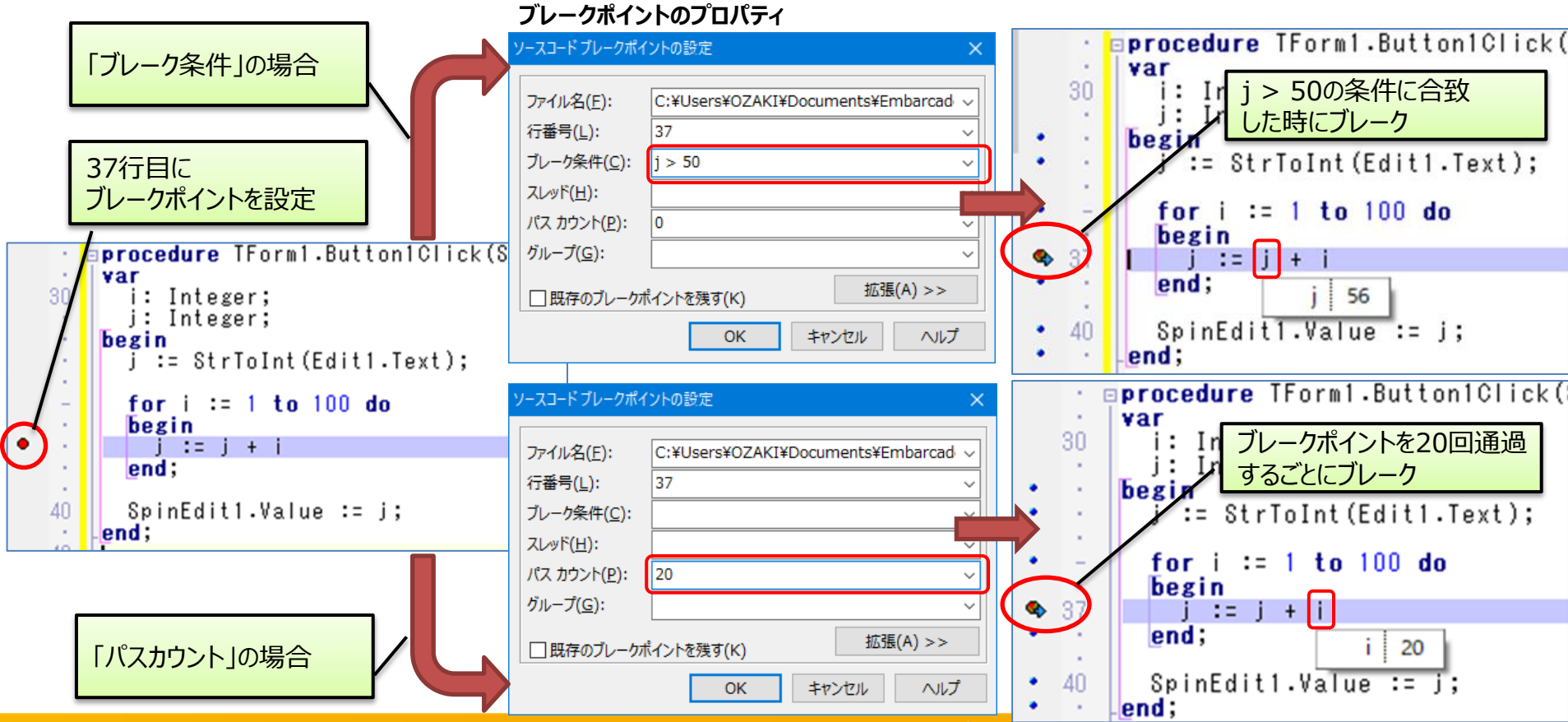
■ （１）デバッグ効率向上テクニック

- ブレークポイント一覧
  - 複数設定したブレークポイントは、一元管理が可能。



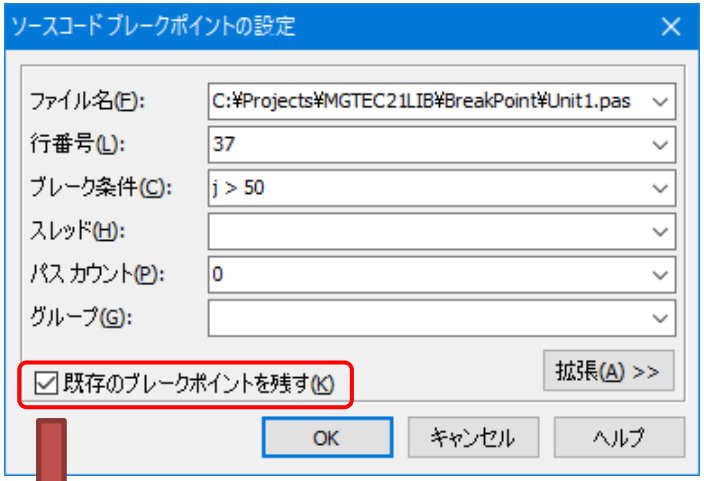
■ （１）デバッグ効率向上テクニック

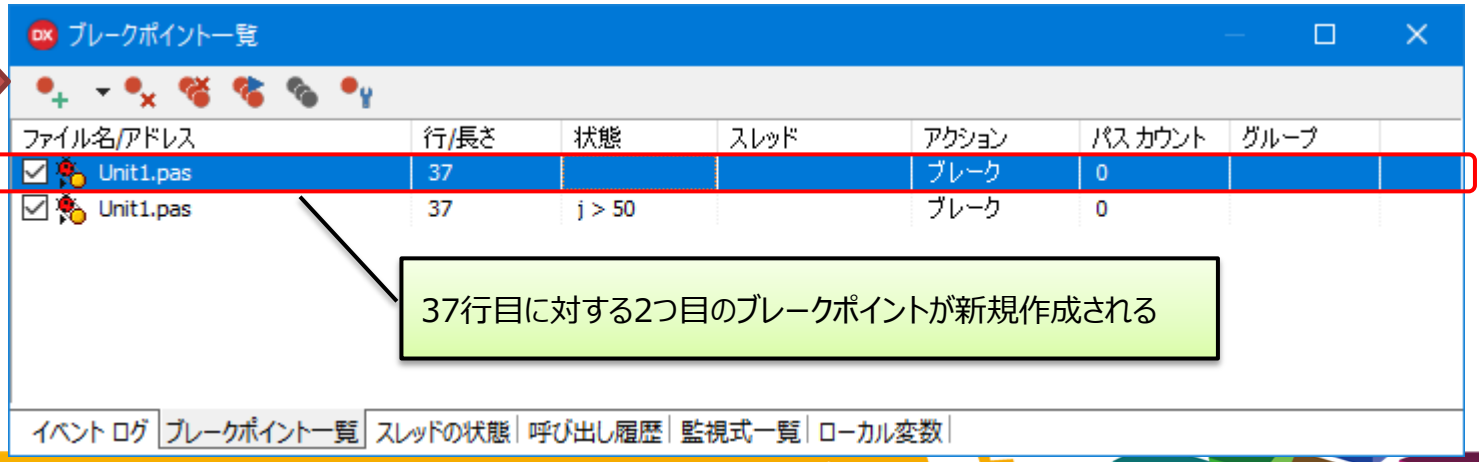
- ブレークポイントのプロパティ
  - ブレーク条件：条件に合致した時にのみブレークさせる。
  - パスカウント　：ブレークポイントを指定回数通過したときにブレークさせる。



■ （１）デバッグ効率向上テクニック

- 複合条件のブレークポイント
  - 同じ行に対して、条件の異なる2つのブレークポイントを設定可能。





■ （１）デバッグ効率向上テクニック

- 明示的なデバッガ例外の無視

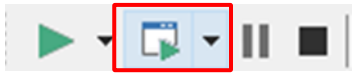
明示的にデバッガ例外を無視することが可能！

```
procedure TForm1.Button1Click(Sender:
var
  i: Integer;
begin
  try
    i := StrToInt(Edit1.Text);
  except
    i := -1;
  end;

  SpinEdit1.Value := i;

end;
```

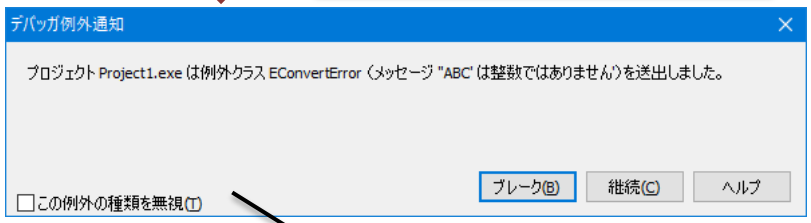
エラー（例外）を考慮したプログラム

例外考慮箇所の前後にブレークポイントを設定

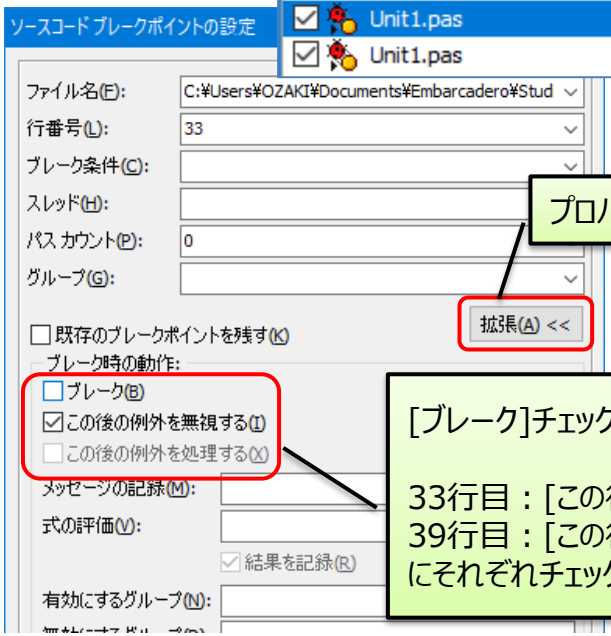
```
try
  i := StrToInt(Edit1.Text);
except
  i := -1;
end;

SpinEdit1.Value := i;
```



デバッグ実行で、Edit1.Text に
‘ABC’をセットしてボタンをクリック。



エラーの発生を考慮してても、例外により
デバッガ例外通知画面が表示されてしまう。



プロパティから[拡張]を選択

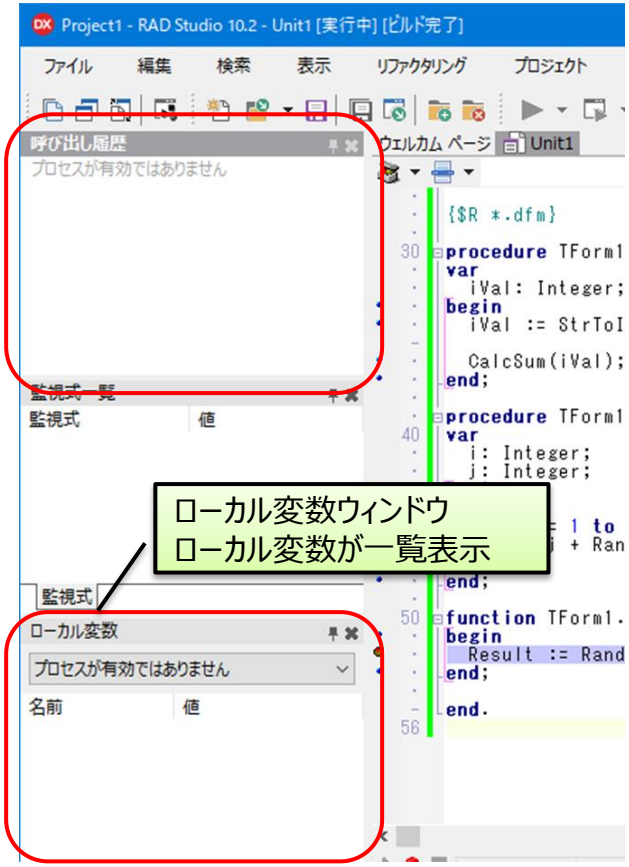
[ブレーク]チェックをOFF

33行目：[この後の例外を無視する]
39行目：[この後の例外を処理する]
にそれぞれチェック。

■ （１）デバッグ効率向上テクニック

- 呼び出し履歴
  - ブレークした時にそれまでに実行された順序を履歴で表示。実行経路がわかる。

デバッグレイアウト時の画面



ソースコード例

Button1のClickイベント

```
30  procedure TForm1.Button1Click(Sender: TObject);
    var
      iVal: Integer;
    begin
      iVal := StrToInt(Edit1.Text);

      SpinEdit1.Value := CalcSum(iVal);
    end;

    function TForm1.CalcSum(AValue: Integer): Integer;
40  var
      i: Integer;
      j: Integer;
    begin
      j := 1;
      for i := 1 to AValue do
        j := j + RandomVal(i);

      Result := j;
    end;
50
    function TForm1.RandomVal(AValue: Integer): Integer;
    begin
      Result := Random(AValue) + 1;
    end;
```

53行目に
ブレークポイントを設定

（１）デバッグ効率向上テクニック

実行してブレークポイントで停止

```
function TForm1.RandomVal(AValue: Integer): Integer;
begin
  Result := Random(AValue) + 1;
end;
```

呼び出し履歴 - スレッド 15656

- Unit1.TForm1.RandomVal(1)
- Unit1.TForm1.CalcSum(10)
- Unit1.TForm1.Button1Click($154F8C0)
- Vcl.Controls.TControl.Click
- Vcl.StdCtrls.TCustomButton.Click
- Vcl.StdCtrls.TCustomButton.CNCommand(???)
- Vcl.Controls.TControl.WndProc((48401, 1830, 41
- Vcl.C roc((48401, 1830,
- Vcl.St dProc((48401, 18
- Vcl.C ???,???,4130598)
- Vcl.Controls.DoControlMsg(???,(値なし))

実行順序が履歴として表示。

ローカル変数 - スレッド 15656

Unit1.TForm1.CalcSum(10)

名前	値
Self	($1584910, 'Form1', 0, $15C6EA...
AValue	10
Result	5348274
i	1
j	1

```
function TForm1.CalcSum(AValue: Integer): Integer;
var
  i: Integer;
  j: Integer;
begin
  j := 1;
  for i := 1 to AValue do
    j := j + RandomVal(i);

  Result := j;
end;
```

それぞれのメソッド内のローカル変数の様子が確認できる。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  iVal: Integer;
begin
  iVal := StrToInt(Edit1.Text);

  SpinEdit1.Value := CalcSum(iVal);
end;
```

ローカル変数 - スレッド 15656

Unit1.TForm1.RandomVal(1)

名前	値
Self	($1584910, 'Form1', 0, $15C6EA...
AValue	1
Result	2

ローカル変数ウィンドウでは、実行場所でのその時点のローカル変数の一覧が表示。

履歴をクリックすると呼出位置に移動。

ローカル変数 - スレッド 15656

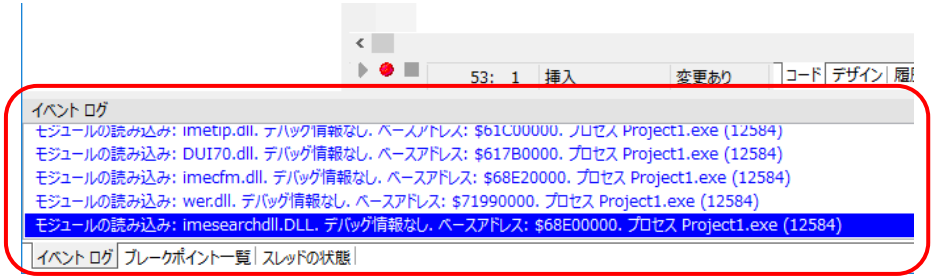
Unit1.TForm1.Button1Click($154F8C0)

名前	値
Self	($1584910, 'Form1', 0, $15C6EA...
Sender	()
iVal	10

■ （１）デバッグ効率向上テクニック

- イベントログ
  - デバッグ中に発生した各種メッセージをログとして表示。
  - デバッグ実行中に、OutPutDebugStringで独自のログを出力することが可能。

デバッグレイアウト時の画面（画面下部）

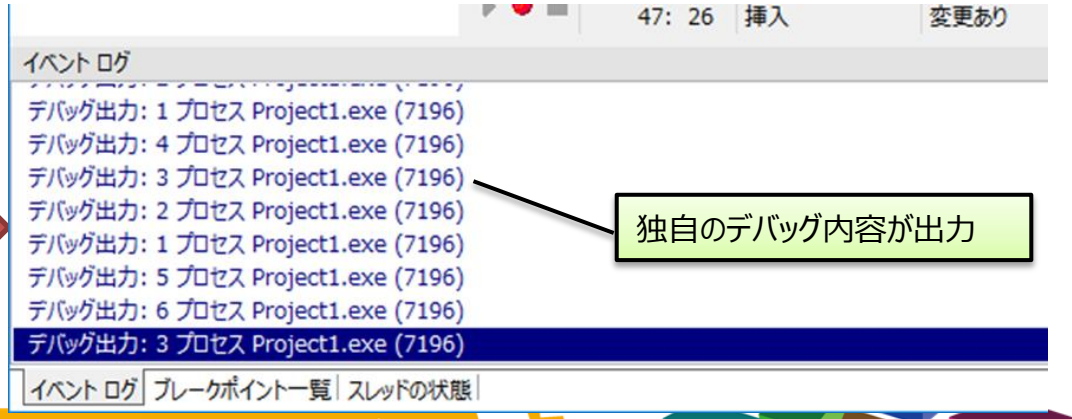


OutPutDebugString手続き

イベントログ上に、引数の文字を出力
実行時の値をログとして出力できる。

```
function TForm1.RandomVal(AValue: Integer): Integer;
begin
  Result := Random(AValue) + 1;
  OutputDebugString(PWideChar(IntToStr(Result)));
end;
```

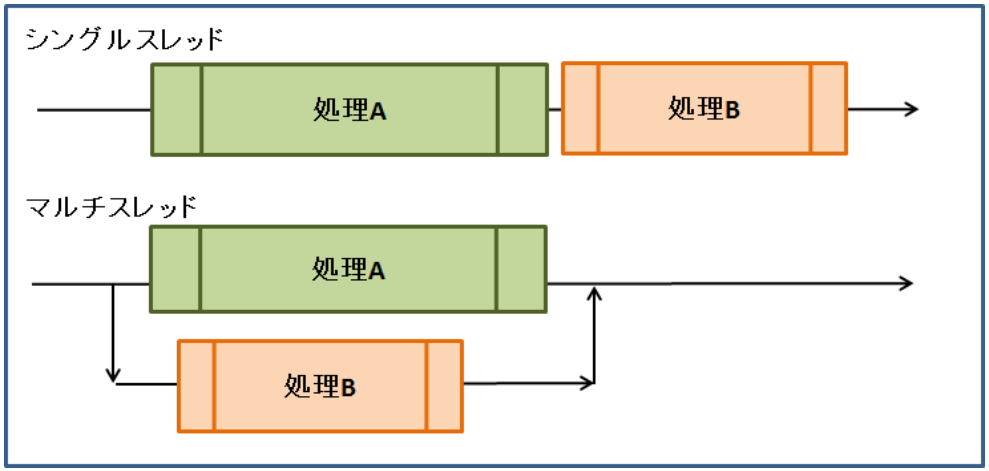
アプリケーションを実行



独自のデバッグ内容が出力

■ （２）マルチスレッドアプリのデバッグ

- シングルスレッドとマルチスレッド
  - プログラムの処理の流れが一本のものを「シングルスレッド」、処理効率を向上する為に同時に複数の処理を行うものを「マルチスレッド」という。

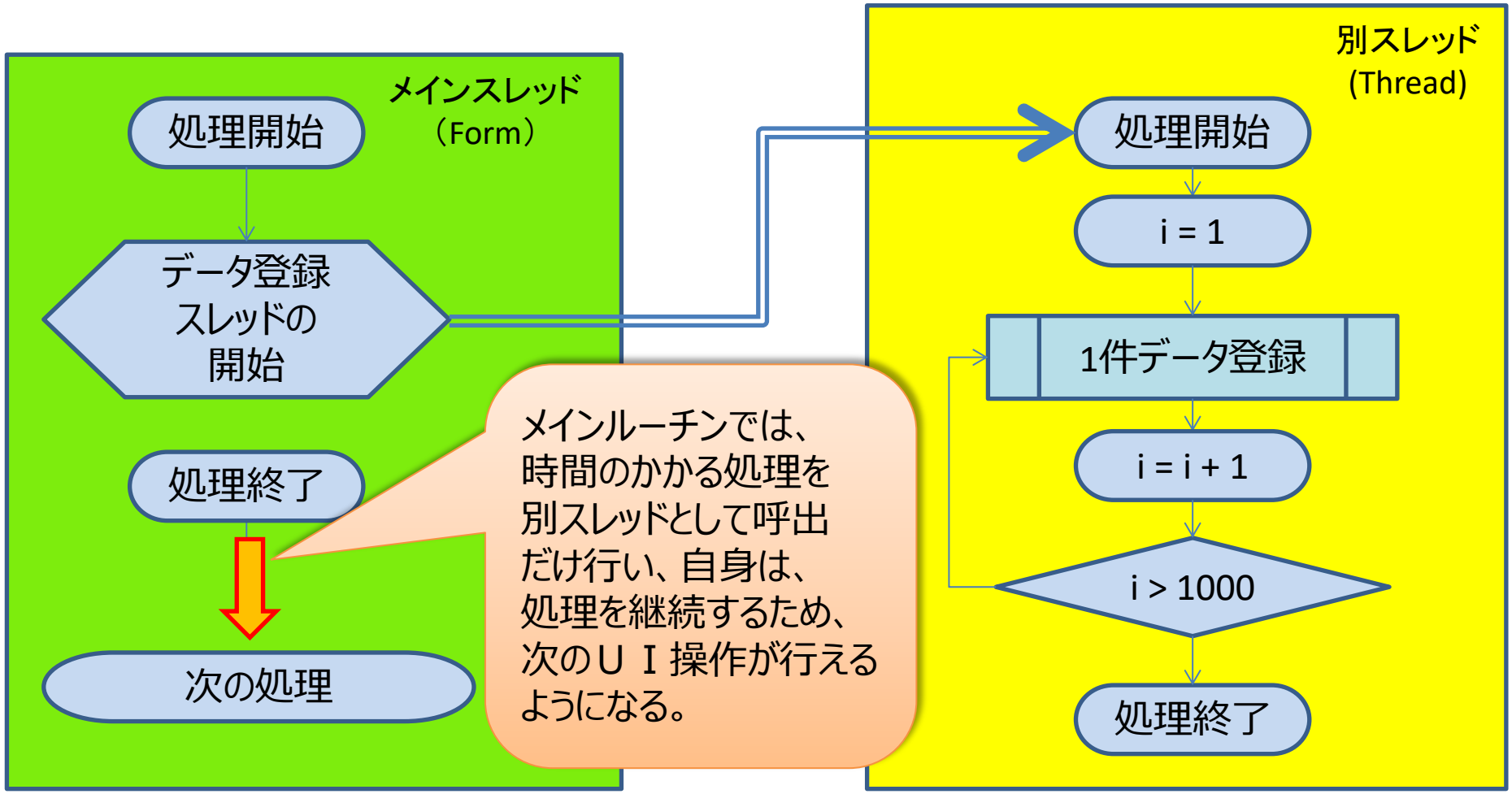


  - マルチスレッドの場合、デバッグを行う際に、どのスレッドに対してデバッグを行うかを意識しなければならないときがある。

複数の処理（スレッド）がある場合のデバッグ方法をご紹介！

■ （２）マルチスレッドアプリのデバッグ

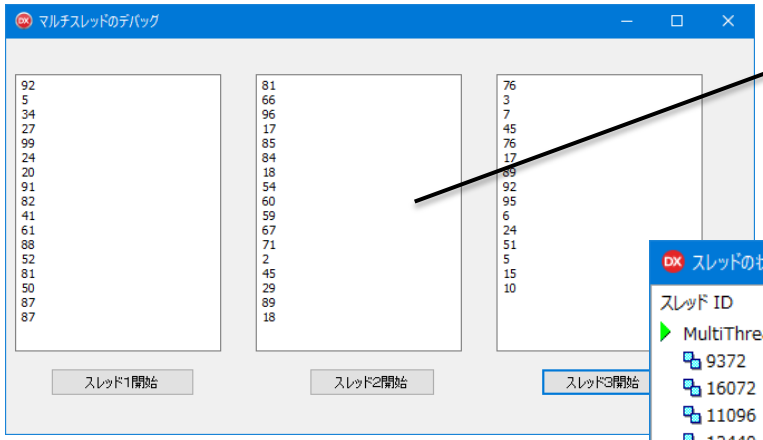
- マルチスレッドプログラムの概要
  - 実行時に、スレッドを追加すると、別スレッドとして異なるスレッドIDが付与。



■ （２）マルチスレッドアプリのデバッグ

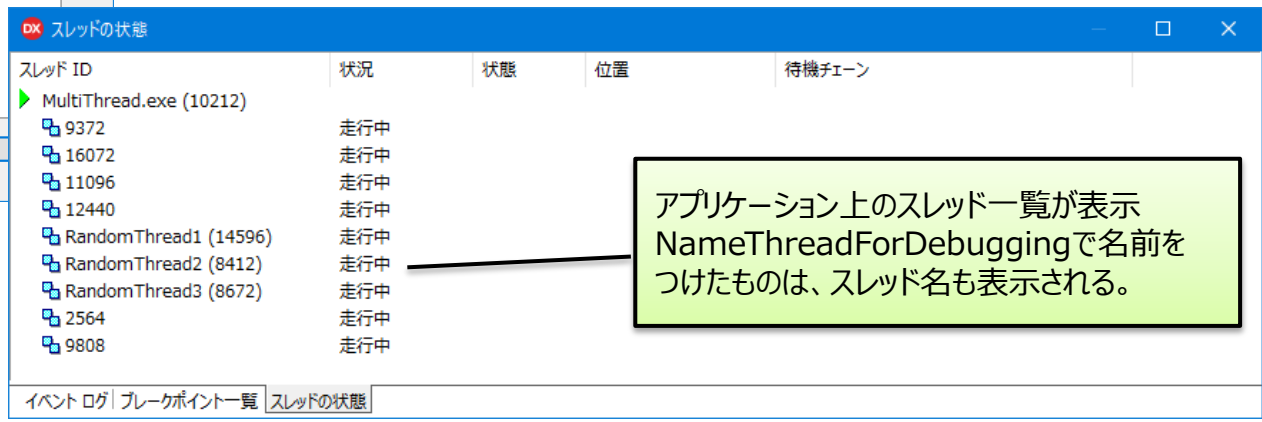
- スレッドの状態
  - 実行中のスレッドが一覧表示。 スレッドIDやスレッド名を確認できる。
  - 特定のスレッドを凍結（停止）させることも可能。
  - NameThreadForDebugging(AThreadName: String)
    - デバッグ用にスレッドに名前をつけることが可能。

マルチスレッドアプリケーション例



メモコンポーネントにランダムな値を
セットするプログラム。
各メモごとに異なるスレッドで実行

スレッドの状態



アプリケーション上のスレッド一覧が表示
NameThreadForDebuggingで名前を
つけたものは、スレッド名も表示される。

■ （２）マルチスレッドアプリのデバッグ

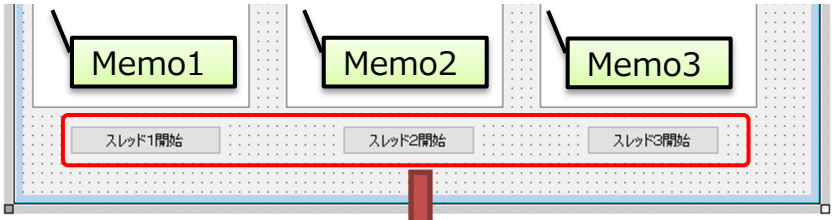
スレッドの宣言部

```
type
  TRandomThread = class(TThread)
  private
    FThreadName: String;
    FMemo: TMemo;
    FValue: Integer;
    procedure SetMemo;
  protected
    procedure Execute; override;
  public
    constructor Create(AMemo: TMemo;
      AThreadName: String); virtual;
  end;
```

処理本体

コンストラクター

フォーム




```
procedure TfrmMain.BtnThread1StartClick(Sender: TObject);
begin
  TRandomThread.Create(Memo1, 'RandomThread1');
end;

procedure TfrmMain.BtnThread2StartClick(Sender: TObject);
begin
  TRandomThread.Create(Memo2, 'RandomThread2');
end;

procedure TfrmMain.BtnThread3StartClick(Sender: TObject);
begin
  TRandomThread.Create(Memo3, 'RandomThread3');
end;
```

スレッドを実行

スレッドの実装部

```
{ TRamdomThread }

constructor TRandomThread.Create(AMemo: TMemo;
  AThreadName: String);
begin
  FThreadName := AThreadName;
  FMemo := AMemo;

  inherited Create(False);
  FreeOnTerminate := True;
end;

procedure TRandomThread.Execute;
begin
  inherited;
  //スレッドにデバッグ用の名前を付与
  NameThreadForDebugging(FThreadName);

  while not Terminated do
  begin
    //乱数でセットする値を決定
    FValue := Random(100);

    //メモに値をセット
    Synchronize(SetMemo);
    Sleep(200);
  end;
end;

procedure TRandomThread.SetMemo;
begin
  FMemo.Lines.Add(IntToStr(FValue));
end;
```

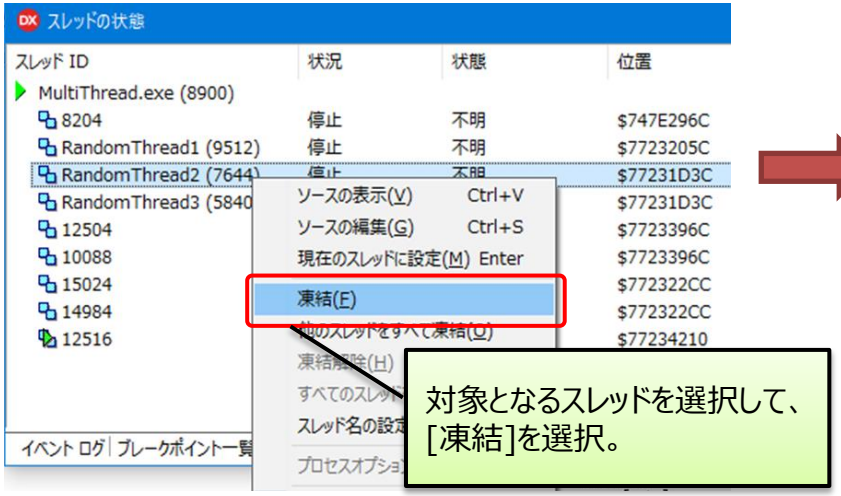
スレッドをすぐに実行する。（Executeが呼出）

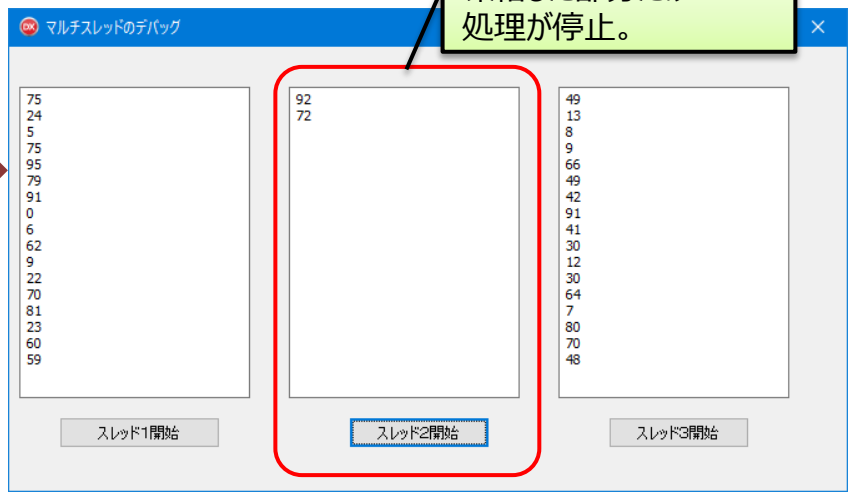
スレッド名をセット

メモコンポーネントに数値を一行追加。

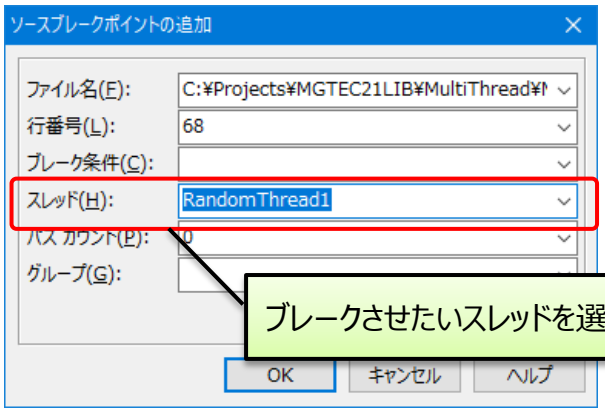
■ （２）マルチスレッドアプリのデバッグ

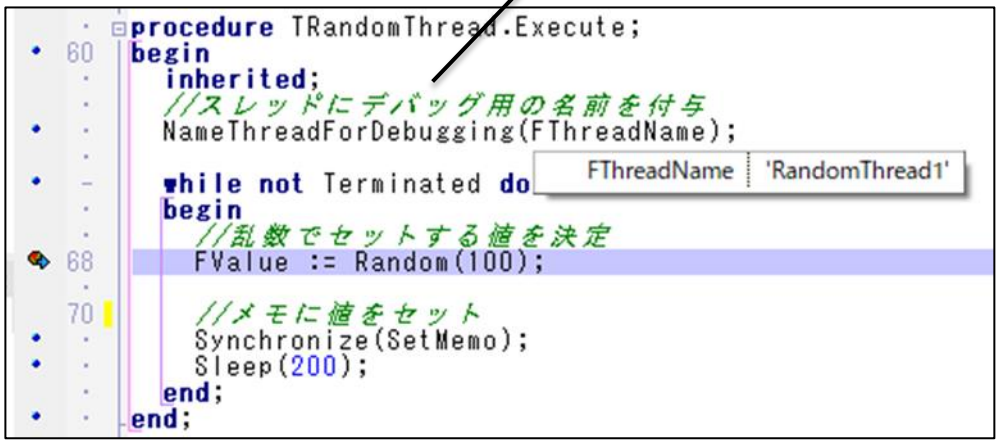
- スレッドの凍結





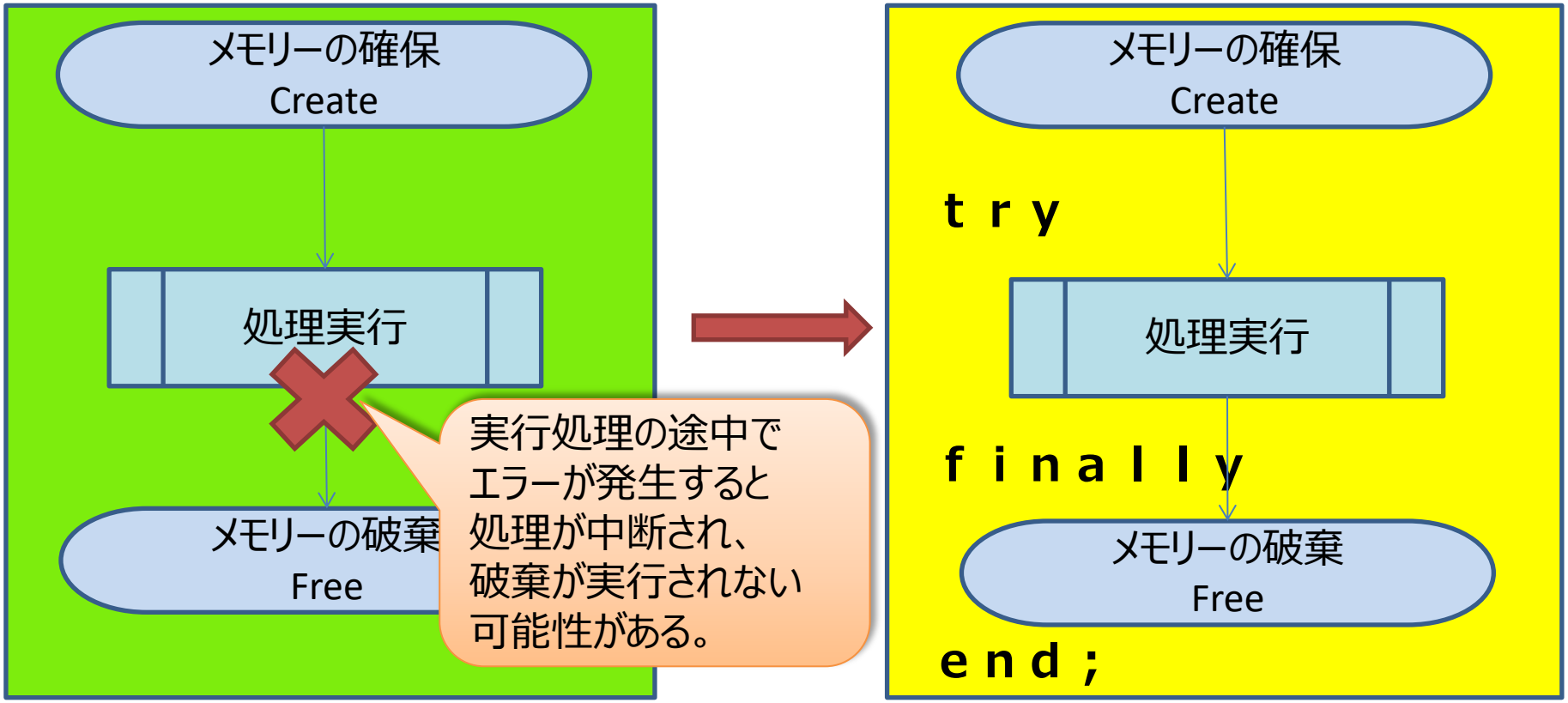
- スレッド単位のデバッグ





■ （３）メモリーリーク調査方法

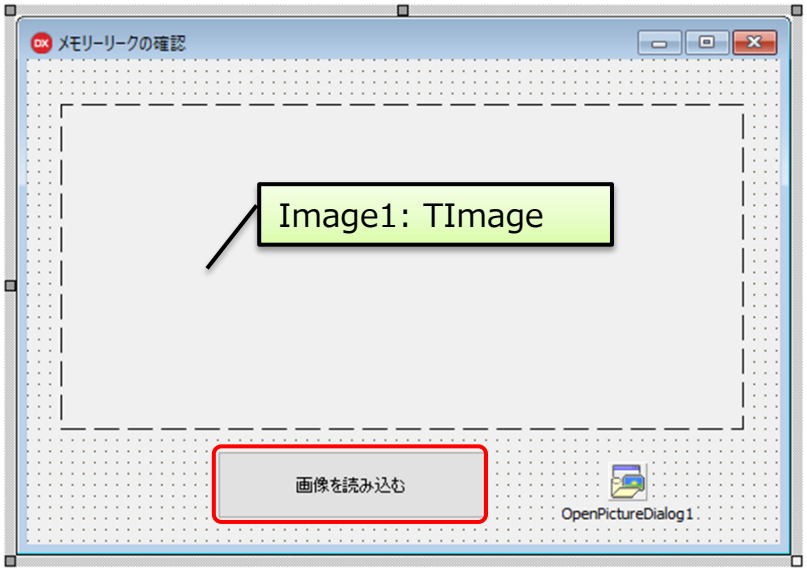
- メモリーリーク
  - プログラムの中で明示的に確保したメモリー領域を使用後に解放せずに残してしまうために、メモリーの空き領域が減っていく現象。



メモリリークがあるかどうかをデバッグする方法をご紹介！

■ （3）メモリーリーク調査方法

- サンプルプログラム



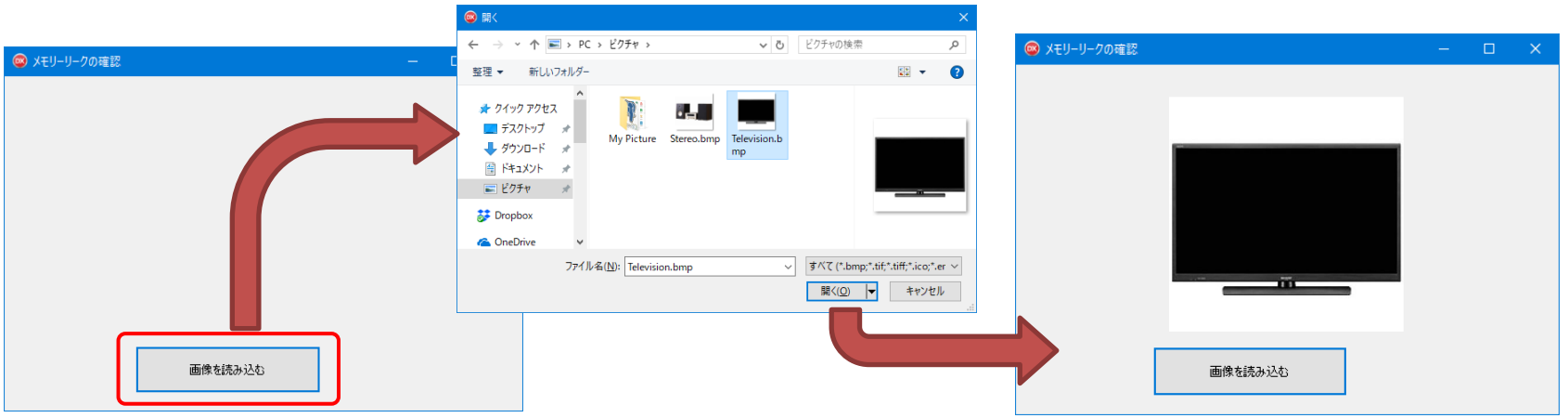
ソースコード

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then Exit;

  FBitMap := TBitmap.Create;
  try
    FBitMap.LoadFromFile(OpenPictureDialog1.FileName);

    Image1.Picture.Assign(FBitMap);

  finally
    FBitMap.Free;
  end;
end;
```

メモリー確保

メモリー破棄

実行例



■ （３）メモリーリーク調査方法

- メモリーリークの調査方法
  - ReportMemoryLeaksOnShutdown
    プログラム停止時にメモリーリークが存在する場合、報告するオプション。

プロジェクトファイル

```
program Project1;

uses
  Vcl.Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  ReportMemoryLeaksOnShutdown := True;

  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

メモリーリークが発生した場合、レポートを表示するオプション。

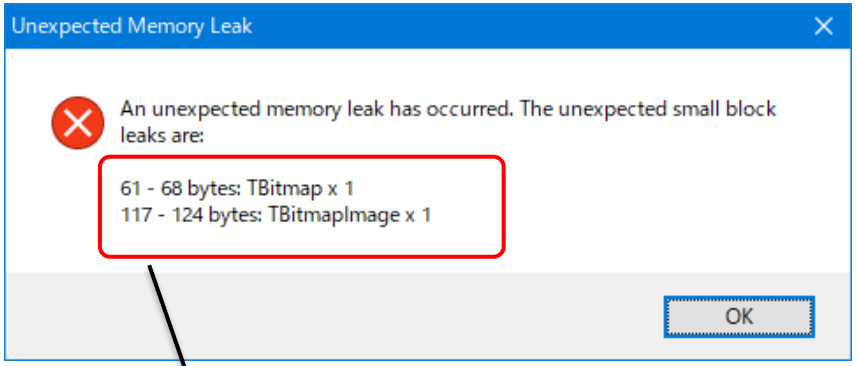
■ （３）メモリーリーク調査方法

- テスト

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then Exit;

  FBitMap := TBitmap.Create;
  try
    FBitMap.LoadFromFile(OpenPictureDialog1.FileName);

    Image1.Picture.Assign(FBitMap);

  finally
    // FBitMap.Free;
  end;
end;
```

破棄する部分をコメント化する

アプリを終了



TBitMap自身と、内部で保持する画像イメージであるTBitmapImageがリークしたことが表示。

■ （４）言語レベルのデバッグ機能

- 言語レベルのデバッグ
  - 通常のデバッグは、実行時にステップを確認したり変数のチェックを行う。

  - プログラムソースレベルでも、デバッグに役立つ機能が用意されている。

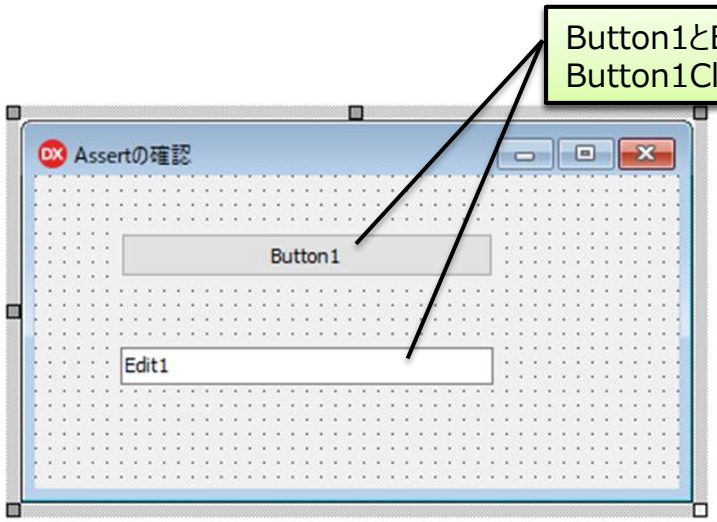
- 今回ご紹介する機能
  - Assert
    - 任意の条件で、例外を発生させる仕組み。
  - XMLドキュメントコメント
    - プログラム上で開発者に使用方法を伝える仕組み。

言語レベルのデバッグ機能は、ソースの品質向上に役立つ！

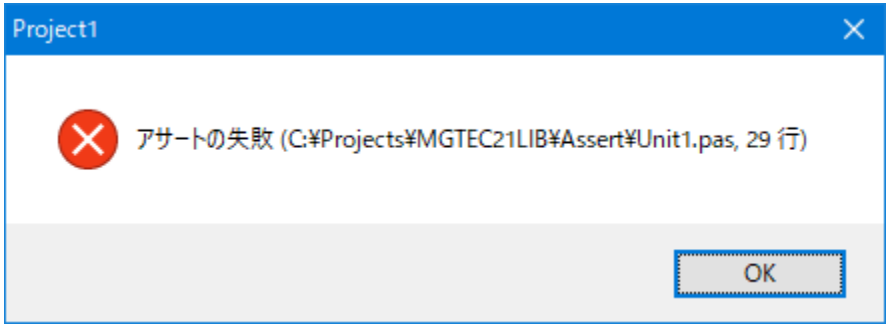
■ （４）言語レベルのデバッグ機能

- Assert
  - 任意のコードで例外を発生させる仕組み。
  - Assert(条件式)で、結果がFalseの場合に例外が生成。

```
procedure TForm1.ButtonClick(Sender: TObject);
begin
  Assert(Sender is TButton);

  TButton(Sender).Caption := Random(100).toString;
end;
```



実行して、Edit1をクリックすると、例外が発生



■ （４）言語レベルのデバッグ機能

- Assertの使いどころ

【プログラム仕様】
function Calc(AValue, AType: Integer): Integer;

AType = 1の場合、AValue × 3を戻り値とする。
AType = 2の場合、AValue + 3を戻り値とする。
AType = 3の場合、AValue ÷ 3の商を戻り値とする。
AType = 4の場合、AValue ÷ 3の余りを戻り値とする。

（AValue は 正の整数がはいることを想定する。）

実装例

```
function TForm1.Calc(AValue, AType: Integer): Integer;
begin
  case AType of
    1: Result := AValue * 3;
    2: Result := AValue + 3;
    3: Result := AValue div 3;
    4: Result := AValue mod 3;
  end;
end;
```

仕様を知らない開発者にパラメータの条件を伝える方法はないか？

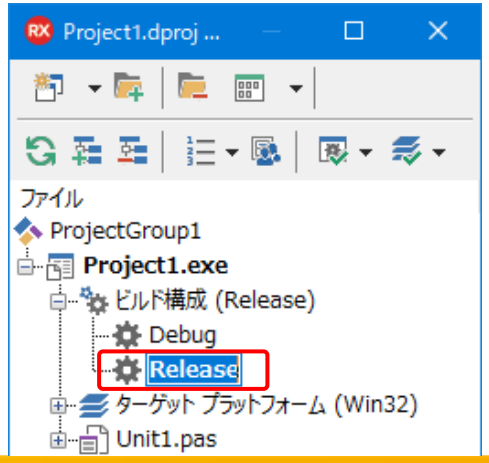
■ （４）言語レベルのデバッグ機能

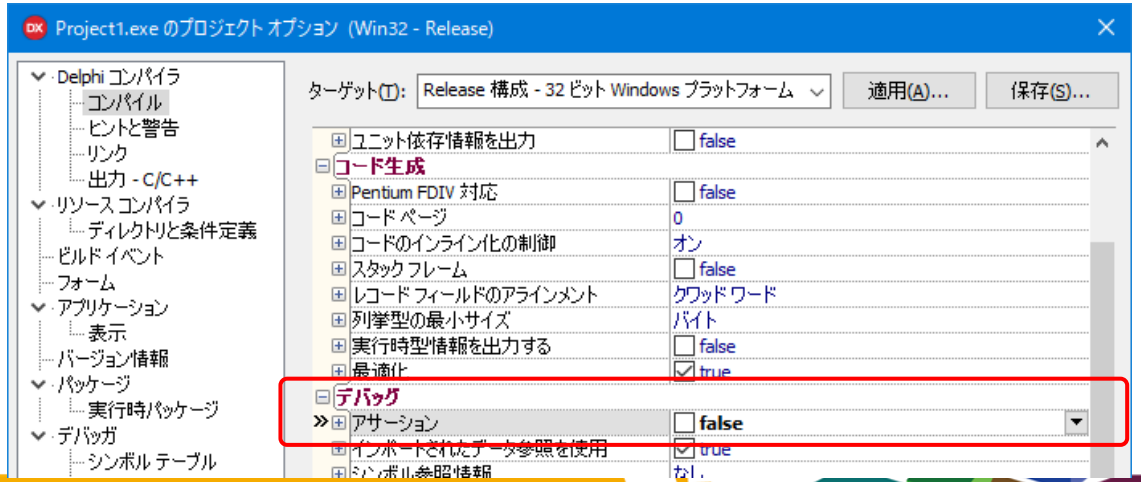
Assertを追加したソースコード

```
function TForm1.Calc(AValue, AType: Integer): Integer;
begin
  Assert(AValue > 0);
  Assert((AType >= 1) and (AType <= 4));

  case AType of
    1: Result := AValue * 3;
    2: Result := AValue + 3;
    3: Result := AValue div 3;
    4: Result := AValue mod 3;
  end;
end;
```

この部分をみれば、この関数のパラメータが要求する条件がわかる。

- 製品版でのビルド時
  - アサーション　オプションをOFFにすると、Assertはコンパイル対象外となる。オブジェクトの最適化が可能。





■ （４）言語レベルのデバッグ機能

- XMLドキュメントコメント
  - 独自に作成した関数や手続きの使用方法をコメントとして作成する機能。

宣言部

```
type
  TCalType = (ctAdd, ctSub, ctMult, ctDiv);

  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    function CalcMethod(AType: TCalType; AParam1, AParam2: Integer): Integer;
  end;
```

この関数を使用する開発者に、使用方法を理解してもらうために作成するのが、XMLドキュメントコメント。

実装部

```
function TForm1.CalcMethod(AType: TCalType; AParam1, AParam2: Integer): Integer;
begin
  case AType of
    ctAdd: Result := AParam1 + AParam2;
    ctSub: Result := AParam1 - AParam2;
    ctMult: Result := AParam1 * AParam2;
    ctDiv: Result := AParam1 div AParam2;
  end;
end;
```

■ （４）言語レベルのデバッグ機能

- 記述例
  - /// から始まるコメント行に、XMLタグ型式で説明文を作成する。

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
///   <summary>2つのパラメータの四則計算を行う</summary>
///   <remarks>ATypeに指定した演算子を使用して、2つの数値の演算を行う
///   </remarks>
///   <param name="Atype">四則演算区分</param>
///   <param name="AParam1">数値１</param>
///   <param name="AParam2">数値２</param>
///   <returns>演算結果</returns>
    function CalcMethod(AType: TCalType; AParam1, AParam2: Integer): Integer;
  end;
```

タグ	内容
<summary>	対象となる関数や手続き、クラスの要約を記述
<remarks>	備考を記述
<param name="パラメータ名">	パラメータの名前と説明を記述
<returns>	対象となる関数の戻り値を記述

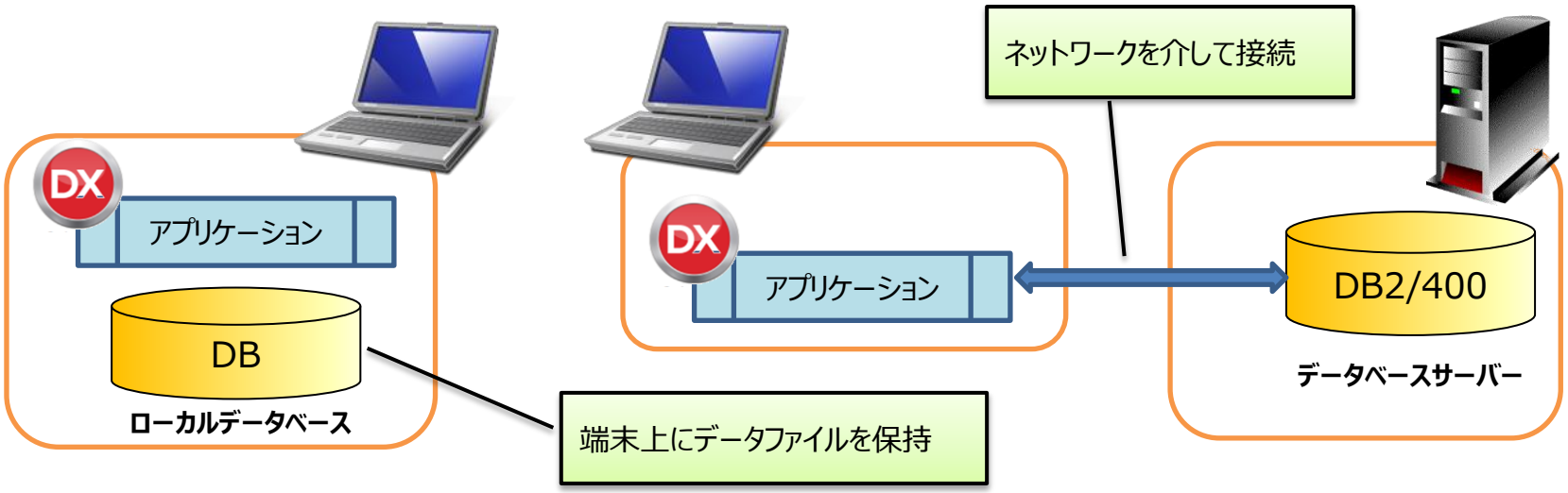
■ （４）言語レベルのデバッグ機能

- ドキュメントコメントの表示

２．ローカルデータベースの活用

■ ローカルデータベースとは？

- ローカルデータベースとデータベースサーバー

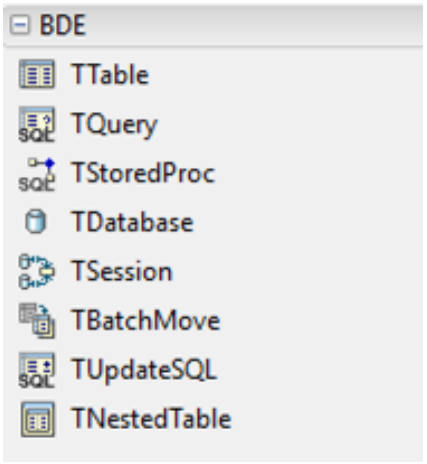


- ローカルデータベースを使用する局面
  - パフォーマンス向上に使用できる。
    - 変更少ないマスターをローカルに取込めば、サーバへのアクセスが減らせるため、レスポンスが向上。
  - 端末単位の設定情報などの保持に使用できる。
    - INIファイルやレジストリの代わりにデータベースに設定情報を保持。
  - ネットワークが使用できない場合でも、データが使用できる。
    - モバイル等でオフラインの際でも、アプリケーションが使用可能。

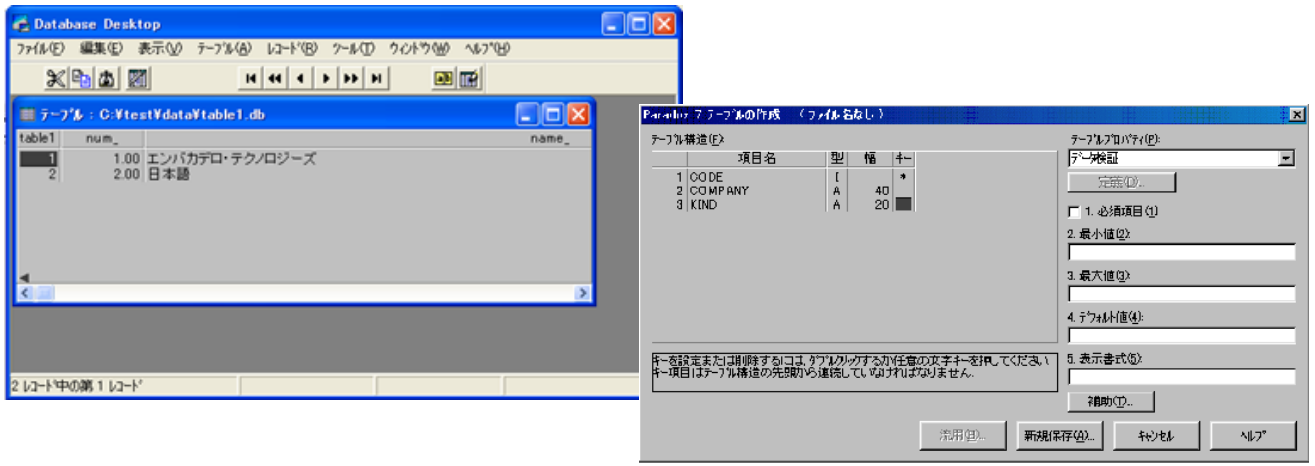
BDEでよく利用されたローカルデータベース

- Paradox
  - BDEで使用可能なローカルデータベース。クライアント端末にBDEを配布すれば、別途ドライバー不要でParadoxを使用できる。
  - 簡単にローカルデータベースが扱えるが、BDE自体のバージョンアップが終了しており、WindowsVista以降のOSで使用されるUACに対応しておらず、また、64ビット化にも対応できない。

BDEコンポーネント



Paradox Database Desktop



Paradoxに変わるローカルデータベースはないか？

■ 現在よく利用されるローカルデータベース

- SQLite：軽量コンパクトなデータベースシステム
  - ライブラリ型のデータベース管理システムの一つ。
  - 誰でも自由に入手、利用ができるパブリックドメインソフトウェア

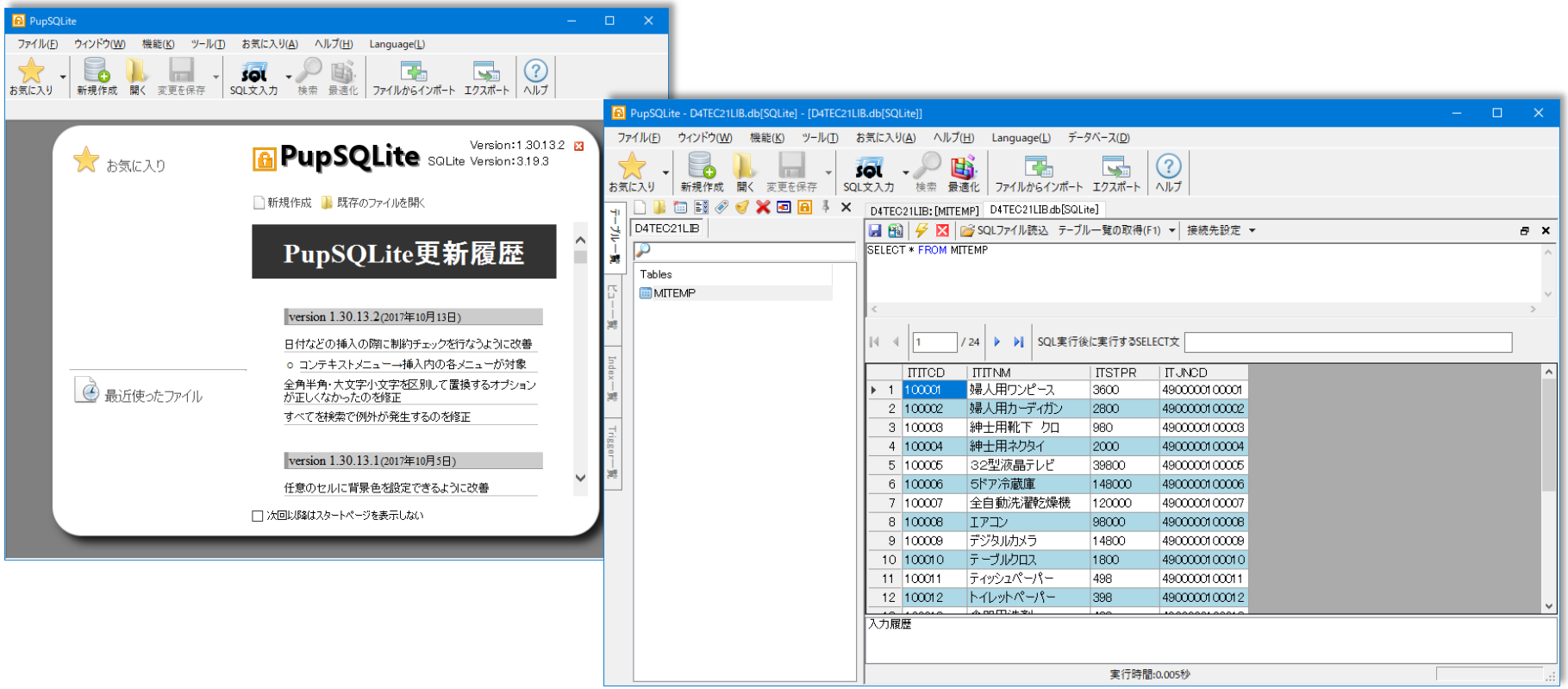
  - 標準的なSQLが使用可能。
  - データベースを一つのファイルで管理可能。（拡張子 *.db）
  - Windows、Macといったデスクトップ系OSだけでなく、Android、iOSといったモバイル系OS上でも使用可能。

Paradoxにあった、データベースを作成・編集・管理する「DataBase Desktop」のような機能がSQLiteには無いのか？

■ 現在よく利用されるローカルデータベース

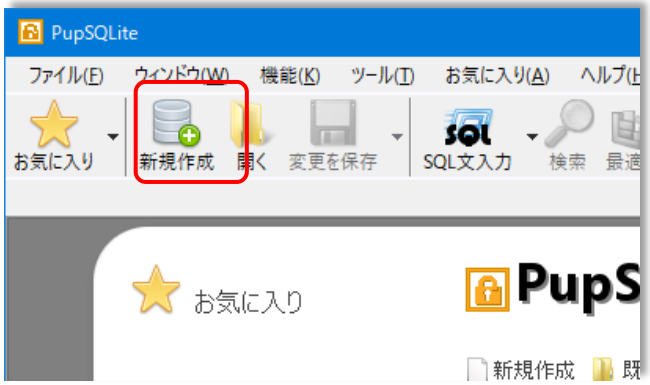
- PupSQLite
  - 日本語で、使用可能なSQLiteのDB管理ツールのひとつ。（フリーウェア）

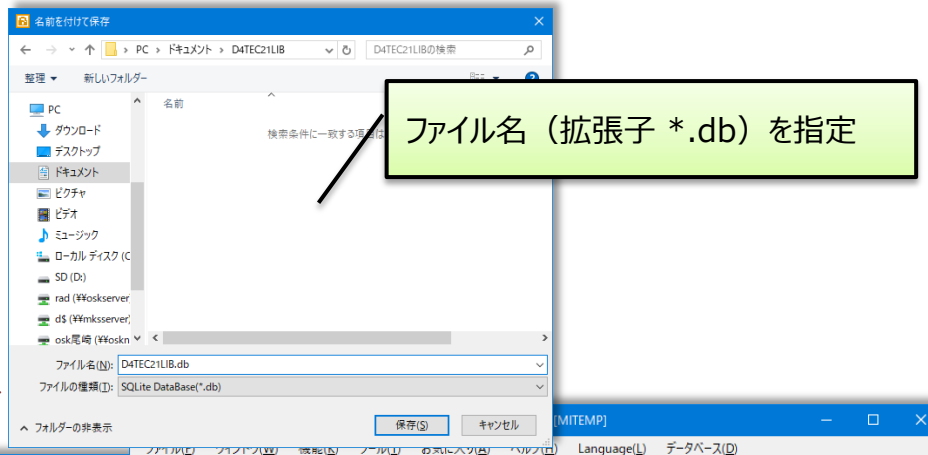
    http://www.vector.co.jp/soft/winnt/business/se449564.html
  - データベースファイルの作成やテーブルの作成、データ保守がGUI画面で可能。

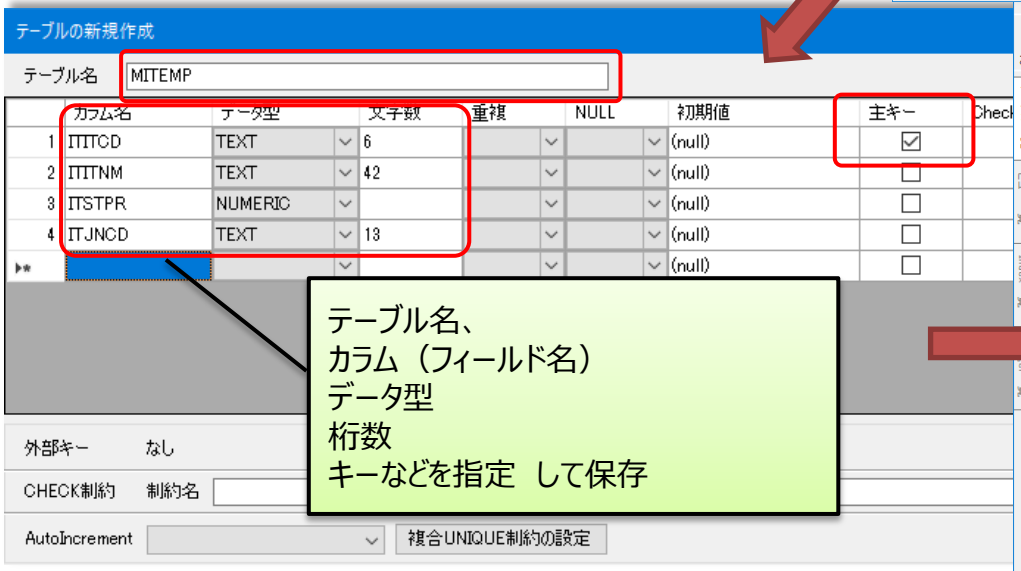


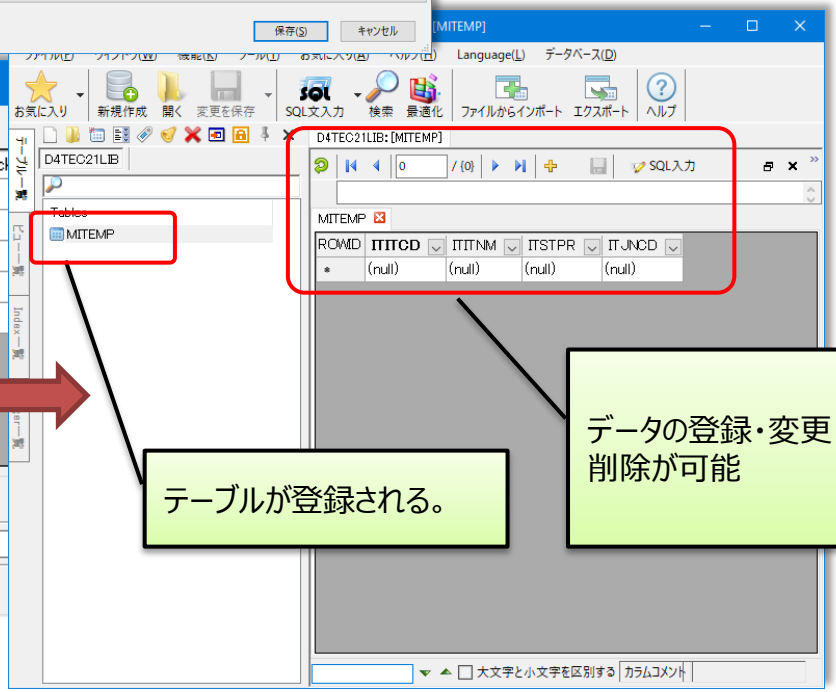
■ 現在よく利用されるローカルデータベース

- SQLiteデータベース作成手順



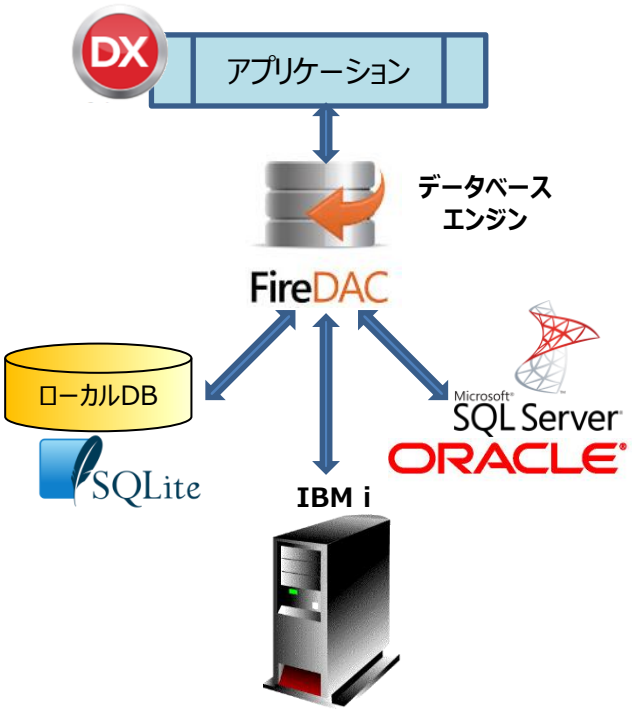






■ ローカルデータベースの活用

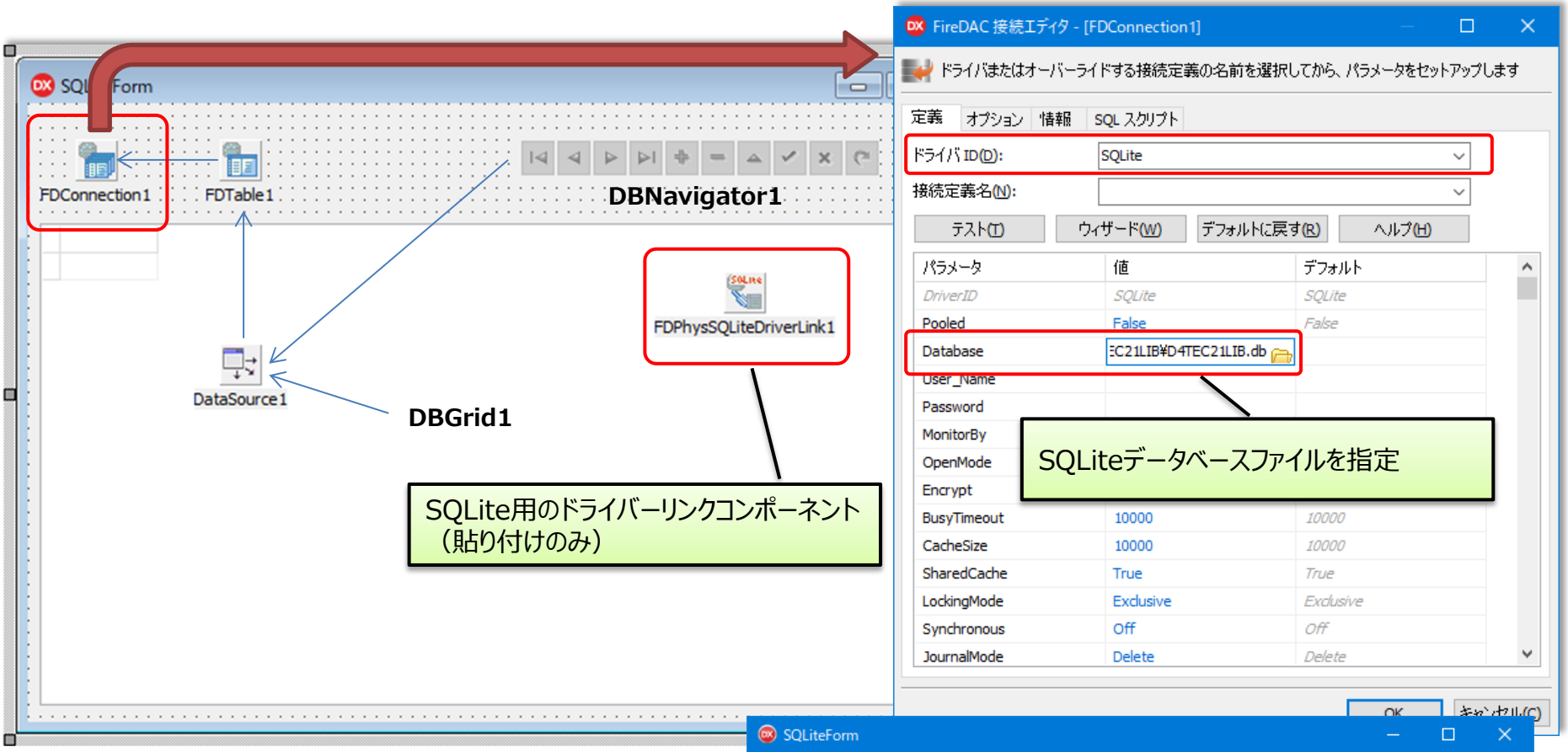
- シームレスに使用できるFireDAC
  - FireDACを使用することで、IBMiへの接続はもちろん、SQLServerやOracleといった各種データベースサーバーも接続できる。
  - SQLiteのようなローカルデータベースでも使用可能。特別なクライアントライブラリも不要なため、モバイルアプリでも使用可能。
    ローカルデータベースを使用することで、アプリ上のレスポンス向上やオフライン対応が実現可能。



- 今回ご紹介する機能
  - （１）FireDACを使用したローカルデータベース接続方法
  - （２）IBMi上のデータをローカルデータベースへ展開する方法
  - （３）モバイルアプリにおけるローカルデータベース活用方法

Delphi/400におけるSQLite使用方法のポイントをご紹介！

■ （１）FireDACを使用したローカルデータベース接続方法



IBMiを使用する時と全く同じ手順で開発可能。

ドライバIDをSQLiteにすればよい。

■ （2）IBMi上のデータをローカルデータベースへ展開する方法

DB2/400

報告書の表示

報告書の幅 . . . . . : 86

行の位置指定 . . . . . _____ 桁移動 . . . . . ____

行	商品コード	商品名称	標準単価
000001	100001	婦人用ワンピース	3,600
000002	100002	婦人用カーディガン	2,800
000003	100003	紳士用靴下　クロ	
000004	100004	紳士用ネクタイ	
000005	100005	３２型液晶テレビ	
000006	100006	５ドア冷蔵庫	
000007	100007	全自動洗濯乾燥機	
000008	100008	エアコン	
000009	100009	デジタルカメラ	
000010	100010	テーブルクロス	
000011	100011	ティッシュペーパー	

IBMiのファイルレイアウト（DDS）

```
0001.00     A*
0002.00     A*      システム　名        : DELPHI/400 テクニカルセミナー
0003.00     A*      ファイル名          : MITEMP
0004.00     A*      ファイル記述        : 商品マスタ
0005.00     A*      作成日              : 2017/10/21
0006.00     A*      作成者              : MIGARO.OZAKI
0007.00     A*
0008.00     A                                   UNIQUE
0009.00     A           R MITEMR                TEXT(' 商品マスタ ')
0010.00     A*
0011.00     A             ITITCD        6A      COLHDG(' 商品コード ')
0012.00     A             ITITNM       42O      COLHDG(' 商品名称 ')
0013.00     A             ITSTPR        9P 0    COLHDG(' 標準単価 ')
0014.00     A             ITJNCD       13A      COLHDG(' ＪＡＮコード ')
0015.00     A*
0016.00     A           K ITITCD
```

IBMi上でのデータをローカルデータベースに展開できれば、オフラインで使用できたり、サーバーへのアクセス回数を減らすことができ、レスポンス向上が期待できる。

DB

ローカルデータベースのファイルレイアウト

テーブルの新規作成

テーブル名　MITEMP

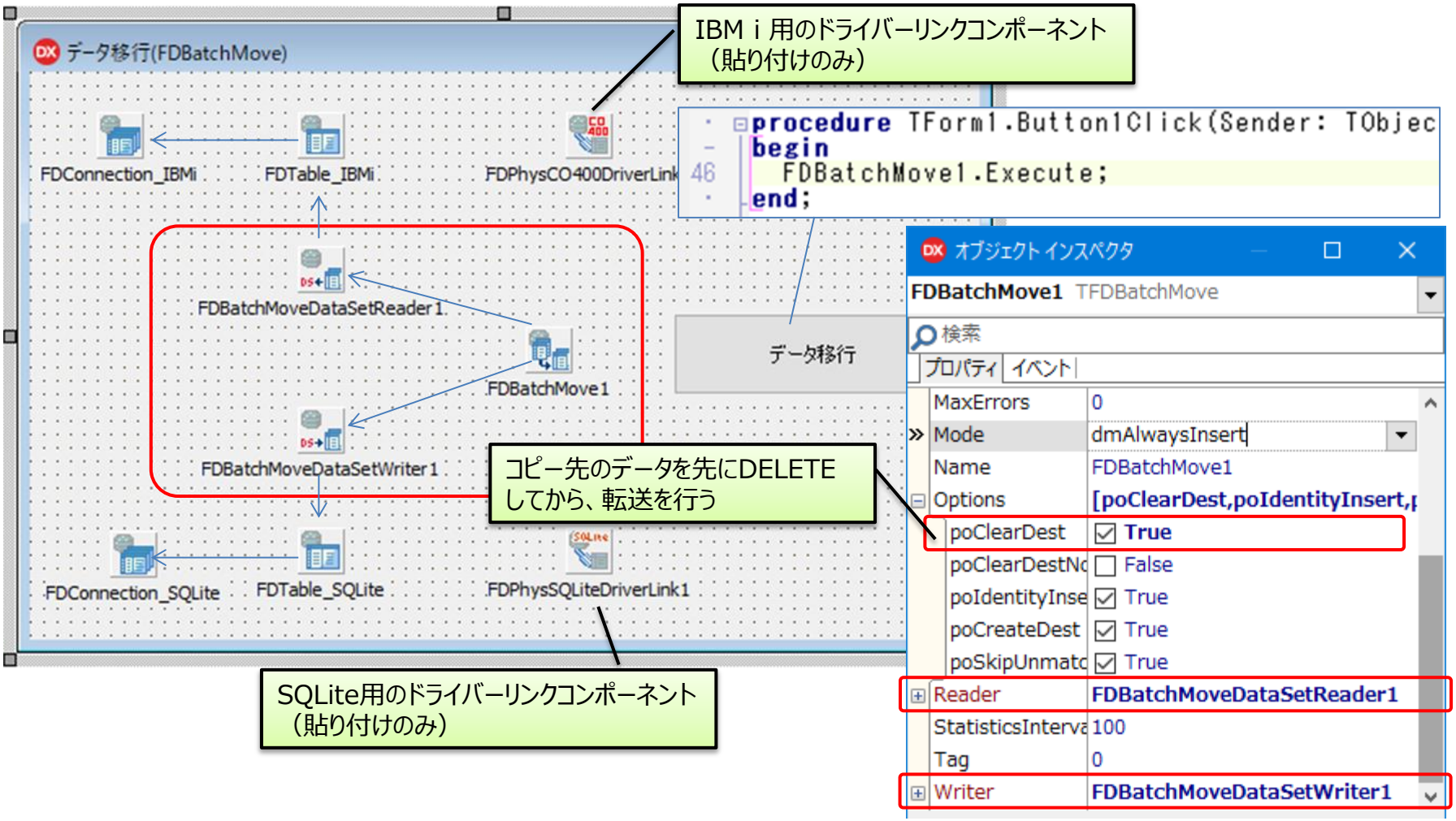
	カラム名	データ型	文字数	重複	NULL	初期値
1	ITITCD	TEXT	6			(null)
2	ITITNM	TEXT	42			(null)
3	ITSTPR	NUMERIC				(null)
4	ITJNCD	TEXT	13			(null)
▶*						(null)

フィールドID、属性、桁数などをIBMiのレイアウトを元に定義。

サーバーのデータを容易にローカルデータベースに展開できないか？

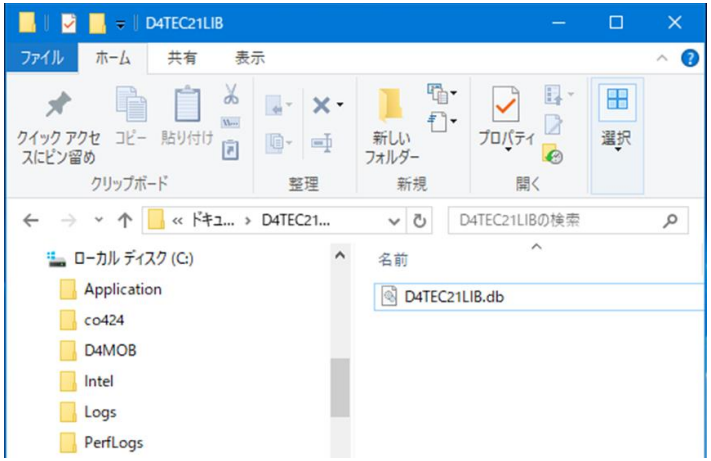
■ （２）IBMi上のデータをローカルデータベースへ展開する方法

- TFDBatchMove
  - 異なるデータソース間のデータコピーを実現

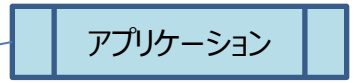


■ （３）モバイルアプリにおけるローカルデータベース活用方法

- データベースファイルの配置先
  - Windowsの場合

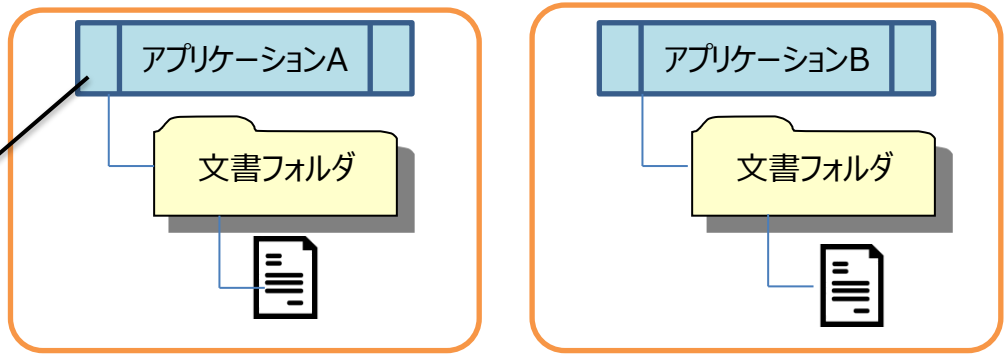


アプリケーションから、任意のフォルダにアクセスできるため、
データベースファイルの配置は自由にデザインできる。



  - iOS、Androidの場合

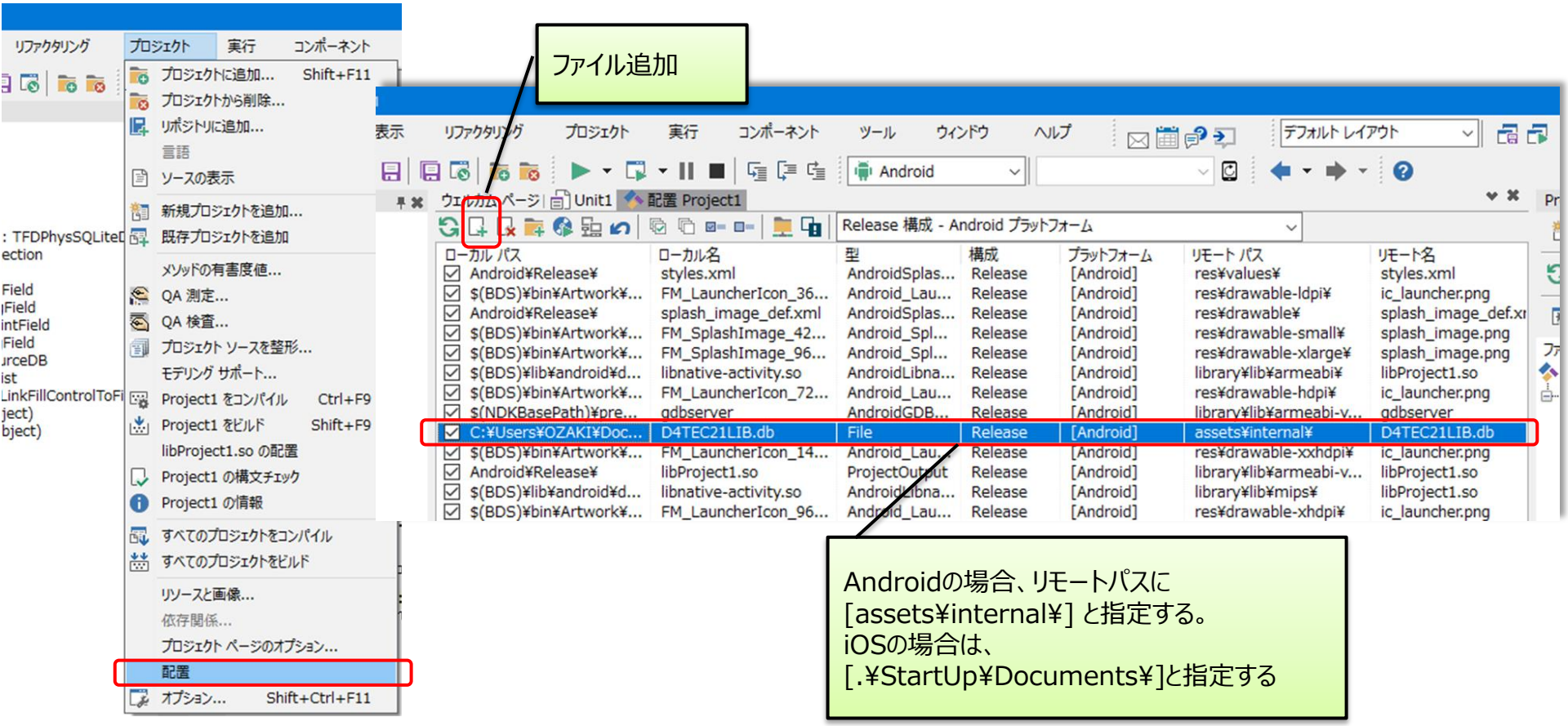
アプリケーション毎に専用の文書フォルダが用意
されており、原則決まった個所にしかアクセスできない。



モバイルの場合、どのようにデータベースファイルを配置するか？

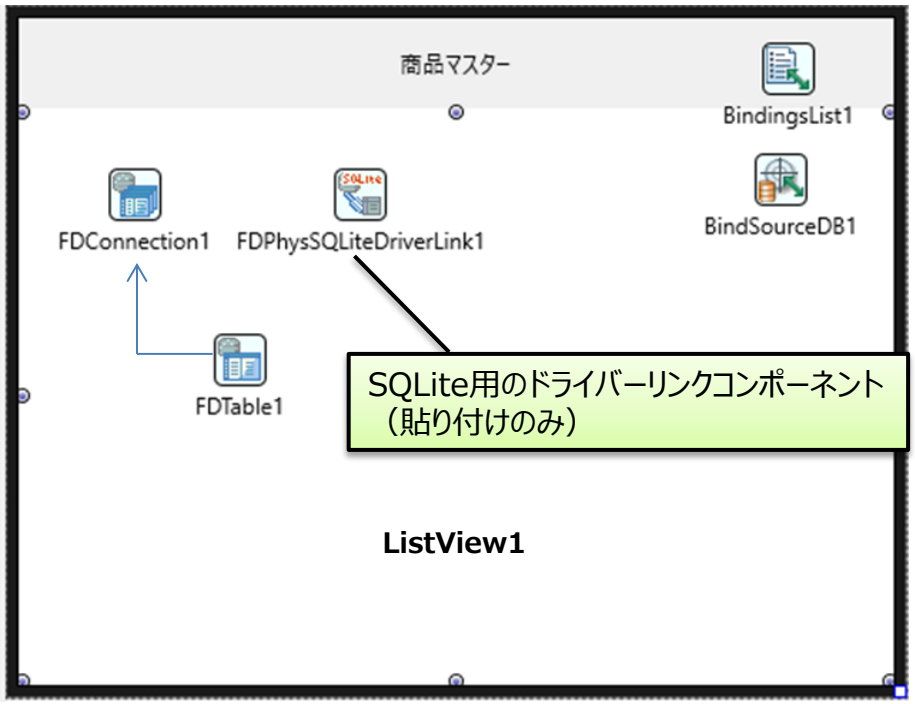
■ （３）モバイルアプリにおけるローカルデータベース活用方法

- データベースファイルの配置
  - 配置マネージャより、独自に追加したいファイルを指定すれば、ビルド時にアプリと一緒に文書フォルダに配置される。



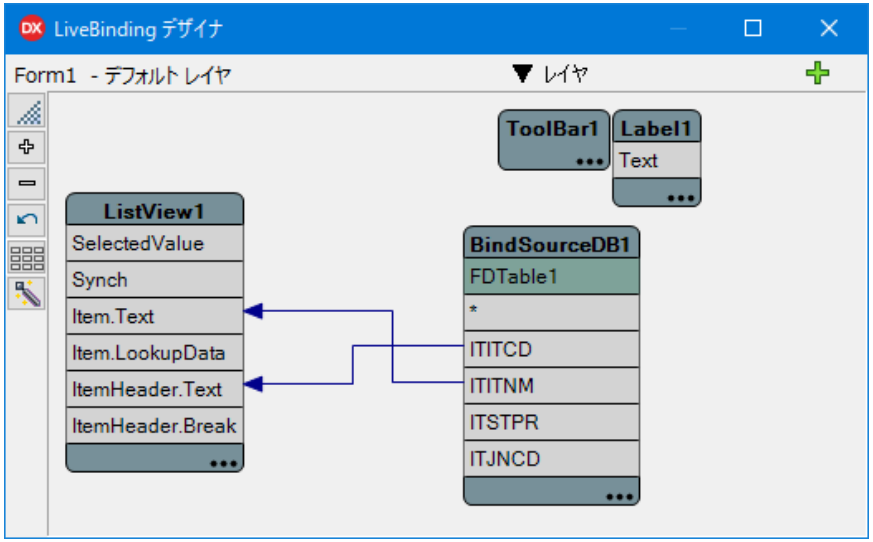
■ （３）モバイルアプリにおけるローカルデータベース活用方法

マルチデバイスアプリケーション（FireMonkey）



FireMonkeyでは、LiveBindingを使用することで、コンポーネントとデータセットを関連付けする方法を利用。

LiveBinding



ソースコード例

```
uses System.IOUtils;

50 procedure TForm1.FormCreate(Sender: TObject);
   begin
     FDConnection1.Params.Values['Database'] :=
         TPath.Combine(TPath.GetDocumentsPath, 'D4TEC21LIB.db');

     FDConnection1.Connected := True;
     FDTable1.Active := True;
   end;
```

アプリケーションの文書フォルダを取得。
(System.IOUtilsに定義)

■ （３）モバイルアプリにおけるローカルデータベース活用方法

- 実行

まとめ

■ まとめ

- Delphi/400デバッグテクニック
  - Delphi/400におけるデバッグの基本
  - ブレークポイントの効果的な使用方法
  - 呼出履歴やローカル変数、イベントログの活用方法
  - マルチスレッドアプリのデバッグ方法
  - メモリーリークの対処法
  - ソースコード上での品質向上テクニック

- ローカルデータベースの活用
  - ローカルデータベース使用の局面
  - SQLiteの使用方法
  - IBMi上のデータをローカルへの展開する方法
  - モバイルからのSQLiteの使用方法

ご清聴ありがとうございました。