【セッションNo. 3】

プログラミングテクニックセッション

開発者が知りたい実践プログラミングテクニック！
～　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

■ 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！

- スマートデバイス対応のWebアプリを構築したい。
  - PC前提のWeb画面のままだと、スマートデバイスでの使い勝手がわるい。
  - タッチ操作で使いやすい画面を構築したい。

1. IntraWebにおけるスマートデバイス向けWebアプリ作成テクニック

- プログラムの品質を向上したい。
  - 仕様にもとづくテストを正しく実施したい。
  - プログラムを修正した際に、デグレートを発生させたくない。

2. ユニットテストフレームワークの活用方法

- Delphi/400の運用環境を効率よく導入したい。
  - 多数のPCにDelphi/400の運用環境を導入しないといけない。

3. Delphi/400サイレントインストーラ作成方法

1．IntraWebにおけるスマートデバイス向けWebアプリ作成テクニック

■ スマートデバイスに最適なWebアプリの検討

- PCブラウザを前提としたWebアプリをスマートフォンで実行

スマートデバイスで使いやすいWebアプリとはどんなものか？

■ スマートデバイスに最適なWebアプリの検討

- スマートデバイスの特徴
  - スマートフォンやタブレットなど、種類や端末により、サイズや解像度がマチマチ
  - 操作は、タッチやスワイプなど指を使用 （細かな入力などは不向き）

→ スマートデバイスに最適なWebアプリとは？

  - デバイスにかかわらず、レイアウトがきちんと表示されるのが理想

スマートフォン

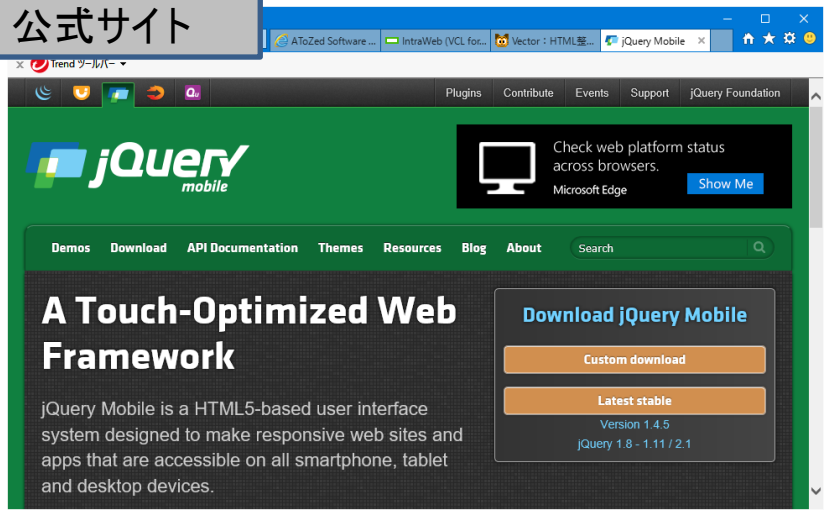
タブレット

スマートデバイスWebアプリ作成には、“jQuery Mobile”が使用されることが多い

■ スマートデバイスに最適なWebアプリの検討

- jQuery Mobile フレームワークとは？
  - スマートデバイス用の画面デザインや部品を提供するJavaScriptライブラリ
    - デバイスやブラウザの違いをフレームワークが吸収
    - スマホやタブレットでのタッチ操作に最適化

公式サイト



http://jquerymobile.com/

（2016/06現在　最新版は、Ver.1.4.5）

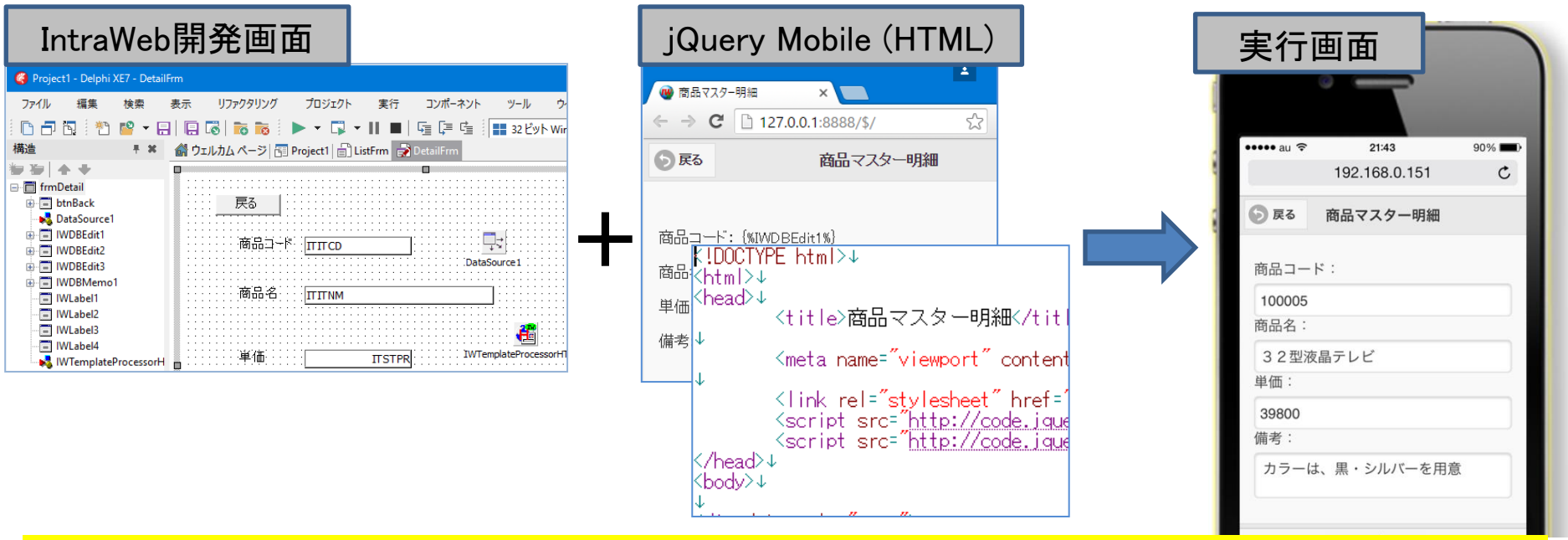
使用されているサイト例

ASCII.jp「jQuery Mobileを使った国内スマホサイトまとめ」より

IntraWebアプリでjQuery Mobileを活用できないか？

■ スマートデバイスに最適なWebアプリの検討

- IntraWebとjQuery Mobileとの連携
  - jQuery Mobileに用意されている画面デザインや部品は、HTMLで記述可能
  - IntraWebは、通常IWフォームにコンポーネントを配置して画面作成するが、画面デザインをHTMLで作成してIWフォームと連携する方法も用意されている。



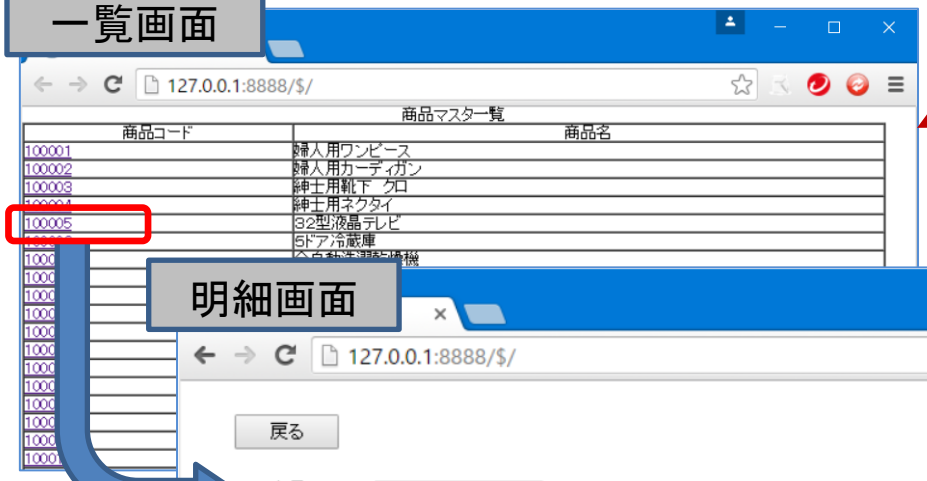
jQuery Mobileの組み合わせでスマートデバイスに最適な画面を実現可能！

■ サンプルプログラム

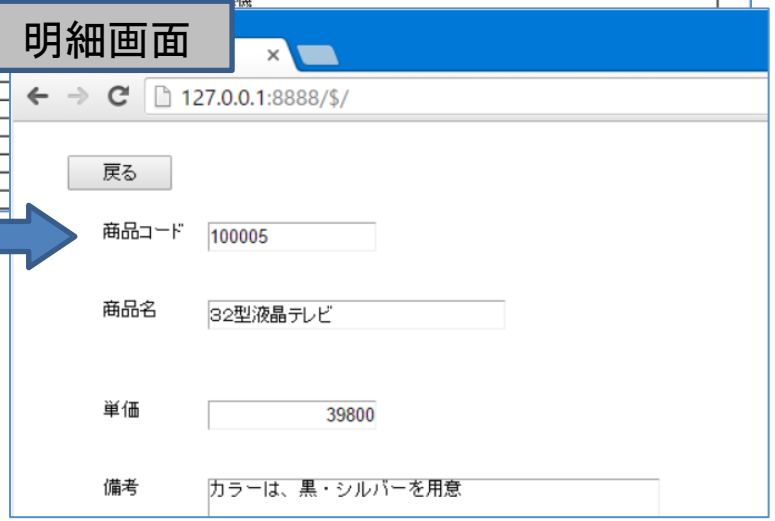
- IntraWebアプリをjQuery Mobileを使用してスマートデバイスに最適化
  - 商品マスター照会プログラム

変更前（ PCブラウザで実行）

一覧画面

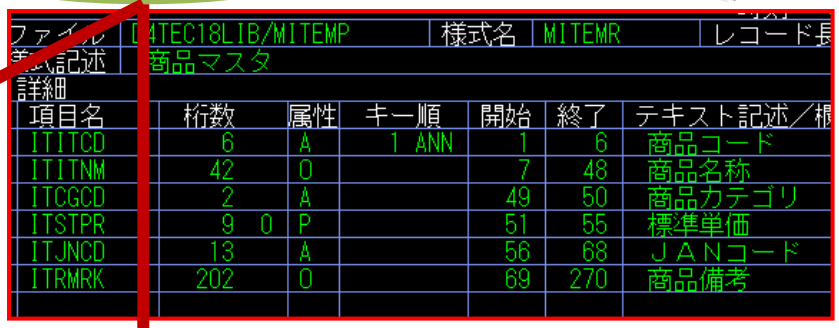


明細画面



MITEMP
（商品マスター）

IBM i



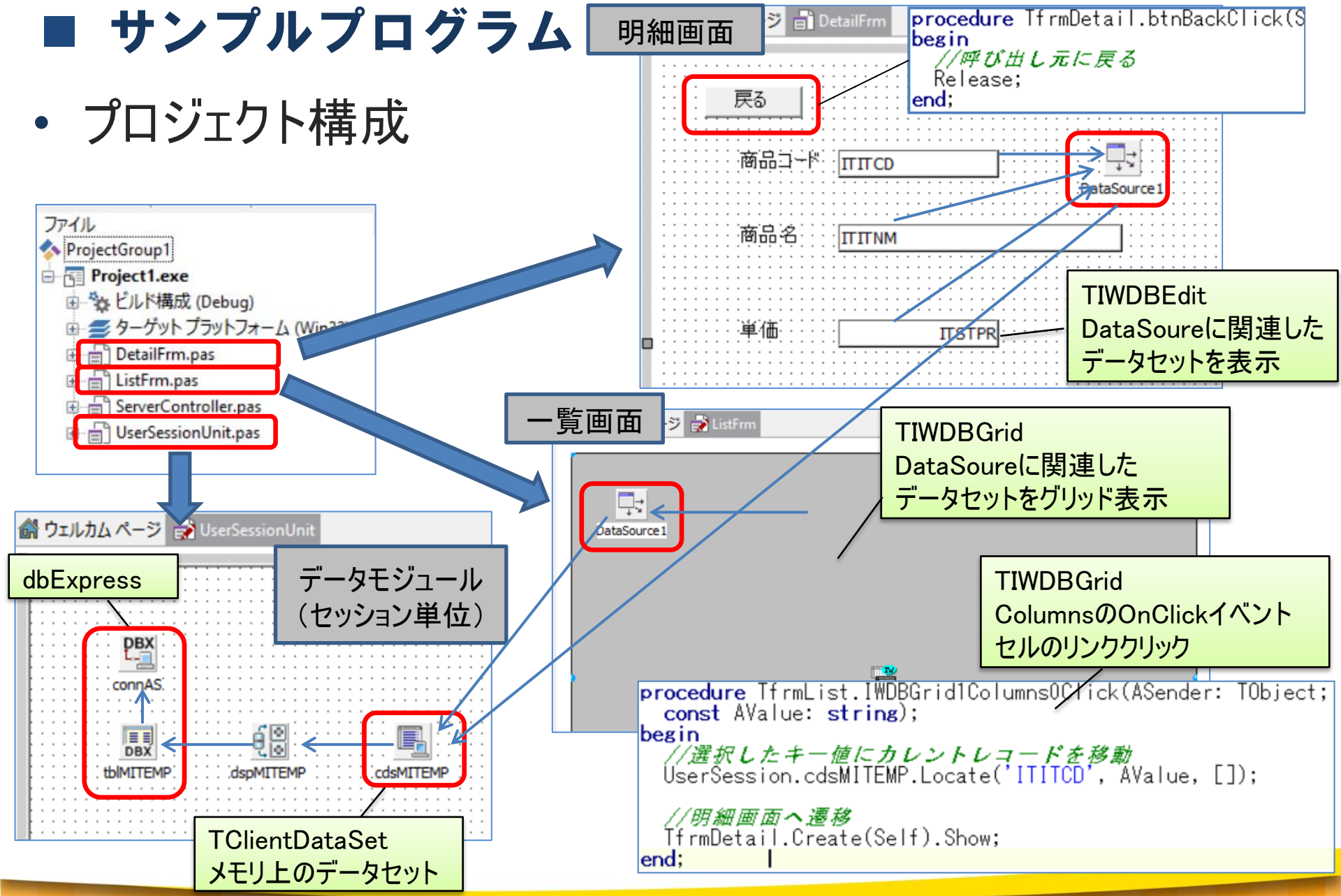
完成イメージ（スマートフォンブラウザで実行）

一覧画面

明細画面

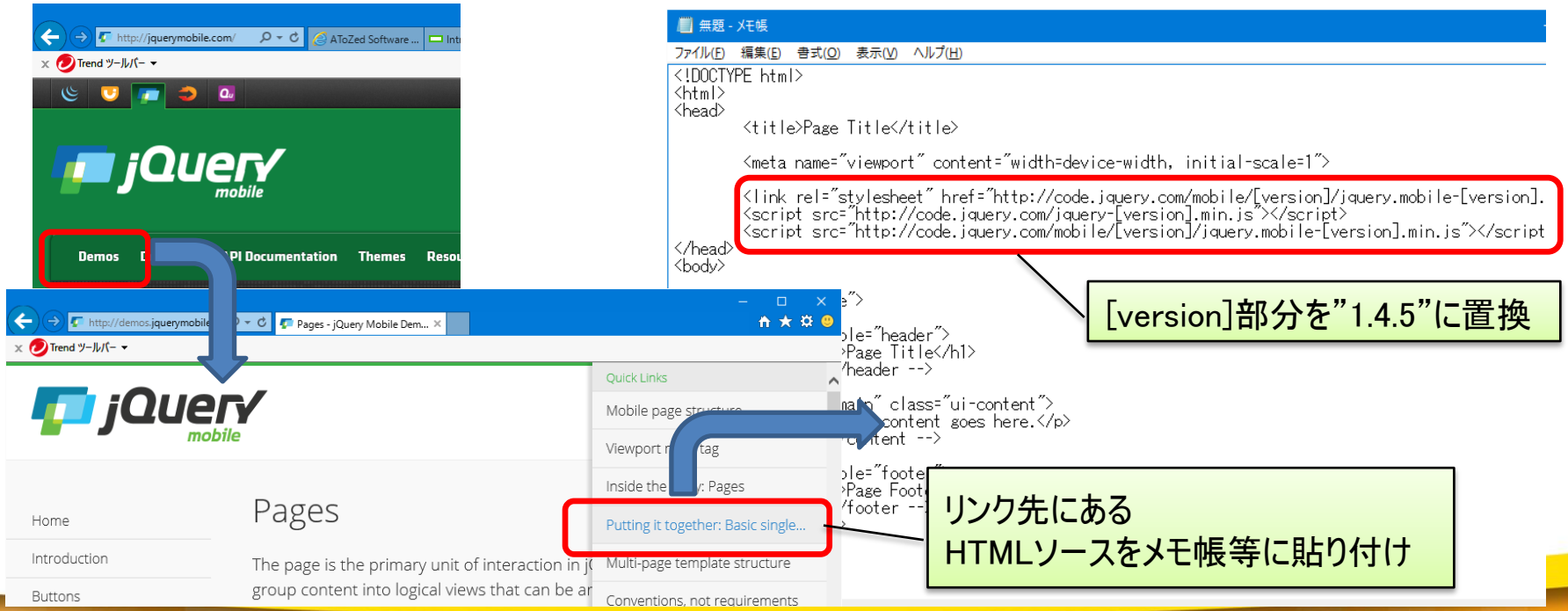
■ サンプルプログラム

- プロジェクト構成



jQuery Mobile 基本ページレイアウト

- jQuery Mobile公式サイトより基本ページのHTMLソースを取得
  - トップページ から [Demos]をクリックし、[jQuery Mobile 1.4.5 Demos]をクリック
  - 「Pages & Navigation」 内の [Pages] をクリック
  - 「Putting it together: Basic single page template」内にあるHTMLソースを コピーして、メモ帳等のテキストエディタを起動して貼り付け
  - HTMLソース内の[version]部分を最新版である”1.4.5”に全て置換



■ jQuery Mobile 基本ページレイアウト

- 基本ページレイアウトのHTML構成

Basic single page template

```
<!DOCTYPE html>
<html>
<head>
	<title>Page Title</title>

	<meta name="viewport" content="width=device-width, initial-scale=1">

	<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
	<script src="http://code.jquery.com/jquery-1.4.5.min.js"></script>
	<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
</head>
<body>

<div data-role="page">

	<div data-role="header">
		<h1>Page Title</h1>
	</div><!-- /header -->

	<div role="main" class="ui-content">
		<p>Page content goes here.</p>
	</div><!-- /content -->

	<div data-role="footer">
		<h4>Page Footer</h4>
	</div><!-- /footer -->
</div><!-- /page -->

</body>
</html>[EOF]
```

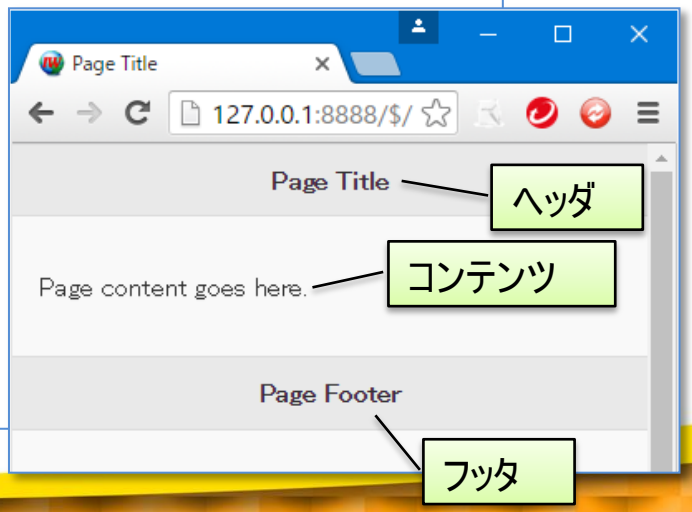
ページ幅、拡大率の設定

ページ全体

ヘッダ領域

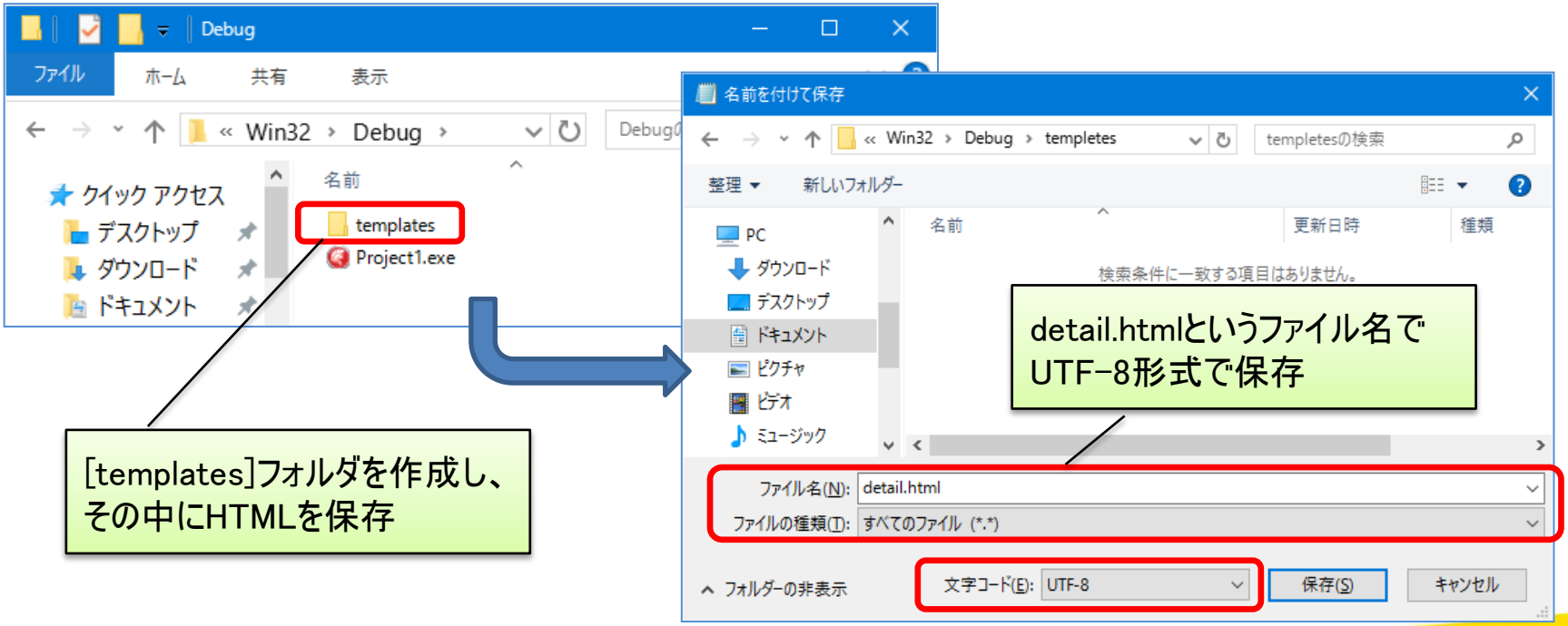
コンテンツ領域

フッタ領域



■ 明細画面の作成

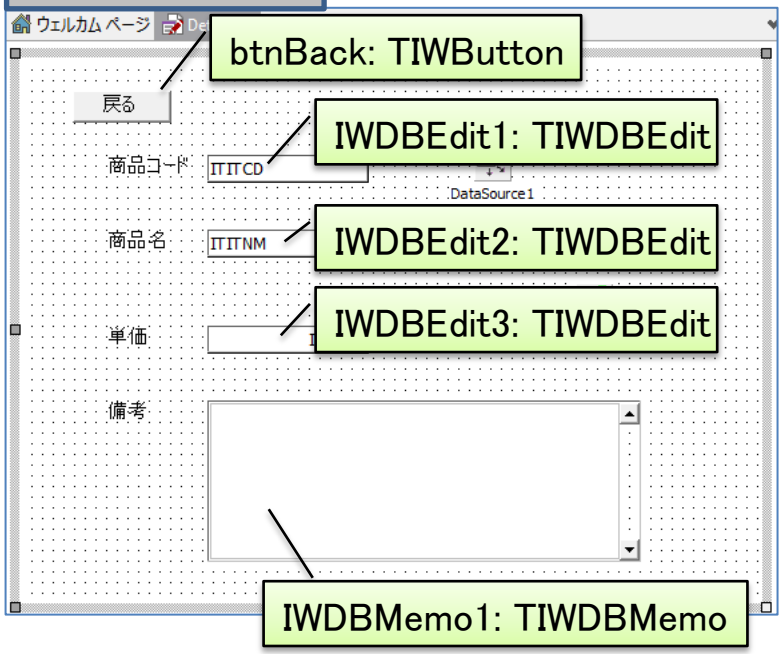
- テンプレート用フォルダにHTMLファイルを保存
  - メモ帳にて[名前を付けて保存]を選択
    - IntraWebアプリ実行フォルダの中に**[templates]フォルダ**を作成
    - [templates]フォルダの中にhtmlを保存
      （明細画面用として、ここではファイル名をdetail.htmlとして保存）
    - 保存の際、文字コードは**”UTF-8”**とする



■ 明細画面の作成

- HTMLとIntraWebコンポーネントとの関連付け
  - HTMLの中に、**{%コンポーネント名%}** と記述
  - ID属性はコンポーネント名の大文字となる。

DetailFrm.pas



detail.html

```
<div data-role="page">↓
    <div data-role="header">↓
        <button id="BTNBACK" class="ui-btn ui-btn-left↓
         ui-icon-back ui-btn-icon-left">戻る</button>↓
        <h1>商品マスター明細</h1>↓
    </div><!-- /header -->↓
↓
    <div role="main" class="ui-content">↓
        <div class="ui-field-contain">↓
            <label for="IWDBEDIT1">商品コード：</label>↓
            {%IWDBEdit1%}↓
            <label for="IWDBEDIT2">商品名：</label>↓
            {%IWDBEdit2%}↓
            <label for="IWDBEDIT3">単価：</label>↓
            {%IWDBEdit3%}↓
            <label for="IWDBMEMO1">備考：</label>↓
            {%IWDBMemo1%}↓
        </div>↓
    </div><!-- /content -->↓
↓
    <div data-role="footer">↓
        <h4>Migaro co., ltd.</h4>↓
    </div><!-- /footer -->↓
</div><!-- /page -->↓
```

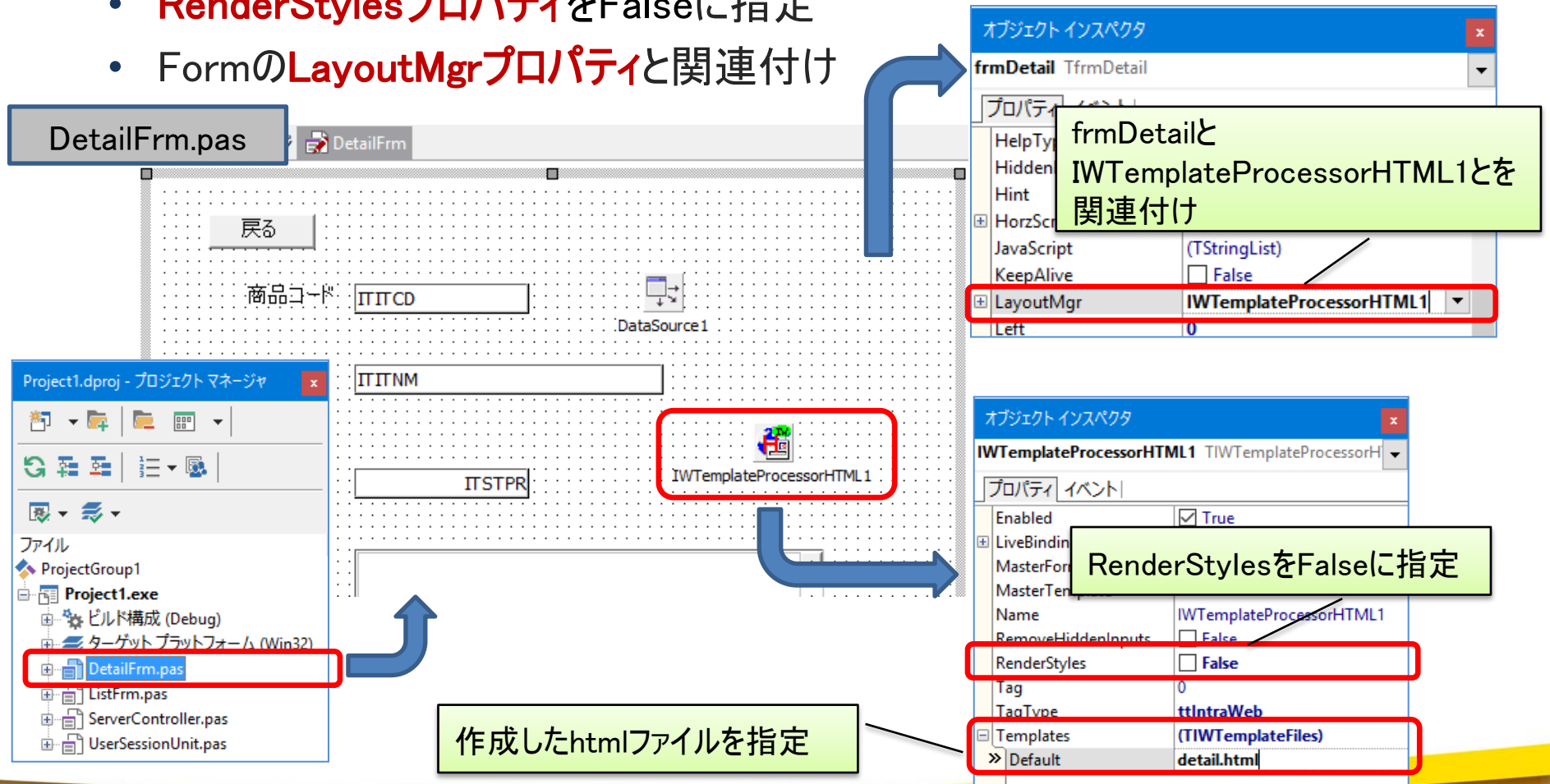
ui-btn
jQuery Mobileのボタン
ID属性により関連付け

ui-field-contain
ラベルとテキスト部品の
レイアウトを画面サイズに
より調整

IWコンポーネントの関連付け

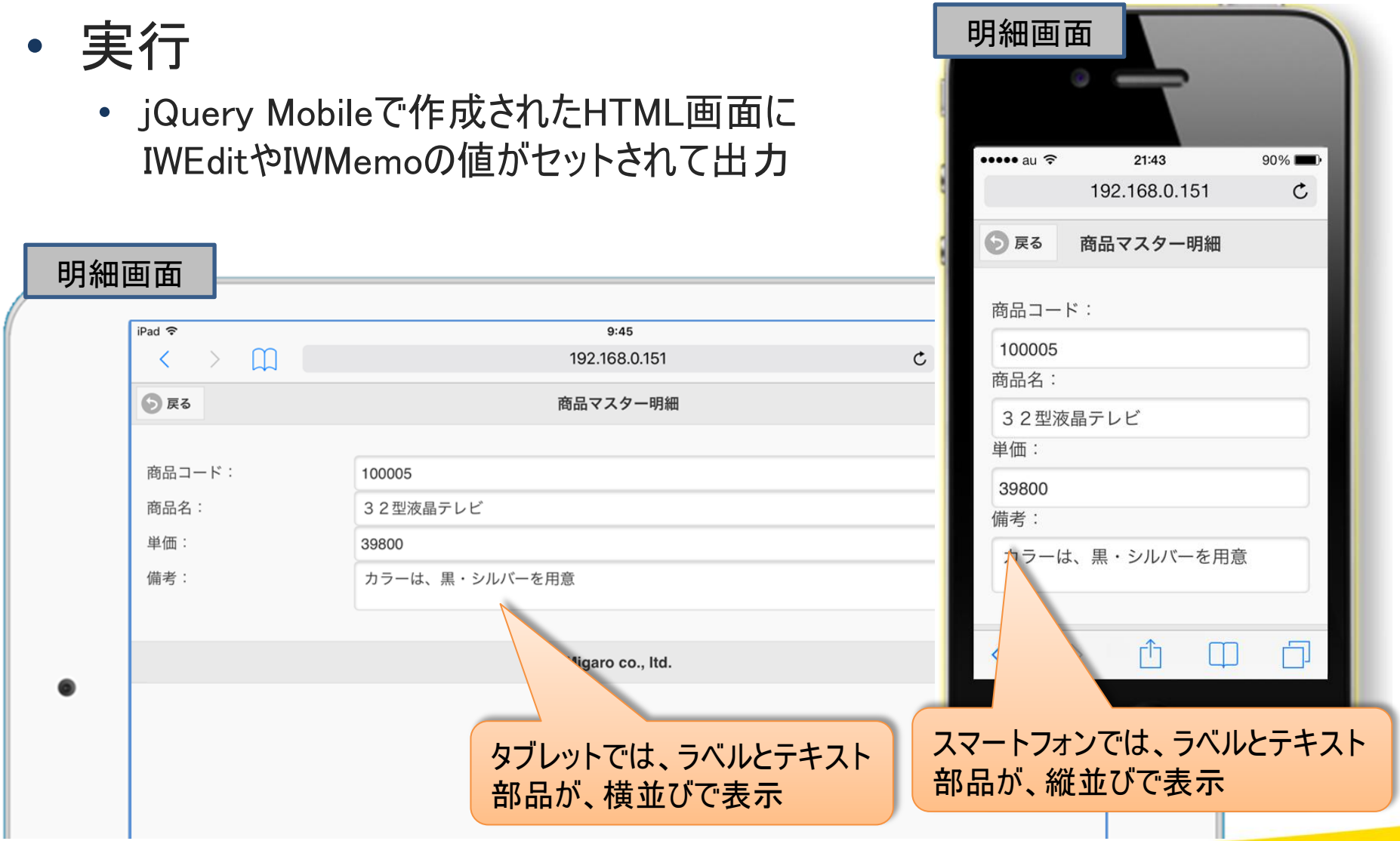
■ 明細画面の作成

- IWTemplateProcessorHTMLコンポーネントを使用
  - TemplatesプロパティのDefaultサブプロパティに　HTMLファイル名を指定
  - RenderStylesプロパティをFalseに指定
  - FormのLayoutMgrプロパティと関連付け



■ 明細画面の作成

- 実行
  - jQuery Mobileで作成されたHTML画面に IWEditやIWMemoの値がセットされて出力



■ 一覧画面の作成

- 一覧画面もテンプレートHTMLで動作するように変更
  - detail.htmlをコピーして、list.htmlを作成
  - ListFrm.pasにIWTemplateProcessorHTMLを追加してHTMLと紐づけを実施
  - list.htmlのコンテンツ領域に {%IWDBGrid1%}を指定
- 実行

一覧画面

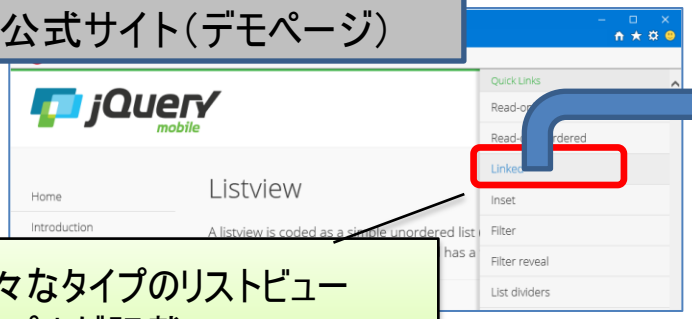
IWGridがそのまま表示されるため、
グリッドが画面からはみ出していて、
商品名が見えない

jQuery Mobileの一覧部品であるリストビューが使用できないか？

■ 一覧画面の作成

- ListView
  - リスト形式の一覧表示を行うjQuery MobileのUI部品

公式サイト(デモページ)



様々なタイプのリストビュー
サンプルが記載

list.html

```
<div data-role="header">↓
        <h1>商品マスター一覧</h1>↓
    </div><!-- /header -->↓
↓
    <div role="main" class="ui-content">↓
        <ul data-role="listview">↓
            <li><a href="#">Acura</a></li>↓
            <li><a href="#">Audi</a></li>↓
            <li><a href="#">BMW</a></li>↓
            <li><a href="#">Cadillac</a></li>↓
            <li><a href="#">Ferrari</a></li>↓
        </ul>↓
    </div><!-- /content -->↓
↓
    <div data-role="footer">↓
        <h4>Migaro co
    </div><!-- /footer --
</div><!-- /page -->↓
```

リンク先にある
htmlソースを
コンテンツ領域に貼り付け

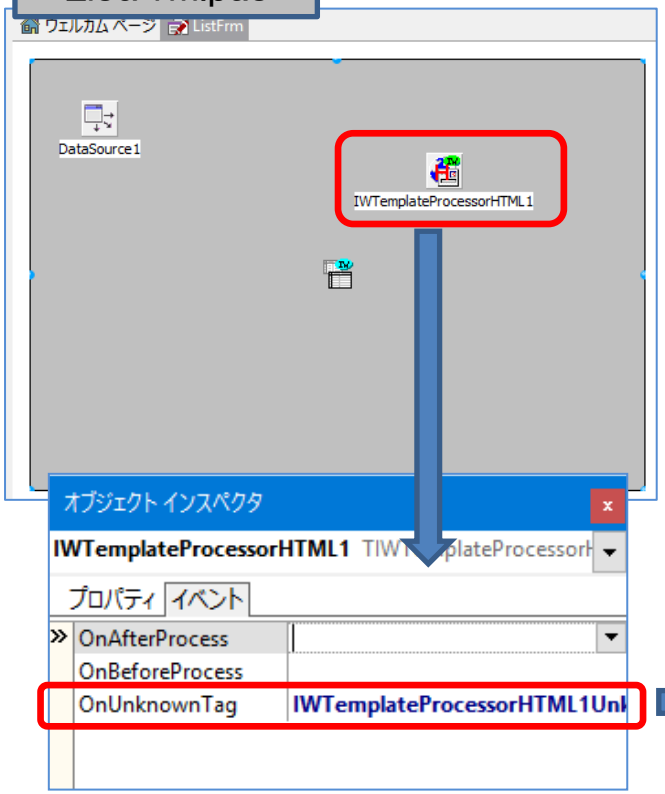
スマートフォンのタッチ操作に
最適なリスト形式で出力

プログラムから動的にタグが作成できないか？

■ 一覧画面の作成

- IWTemplateProcessorHTMLコンポーネントの OnUnknownTag イベントを使用
  - HTML上で **{% 名前 %}**で定義された部分に対して、ロジックでタグを作成可能

ListFrm.pas



list.html

```
data-role="page">↓
↓
        <div data-role="header">↓
                <h1>商品マスター一覧</h1>↓
        </div><!-- /header -->↓
↓
        <div role="main" class="ui-content">↓
                {%ListItem%}↓
        </div><!-- /content -->↓
↓
        <div data-role="footer">↓
                <h4>Migaro co., ltd.<
        </div><!-- /footer -->↓
</div><!-- /page -->↓
```

タグを埋め込みたい箇所に {% 名前 %} と定義

```
procedure TfrmList.IWTemplateProcessorHTML1UnknownTag(const AName: string;
  var VValue: string);
begin
  //ListItem時のタグ作成
  if AName = 'ListItem' then
  begin
    VValue := (* プログラムでタグ文字列を作成 *)
  end;
end;
```

ANameにセットされた名前より、タグ文字列をVValueにセット

■ 一覧画面の作成

- データにもとづく動的なListViewの作成

ListFrm.pas

```
procedure TfrmList.IWTemplateProcessorHTML1UnknownTag(const AName: string;
  var VValue: string);
var
  sl: TStringList;
begin
  if AName = 'ListItem' then
  begin
    sl := TStringList.Create;
    try
      sl.Add('<ul data-role="listview">');

      with UserSession.cdsMITEMP do
      begin
        First;
        while not Eof do
        begin
          sl.Add('<li><a href="#">' + FieldByName('ITITNM').AsString + '</a></li>');
          Next;
        end;
      end;
      sl.Add('</ul>');

      VValue := sl.Text;
    finally
      sl.Free;
    end;
  end;
end;
```

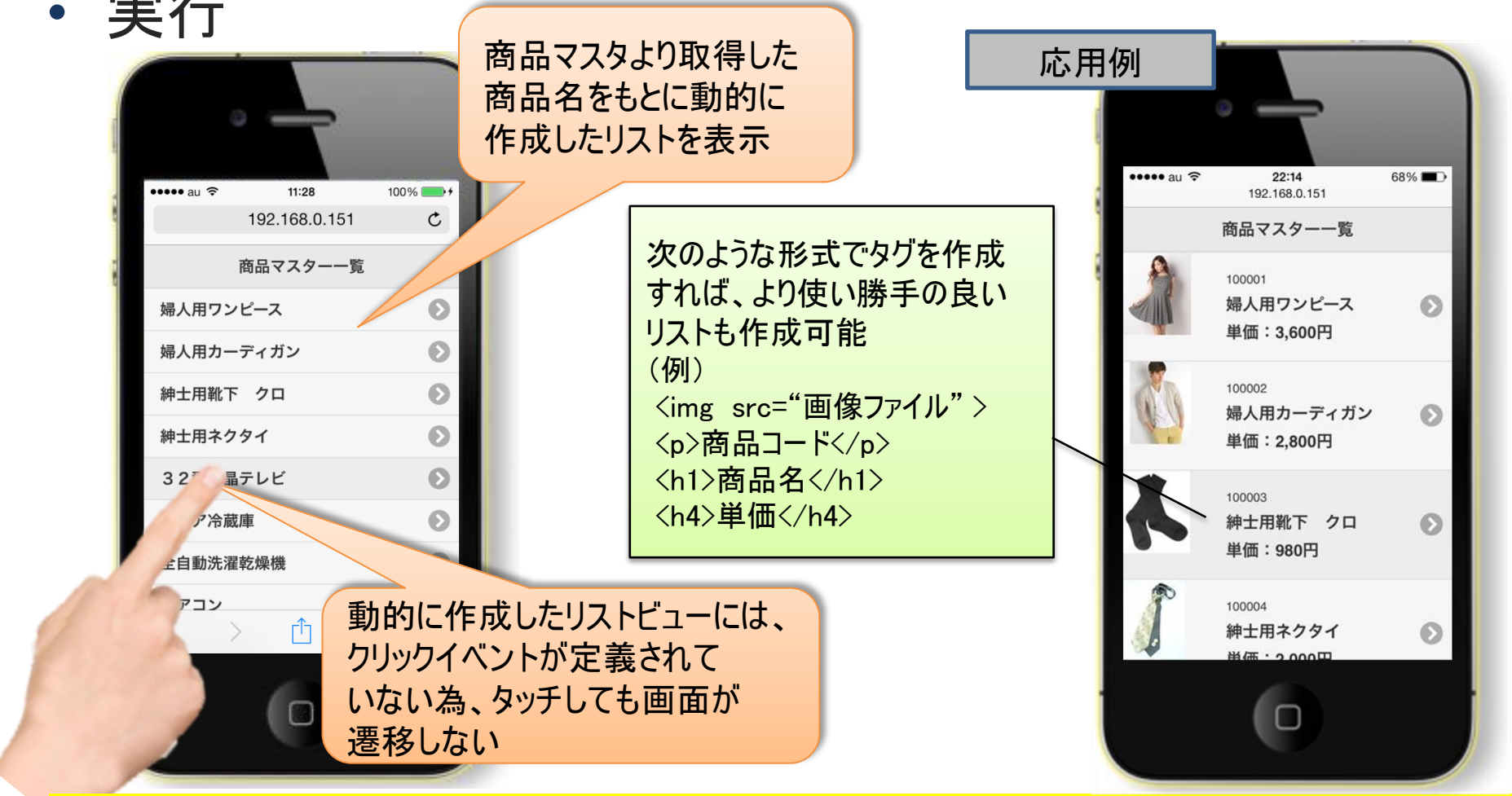
タグ文字列を作成するための文字列リスト変数

データセットをループでまわしながら、
リスト項目（<li>タグ）文字列を作成

完成した文字列リストの文字列をVValueにセット

■ 一覧画面の作成

- 実行

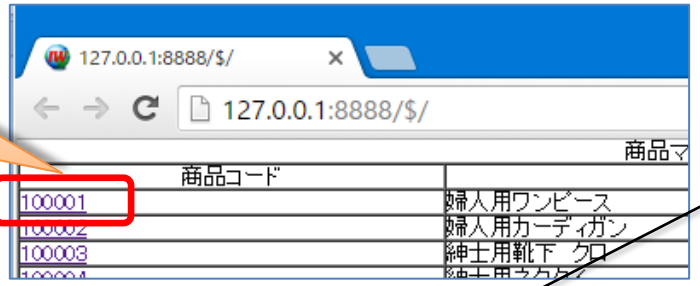


Delphiのイベントと関連付けることはできないか？

■ 一覧画面の作成

- IntraWebで生成されたHTMLとイベントとの関連
  - リンクのクリックがSubmitClickConfirmサブルーチンを呼び出し、Delphiのイベントが実行

IntraWebフォームで実行した一覧画面のソースを表示



コンポーネント名（大文字）

列番号 ＋ キー値

HTMLソース

```
<a href="#" onclick="return SubmitClickConfirm('IWDBGRID1','0_100001', true, '');">100001</a>↓
```

ListFrm.pas

```
procedure TfrmList.IWDBGrid1ColumnsOClick(ASender: TObject;
  const AValue: string);
begin
  //選択したキー値にカレントレコードを移動
  UserSession.cdsMITEMP.Locate('ITITCD', AValue, []);

  //明細画面へ遷移
  TfrmDetail.Create(Self).Show;
end;
```

動的に作成するListViewにonclick属性を追加すればよい！

■ 一覧画面の作成

- 動的なListViewの改良

ListFrm.pas

```
procedure TfrmList.IWTemplateProcessorHTML1UnknownTag(const AName: string;
  var VValue: string);
const
  ONCLICK = 'onclick="return SubmitClickConfirm(''IWDBGRID1'',''0_%s'', true, '''');"';
var
  sl: TStringList;
  sOnClickStr: String;
begin
  if AName = 'ListItem' then
  begin
    sl := TStringList.Create;
    try
      sl.Add('<ul data-role="listview">');

      with UserSession.cdsMITEMP do
      begin
        First;
        while not Eof do
        begin
          sOnClickStr := Format(ONCLICK, [FieldByName('ITITCD').AsString]);
          sl.Add('<li><a href="#" ' + sOnClickStr + '>'
            + FieldByName('ITITNM').AsString + '</a></li>');
          Next;
        end;
      end;
      sl.Add('</ul>');
      VValue := sl.Text;
    finally
      sl.Free;
    end;
  end;
end;
```

onclick属性 固定値
（可変部を%sで定義）

商品コードを可変部にセットし、
<a>要素にonclick属性の
文字列を追加

■ 一覧画面の作成

- 実行

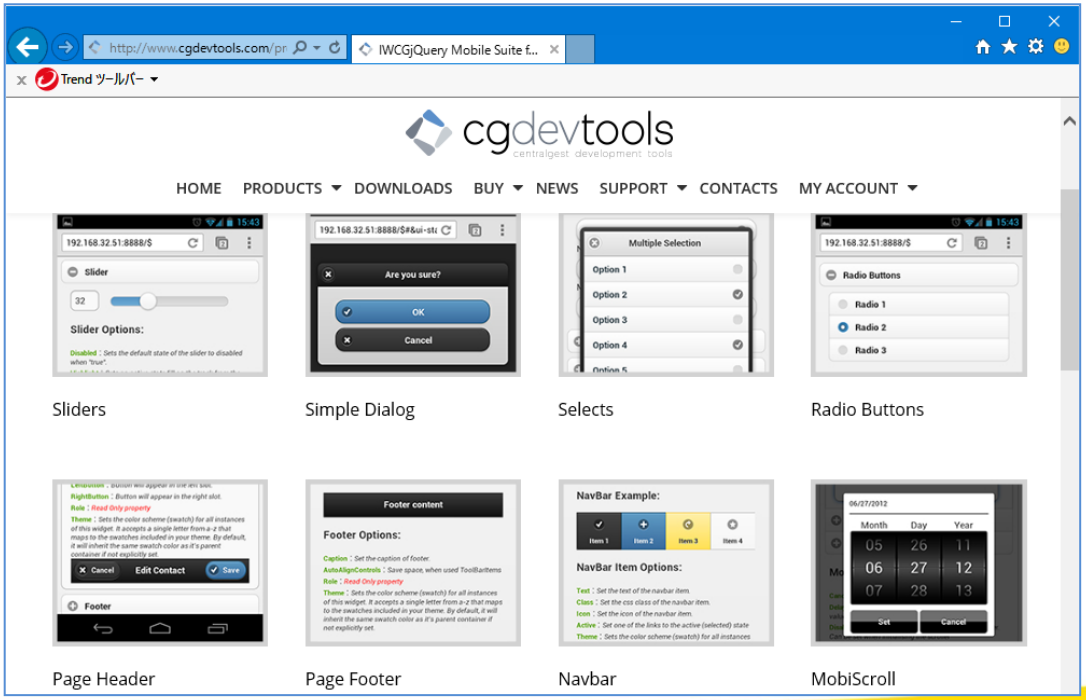
jQuery Mobileの活用で、スマートデバイスに適した画面が作成可能！

（参考）IWCGjQuery Mobileについて

- IntraWebを拡張する市販コンポーネント
  - jQuery Mobileを使用したスマートデバイスに最適なWebアプリをHTMLを使用せずに直接コンポーネントで作成可能

IWCGjQuery Mobile Suite

http://www.cgdevtools.com/products/iwcgjquery-mobile-suite/



2. ユニットテストフレームワークの活用方法

■ ユニットテストとは？

- ユニットテスト（単体テスト）とは、ソフトウェアやシステムのテスト手法の一つで、単一の部品（モジュール）を対象に行うテスト。

→ Delphiの場合、procedureやfunction 単位にテスト実施

メソッドの宣言部

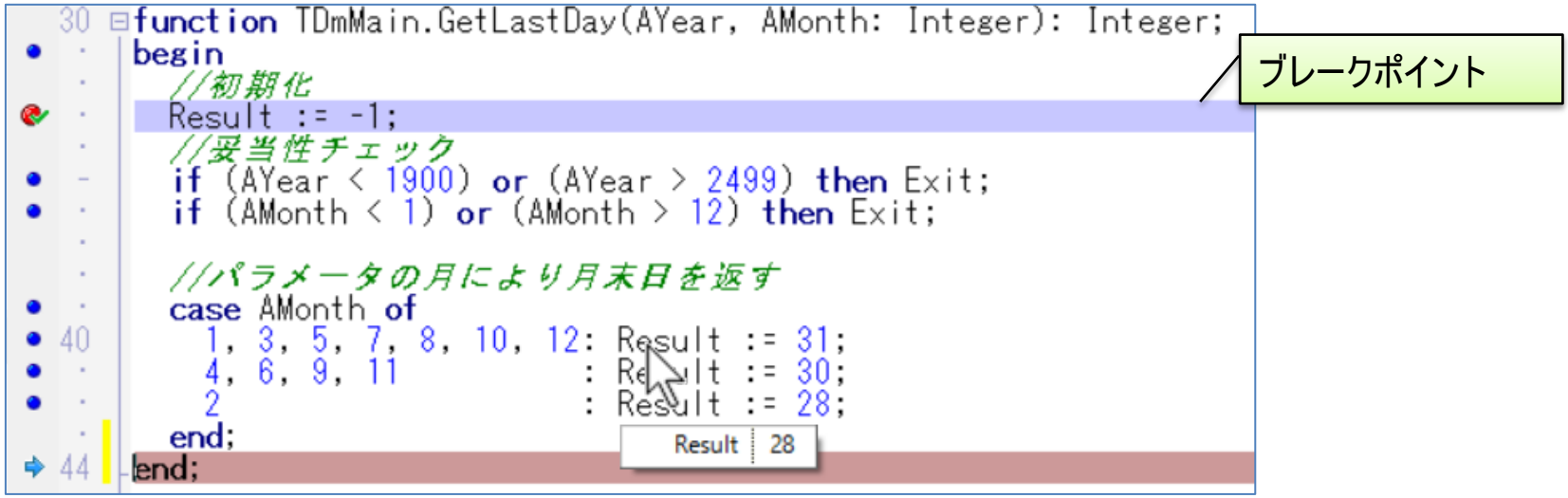
```
type
  TDmMain = class(TDataModule)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    function GetLastDay(AYear, AMonth: Integer): Integer;
  end;
```

ユニットテストでは、メソッド（関数や手続き）単位に仕様どおりに実行できるか確認する。

ユニットテストはどのように実施しているか？

■ 一般的なユニットテストの実施方法

- デバッグ実行
  - ブレークポイントを設定して、デバッグすることで実際に動作させながら値の確認などが行える。



- デバッグ実行の課題
  - ユニットテスト結果を保管することができない。
  - 再テストの際、都度再設定して実行する必要がある。
  - 必要なテストパターンを考慮して画面から実行するのが困難なことがある。

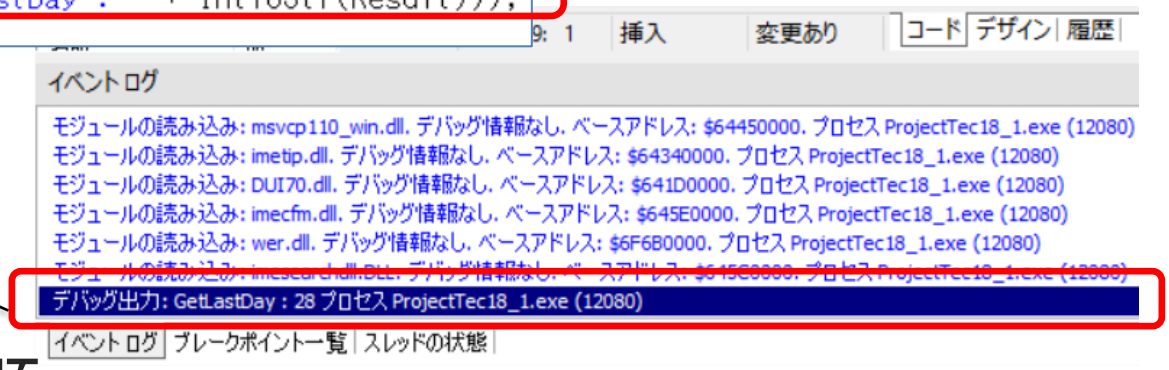
■ 一般的なユニットテストの実施方法

- デバッグプリント
  - 実装コードの中にOutputDebugString手続きを埋め込むことで、実行時にデバッグウィンドウに値を出力することができる。

```
  case AMonth of
    1, 3, 5, 7, 8, 10, 12: Result := 31;
    4, 6, 9, 11          : Result := 30;
    2                    : Result := 28;
  end;

  //処理結果をデバッグウィンドウに出力
  OutputDebugString(PChar('GetLastDay : ' + IntToStr(Result)));
end;
```

デバッグ出力ロジック

デバッグ実行時
イベントログに結果が出力

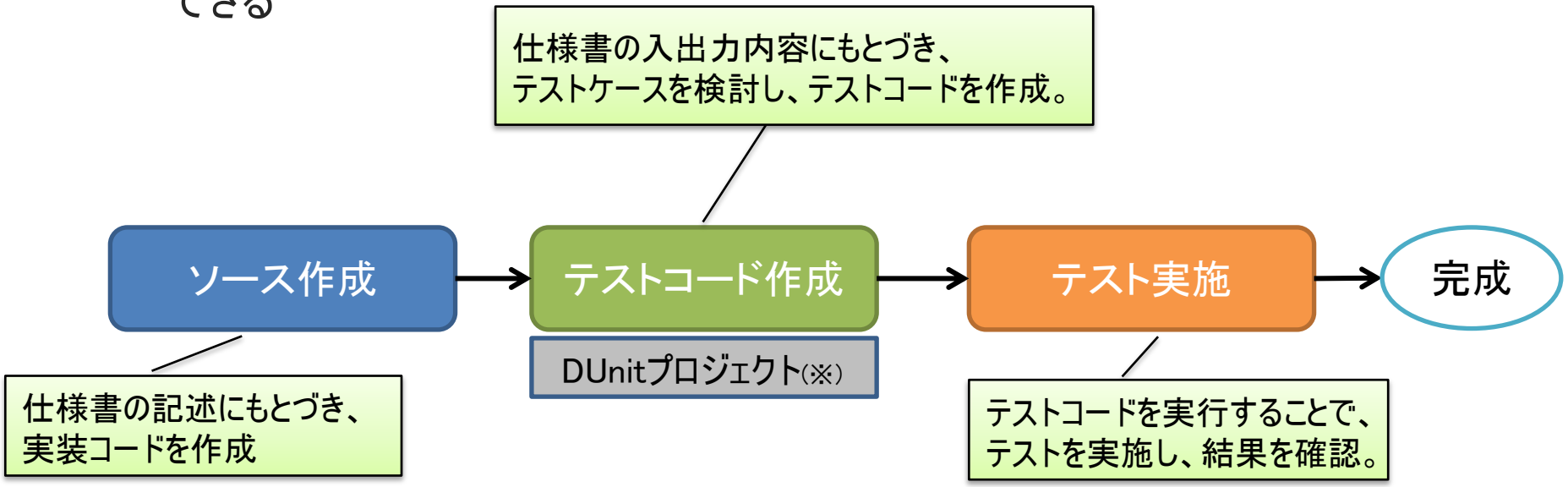


- デバッグプリントの課題
  - 実装コードの中に直接デバッグ用コードを埋め込む必要がある。

ユニットテストの課題を解決する方法はないか？

■ テストフレームワーク DUnit

- ユニットテストを行うためのフレームワーク
  - 実装コードとは、別にテスト用コードを用意し、テスト実施を自動化可能
  - 仕様にもとづくテストケースを用意しておけば、何回でもテストを繰り返すことができる

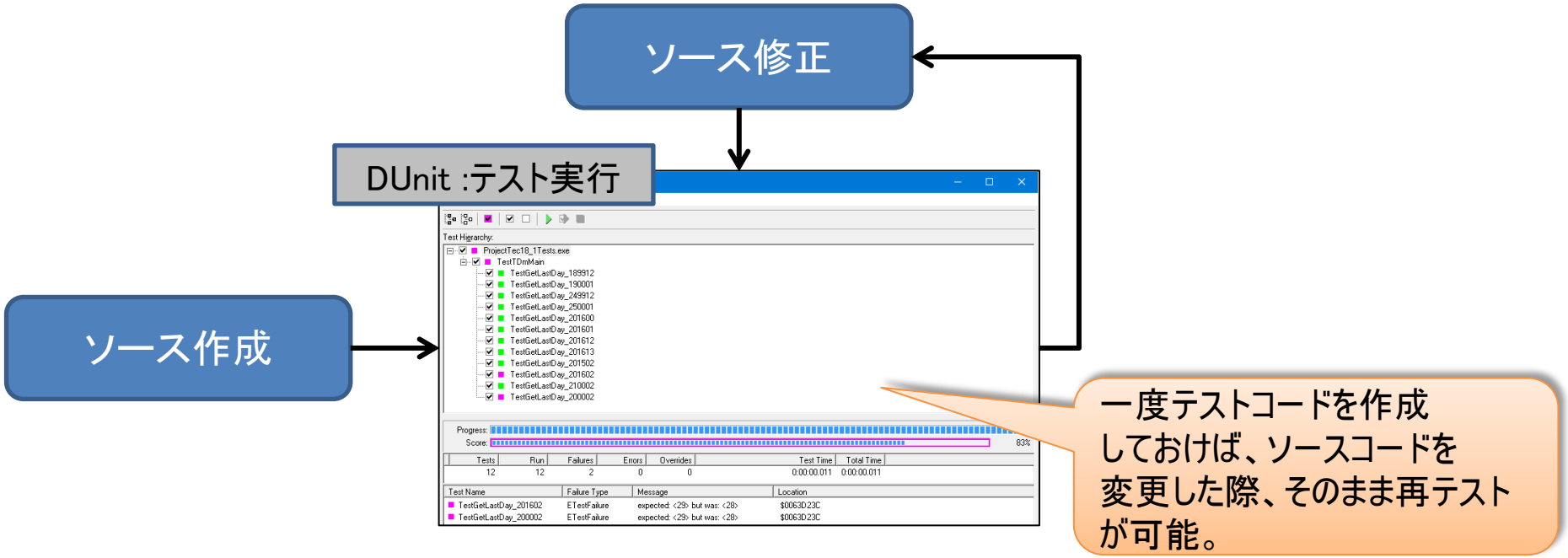


※DUnitは、Delphi/400 Ver.2005以降に統合

※テストケースとは、「この状態でこの入力を行えば、この処理が行われてこの結果が期待される」といった内容を記載したもの。テストフレームワークでは、これをプログラムで表現する。

DUnitを使用するメリット

- ユニットテストが自動化できる
  - テストコードを作成しておけば、何度でも再テストを行える。
    （コードを修正した際のデグレードを防ぐことができる。）
  - ソースコード/プロジェクトに手を加えることなく、テストコードを追加できる。
  - テストコード自体が、テスト実施のエビデンスとなる。



■ サンプルプログラム

- 月末日取得サブルーチン（メソッド） の作成
  - （仕様）
    - パラメータにセットした“年”、“月”の月末日を算出し、結果を返す。
    - “年”の有効範囲は、1900～2499
    - “月”の有効範囲は、1～12
    - “年”“月”の有効範囲が正しくない場合、結果として-1を返す。
  - （書式）
    - function GetLastDay(AYear, AMonth: Integer): Integer;

■ サンプルプログラム

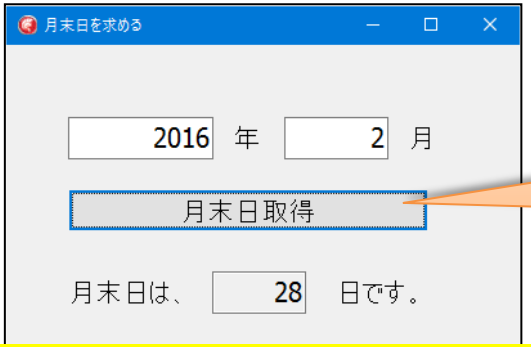
- GetLastDayメソッドのコーディング

```
function TDmMain.GetLastDay(AYear, AMonth: Integer): Integer;
begin
  //初期化
  Result := -1;
  //妥当性チェック
  if (AYear < 1900) or (AYear > 2499) then Exit;
  if (AMonth < 1) or (AMonth > 12) then Exit;

  //パラメータの月により月末日を返す
  case AMonth of
    1, 3, 5, 7, 8, 10, 12: Result := 31;
    4, 6, 9, 11          : Result := 30;
    2                    : Result := 28;
  end;
end;
```

パラメータが対象範囲となっているかのチェック

パラメータの月により、月末日を返す

- 実行

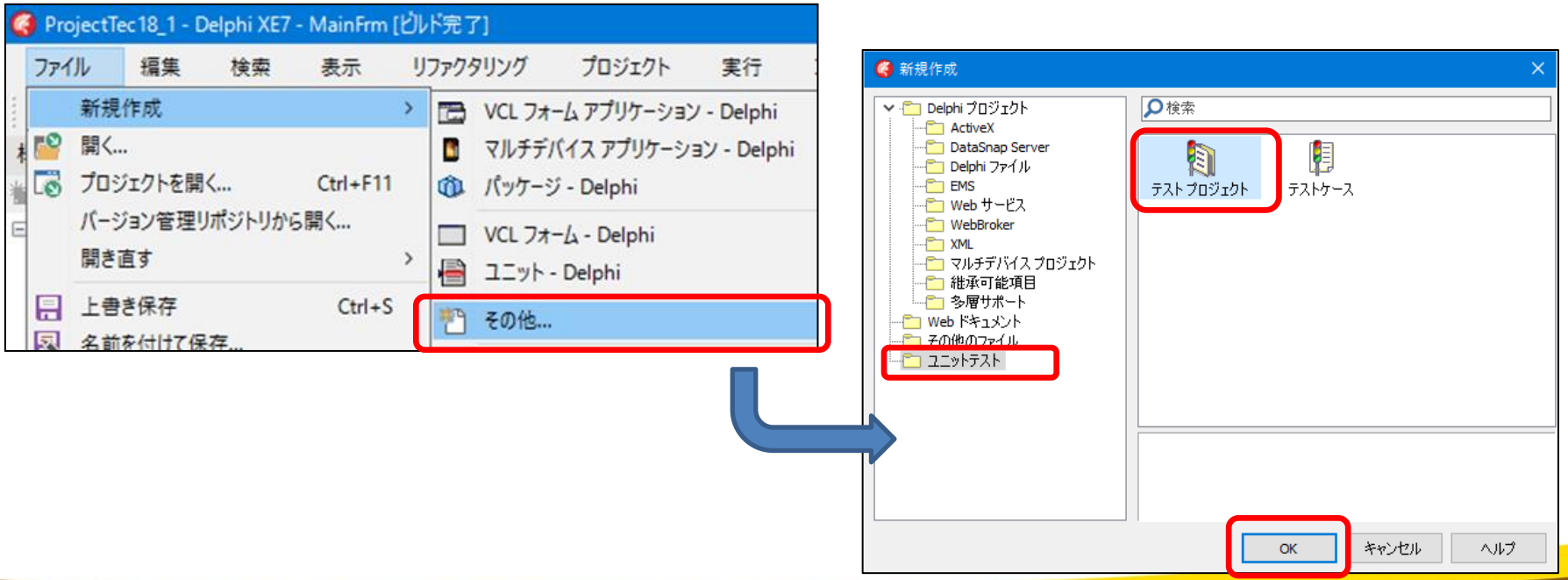


うるう年の考慮がないので、
今年(2016年)は、2月は29日
のはずが、28日と表示。
（メソッドの修正が必要）

GetLastDayメソッドをテストする方法を検討！

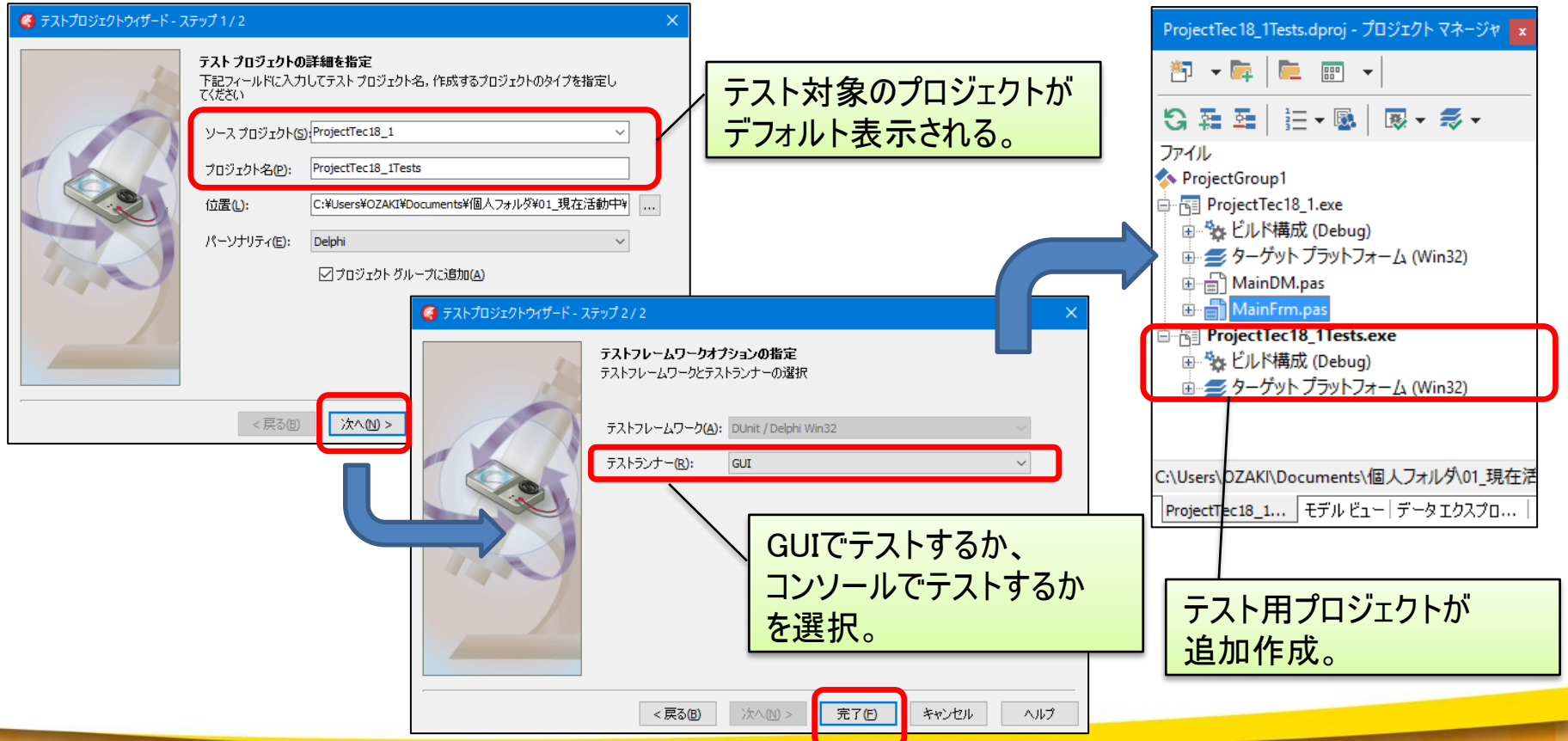
■ ユニットテストプロジェクトの作成方法

- テストプロジェクトの新規作成(1/2)
  - [ファイル]→[新規作成]→[その他]を選択
  - 新規作成ダイアログの左側ツリーより、[ユニットテスト]を選択
  - [テスト プロジェクト]を選択し、[OK]を押下



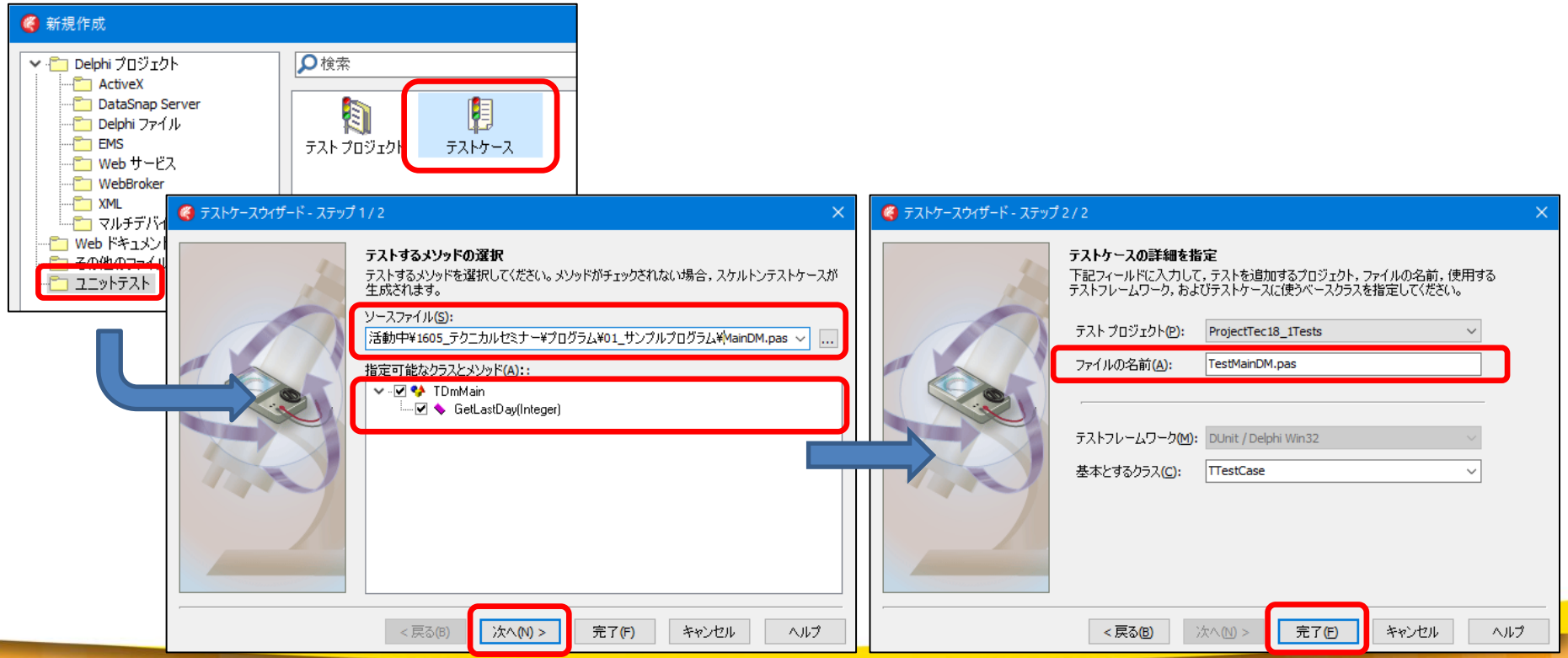
■ ユニットテストプロジェクトの作成方法

- テストプロジェクトの新規作成(2/2)
  - ソースプロジェクト選択、ユニットテスト用プロジェクト名を指定して[次へ]を押下
  - テストランナーを指定して[完了]を押下

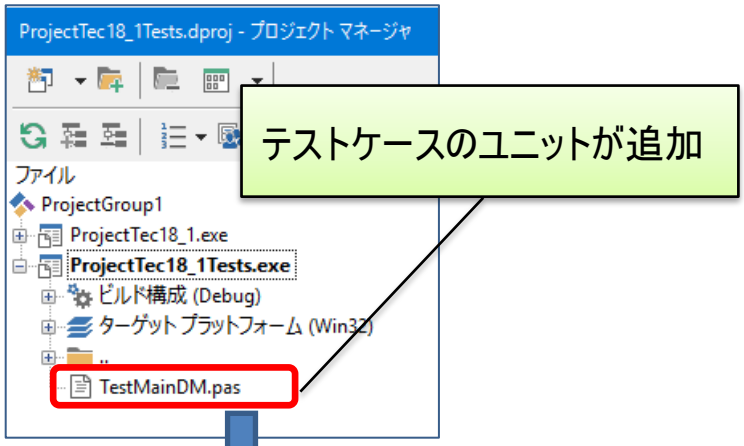


■ テストケースの作成方法

- テストケースの新規作成
  - 新規作成ダイアログの[ユニットテスト]より[テストケース]を選択し、[OK]を押下
  - テストを行うユニットを選択し、テスト対象のサブルーチン（メソッド）を選択
  - ファイル名を指定して[完了]を押下



■ テストケースの作成方法



テストケースのユニットが追加

宣言部

```
type
  // クラスのテスト メソッド TDmMain

  TestTDmMain = class(TTestCase)
  strict private
    FDmMain: TDmMain;
  public
    procedure SetUp; override;
    procedure TearDown; override;
  published
    procedure TestGetLastDay;
  end;
```

実装部

```
implementation

procedure TestTDmMain.SetUp;
begin
  FDmMain := TDmMain.Create;
end;

procedure TestTDmMain.TearDown;
begin
  FDmMain.Free;
  FDmMain := nil;
end;

procedure TestTDmMain.TestGetLastDay;
var
  ReturnValue: Integer;
  AMonth: Integer;
  AYear: Integer;
begin
  // TODO: メソッド呼び出しパラメータのセットアップ
  ReturnValue := FDmMain.GetLastDay(AYear, AMonth);
  // TODO: メソッド結果の検証
end;

initialization
  // テスト ケースをテスト ランナーに登録する
  RegisterTest(TestTDmMain.Suite);
end.
```

テスト開始前に実行
（テストに必要な初期化）

テスト終了後に実行
（終了処理を記述）

テストパラメータ変数

ここにテスト用ロジックを追加

publishedのメソッドに実行したい条件のテストを記述する！

■ テストケースの作成方法

- 例）2016年2月が正しく動作するかのテストケースを作成

```
procedure TestTDmMain.SetUp;
begin
//  FDmMain := TDmMain.Create;
  FDmMain := TDmMain.Create(nil);
end;

procedure TestTDmMain.TestGetLastDay;
var
  ReturnValue: Integer;
  AMonth: Integer;
  AYear: Integer;
  ExpectedValue: Integer;   //期待値
begin
  // TODO: メソッド呼び出しパラメータのセットアップ
  //---- テストで使用するパラメータの定義
  AYear := 2016;
  AMonth := 2;
  //---- 想定される正解値
  ExpectedValue := 29;

  ReturnValue := FDmMain.GetLastDay(AYear, AMonth);
  // TODO: メソッド結果の検証
  //---- 実行結果と想定値との比較
  CheckEquals(ExpectedValue, ReturnValue);
end;
```

データモジュールやフォームのメソッドを対象とする場合、オーナーの指定が必要。

テスト結果として想定される期待値を保持する変数を定義。

テストケース： 2016年02月

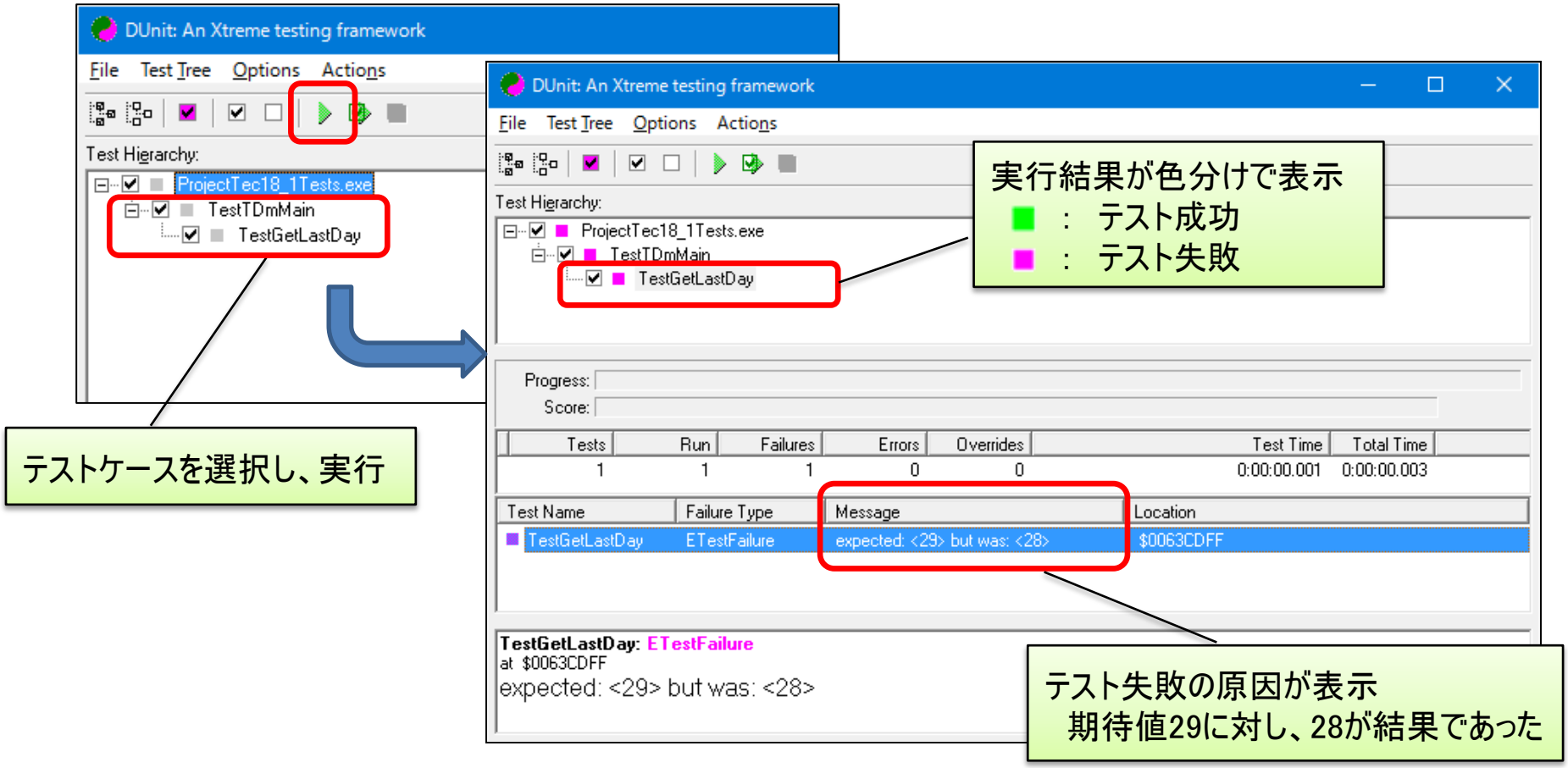
期待したい値： うるう年の為、29日

メソッド実行

テスト結果の確認
CheckEquals手続き ：
2つの値が等しいことを確認

■ テストケースの作成方法

- テストプロジェクトを[実行]



仕様書に基づき、必要なテストケースを同様に追加すればよい！

■ 仕様にもとづくテストケースの作成

- 仕様にもとづくテストケースの作成
  - ブラックボックステスト （入出力仕様に合わせたテスト仕様を作成）
    - 同値分割と限界値分析

同値分割	正常に処理される有効同値クラスと、エラー処理される無効同値クラスに分けてそれぞれの代表的な値をテストデータにする。
境界値分析	有効値と無効値の境界値をテストデータにする

  - 必要なテストケースのパターンを検討
    - 年の境界値　1899, 1900 … 2499, 2500
    - 月の境界値　0, 1 … 12, 13
    - 年の同値分割　2015年2月 （平年：年が4で割り切れない）
      2016年2月 （うるう年：年が4で割り切れる）
      2100年2月 （平年：年が100で割り切れる）
      2000年2月 （うるう年：年が400で割り切れる）

■ 仕様にもとづくテストケースの作成

宣言部

```
type
  // クラスのテスト メソッド TDmMain

  TestTDmMain = class(TTestCase)
  strict private
    FDmMain: TDmMain;
  public
    procedure SetUp; override;
    procedure TearDown; override;
    procedure TestGetLastDay(AYear, AMonth, ExpectedValue: Integer);
  published
    procedure TestGetLastDay_189912; //-- 1899年
    procedure TestGetLastDay_190001; //-- 1900年
    procedure TestGetLastDay_249912; //-- 2499年
    procedure TestGetLastDay_250001; //-- 2500年
    procedure TestGetLastDay_201600; //-- 0月
    procedure TestGetLastDay_201601; //-- 1月
    procedure TestGetLastDay_201612; //-- 12月
    procedure TestGetLastDay_201613; //-- 13月
    procedure TestGetLastDay_201502; //-- 2015年2月
    procedure TestGetLastDay_201602; //-- 2016年2月
    procedure TestGetLastDay_210002; //-- 2100年2月
    procedure TestGetLastDay_200002; //-- 2000年2月
  end;
```

【P.36の宣言部を修正】
- テストケースを汎用化するために、引数を追加
- published → public に変更

テストケースに沿ったメソッドを追加で宣言

実装部（P.37のソースを修正）

```
procedure TestTDmMain.TestGetLastDay(AYear, AMonth, ExpectedValue: Integer);
var
  ReturnValue: Integer;
begin
  //---- テスト実行
  ReturnValue := FDmMain.GetLastDay(AYear, AMonth);
  //---- 実行結果と想定値との比較
  CheckEquals(ExpectedValue, ReturnValue);
end;
```

テスト実行と結果の評価

■ 仕様にもとづくテストケースの作成

テストケース実装部（一部抜粋）

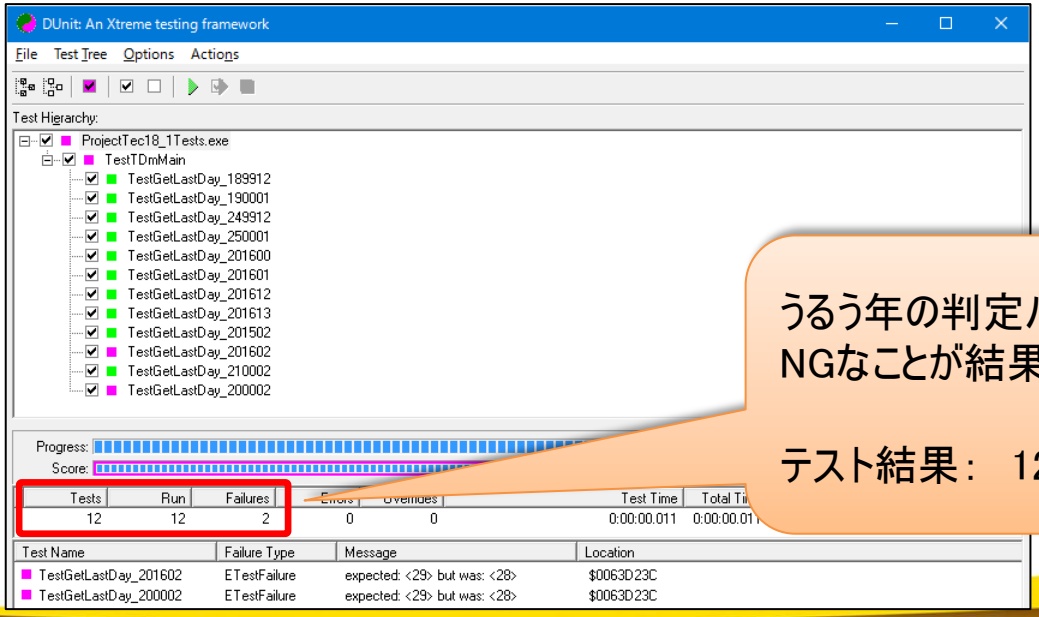
```
procedure TestTDmMain.TestGetLastDay_189912;
begin
  TestGetLastDay(1899, 12, -1);
end;

procedure TestTDmMain.TestGetLastDay_190001;
begin
  TestGetLastDay(1900, 1, 31);
end;

procedure TestTDmMain.TestGetLastDay_200002;
begin
  TestGetLastDay(2000, 2, 29);
end;
```

1899年は有効範囲外の為、-1となる

1900年は有効の為、大の月の31となる

2000年はうるう年の為、29となる

- テスト実施



うるう年の判定パターンが
NGなことが結果としてわかる

テスト結果：　12件中2件NG

GetLastDayメソッドのロジック修正

- うるう年判定ロジックを追加

```
function TDmMain.GetLastDay(AYear, AMonth: Integer): Integer;
const
  Days: array[1..12] of Integer = (31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31);
var
  UruuFlag: Boolean;
begin
  //初期化
  Result := -1;
  //妥当性チェック
  if (AYear < 1900) or (AYear > 2499) then Exit;
  if (AMonth < 1) or (AMonth > 12) then Exit;

  //うるう年の判定
  UruuFlag := (((AYear mod 4) = 0) and
    (((AYear mod 100) <> 0) or ((Ayear mod 400) = 0)));

  //結果を返す
  Result := Days[AMonth];
  if UruuFlag and (AMonth = 2) then Result := Result + 1;
end;
```

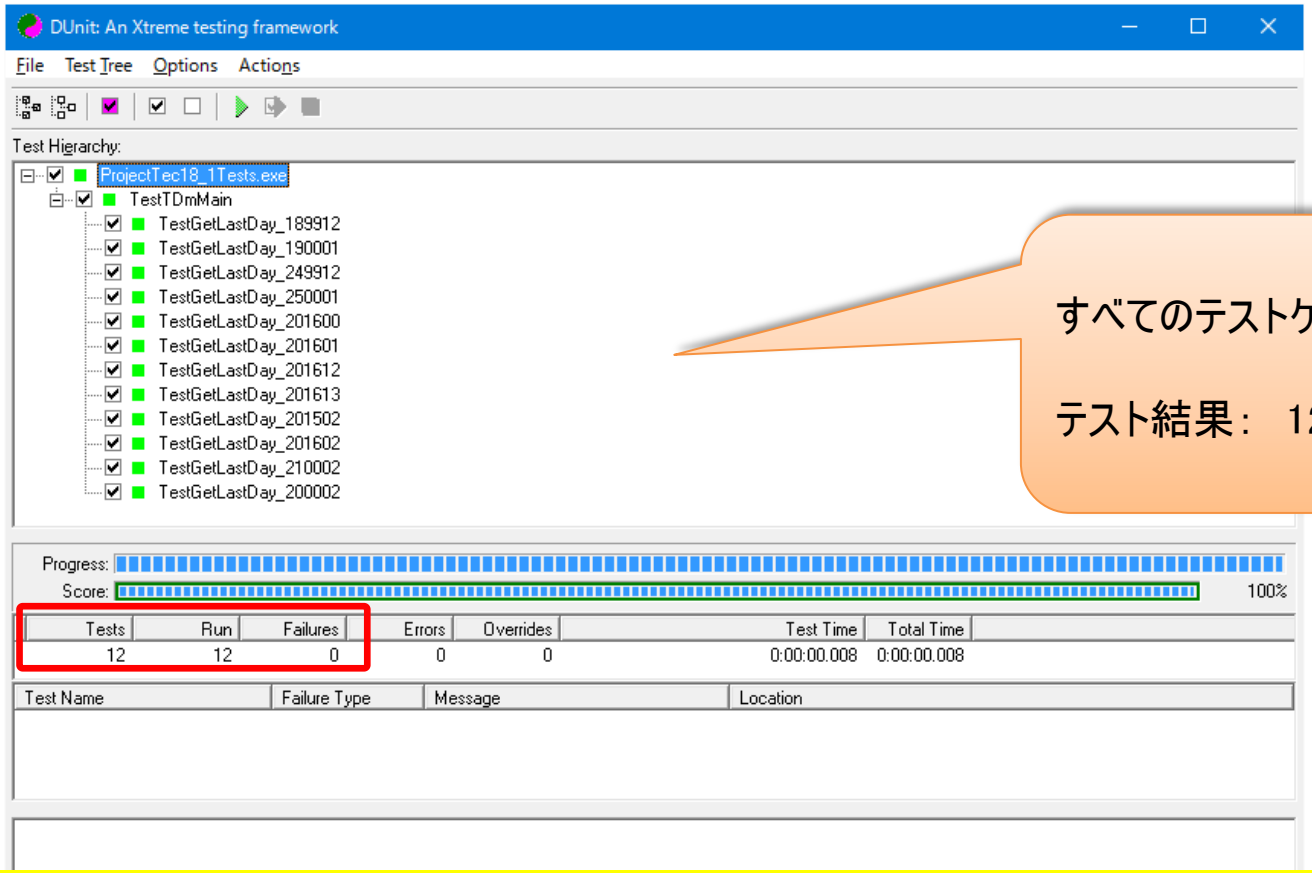
うるう年の判定ロジック

（年が4で割り切れる） かつ
（100で割り切れない
あるいは400で割り切れる） → うるう年

配列を使用し処理を簡略化

GetLastDayメソッドのロジック修正

- ロジック修正後も、容易に再テスト可能



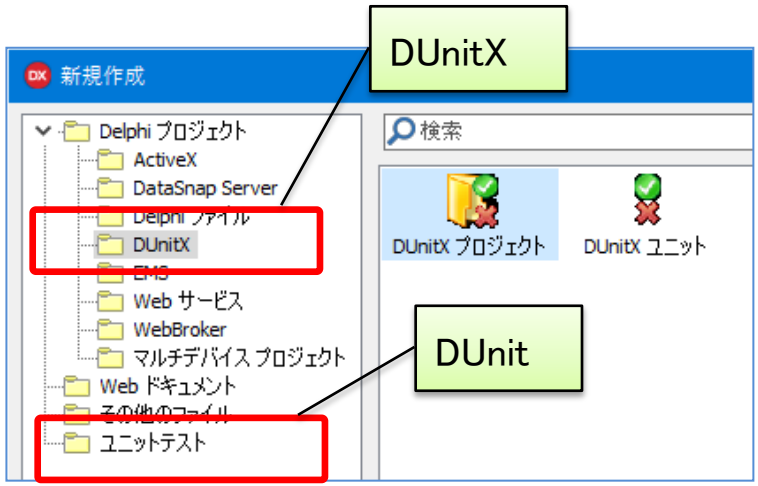
すべてのテストケースに合格

テスト結果： 12件中12件OK

必要なテストケースがあれば、何度でも再テスト可能でデグレードを防止！

（参考）DUnitXについて

- 新しいテストフレームワークDUnitX
  - Delphi/400 10 Seattleからは、DUnitXが推奨テストフレームワークとなっている。（DUnitも使用可能）



＜DUnitXの概要＞

http://docwiki.embarcadero.com/RADStudio/Seattle/ja/DUnitX_の概要

DUnitXは、Delphi/400 Ver.2010以降であれば、下記よりダウンロードして使用可能

https://github.com/VSoftTechnologies/DUnitX

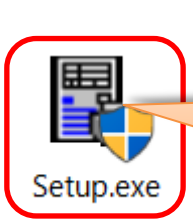
DUnitで作成したテストプロジェクトは、DUnitXに変換可能

http://docwiki.embarcadero.com/RADStudio/Seattle/ja/DUnit_テストを_DUnitX_に変換する方法

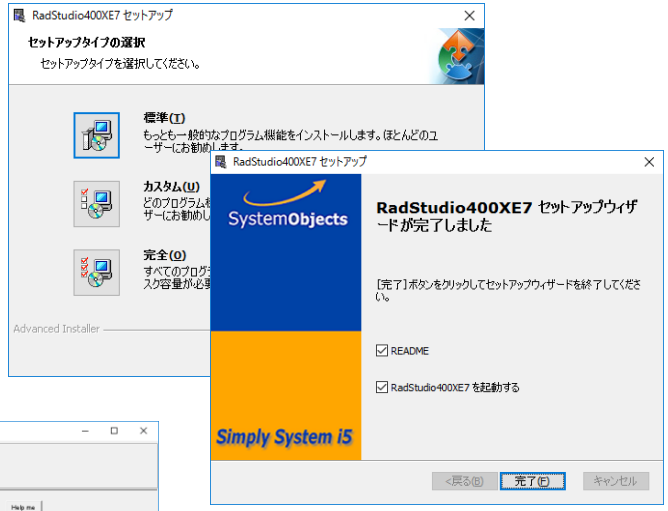
３．Delphi/400サイレントインストーラ作成方法

Delphi/400運用環境のセットアップ

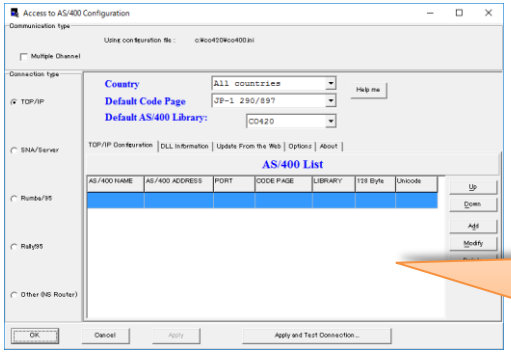
- 運用版CD-ROMよりインストール
  - SETUP.exeを実行
  - ウィザードに従って、インストール条件を指定
    1. 使用許諾同意
    2. インストール先の指定
    3. セットアップタイプ
       カスタムの場合セットアップ機能を指定
    4. インストール開始
  - インストール実行、終了後完了画面が表示。[完了]を押下。



Setup.exeを
ダブルクリックして実行



- Configration設定
  - 接続するIBM i (AS/400)の定義を登録
    接続名、IPアドレス、コードページ
  - 接続テスト
  - 保存、終了



Configration で
接続情報を定義

効率よく運用環境を導入する方法はないか？

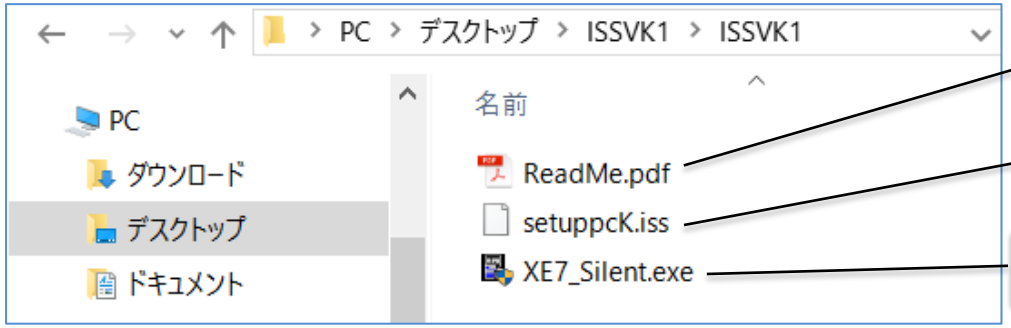
■ 運用版サイレントインストール

- サイレントインストールとは？
  - インストール実行時のダイアログによるインストール指示やConfigrationの初期設定を自動化するインストール方法
  - ミガロ. Delphi/400メンテナンスページより取得可能
  - サイレントインストールは、運用版のみ （開発版は通常のセットアップで導入）

運用版メンテナンスページ

■ 運用版サイレントインストール

- ファイルの内容



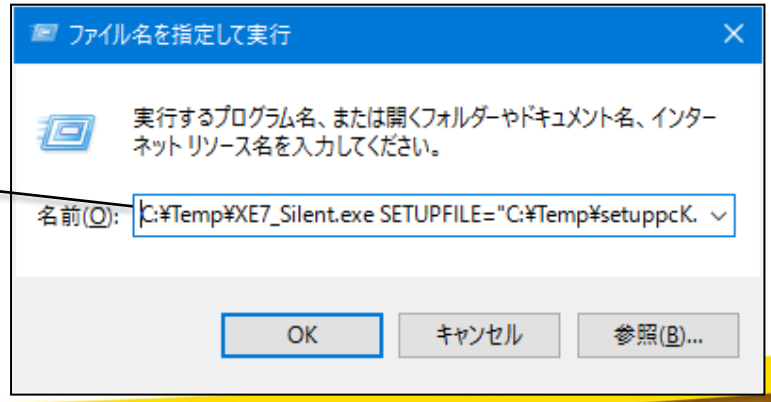
説明資料（使用方法、issファイル設定方法）

セットアップ応答ファイル（ひな形）

サイレントインストール用モジュール

- サイレントインストーラ使用方法
  1. issファイル(応答ファイル)にインストールの指定やConfigrationの定義を登録
  2. サイレント用インストーラ(XE7_Silent.exe)に応答ファイルを指定して実行

【実行コマンド例】
C:¥Temp¥XE7_Silent.exe SETUPFILE=″C:¥Temp¥setuppcK.iss″

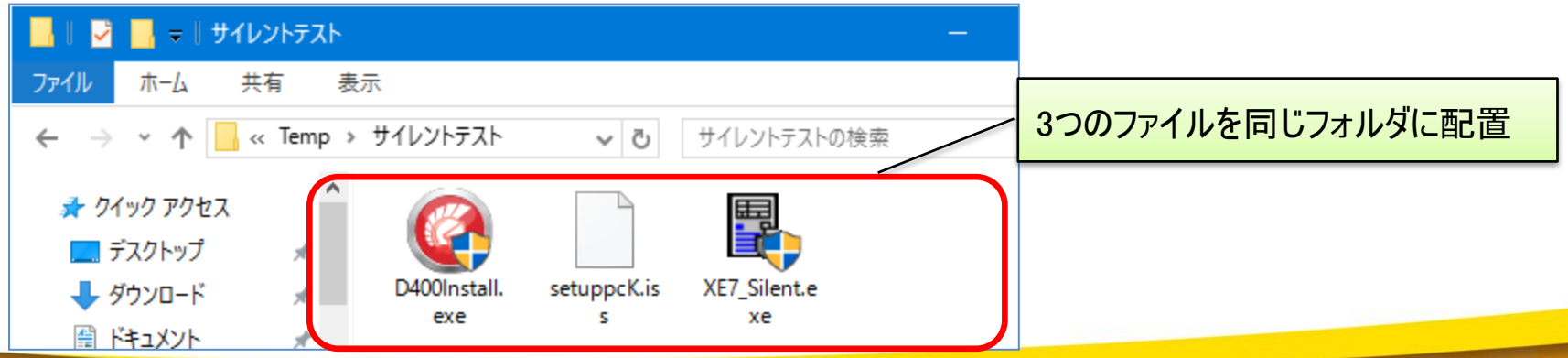


■ 起動用バッチプログラムの作成

- サイレントインストーラ起動バッチプログラムをDelphiで作成
  - プログラム化することにより、インストール前後に独自の処理を追加可能

    【例】
    - 旧バージョンをアンインストールする処理を追加
      - アンインストーラの実行処理、旧バージョンフォルダの削除やWin.iniのクリア処理等
    - Configrationに複数の接続先を指定可能
      - ALIASES.CFGファイル（接続先定義）をCO420フォルダにコピー

- 実行環境
  - バッチプログラムと、サイレントインストーラを同じフォルダに配置して、バッチプログラム(D400Install.exe)を実行



■ 起動用バッチプログラムの作成

MainDM.pas（データモジュール） 一部抜粋

```
procedure TdmMain.Execute;
begin
  //実行中ダイアログ生成
  frmMain := TfrmMain.Create(nil);
  try
    //実行中ダイアログ表示
    frmMain.Show;
    Application.ProcessMessages;
    //運用版のインストール
    Install;
  finally
    //実行中ダイアログ終了
    frmMain.Close;
    FreeAndNil(frmMain);
  end;
end;

procedure TdmMain.Install;
var
  sExePath: String;
begin
  try
    sExePath := ExtractFilePath(ParamStr(0));
    ExecProcess(sExePath + 'XE7_Silent.exe SETUPFILE="' + sExePath + 'setuppcK.iss"');
  except
    on E: Exception do
    begin
      WriteLog(E.Message); //エラーログ出力
      raise;
    end;
  end;
end;
```

メインルーチン

実行中メッセージフォームを表示

コマンドを実行

付録のCD-ROMに全ソースを収録。

■ 起動用バッチプログラムの作成

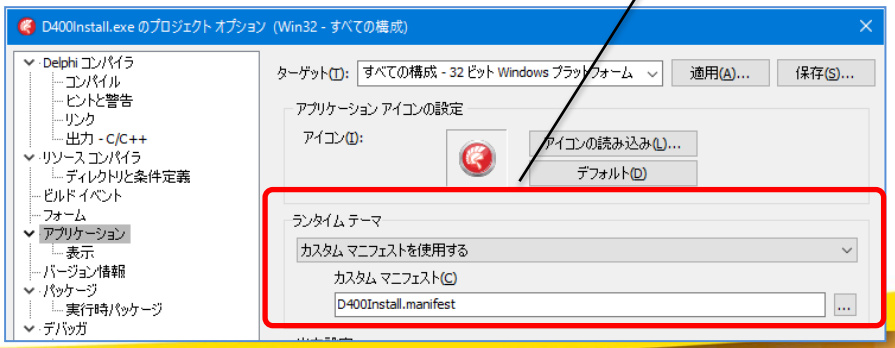
- マニフェストファイルの作成
  - UTF-8形式で下記ファイルを作成 （"[プロジェクト名].manifest"としてファイル保存）

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>↓
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">↓
  <assemblyIdentity ↓
    version="1.0.0.0" ↓
    processorArchitecture="*" ↓
    name="DelphiApplication1"↓
    type="win32"/>↓
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">↓
    <security>↓
      <requestedPrivileges>↓
        <requestedExecutionLevel ↓
          level="requireAdministrator"/>↓
      </requestedPrivileges>↓
    </security>↓
  </trustInfo>↓
</assembly>[EOF]
```

実行時、「管理者」への昇格が必須となる。

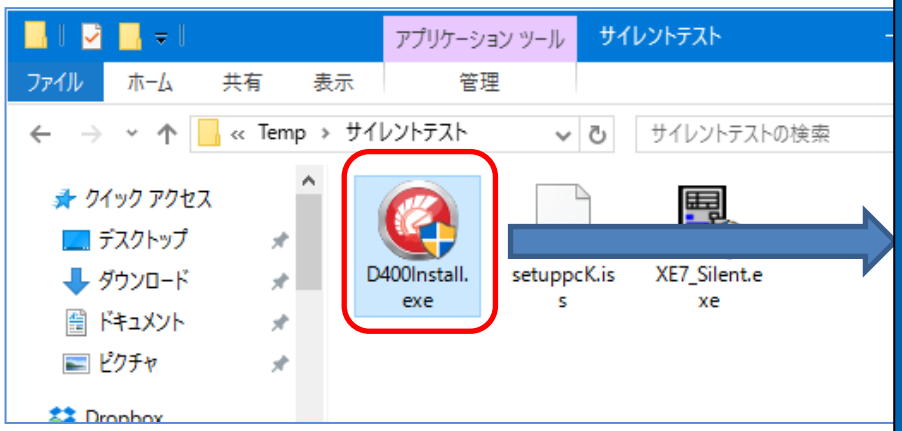
マニフェストファイルを指定。

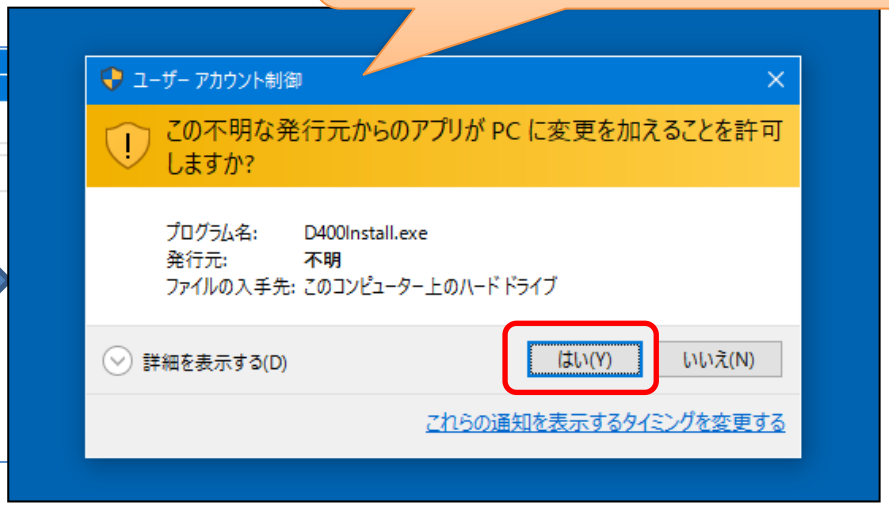
- プロジェクトに対し、マニフェストファイルを指定
  - [ツール]→[オプション]よりプロジェクトオプションを開き、[アプリケーション]のランタイムテーマを「カスタムマニフェストを使用する」に変更。



■ 起動用バッチプログラムの作成

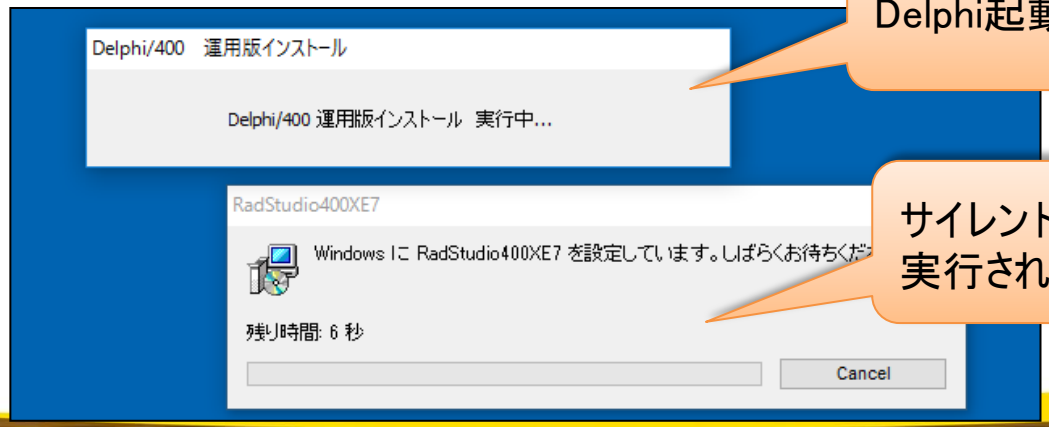
- バッチプログラムの実行
  - D400Install.exe を実行





管理者に昇格して実行してよいかの確認画面が表示

  - 管理者としてバッチプログラムが実行できる



Delphi起動バッチプログラム

サイレントインストーラにより実行された経過画面（応答不要）

まとめ

■ まとめ

- モバイル対応したWebアプリを構築したい。
  - IntraWebとjQuery Mobileとの連携によるスマートデバイス画面の作成
  - 動的なリストビューの作成
  - リストビューとイベントの割り当て

- プログラムの品質を向上したい。
  - ユニットテストフレームワーク DUnit の概要とメリット
  - ユニットテスト手順

- Delphi/400の運用環境を効率よく導入したい。
  - サイレントインストールの使用方法
  - Delphiを使用した起動用バッチプログラムの紹介

ご清聴ありがとうございました。