開発者が知りたいDelphi実践テクニック！
明日から使えるテクニック集

第32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　営業・営業推進課
尾崎　浩司

embarcadero®
DEVELOPER CAMP

自己紹介

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®

SystemObjects

Advanced Business Partner IBM

RAD事業部 営業・営業推進課　課長
尾崎　浩司（おざき　こうじ）

- Delphi/400等弊社取り扱い製品の販売促進や製品販売の技術支援を中心に活動
- 各種イベントや弊社主催セミナーでの講師等も多数担当

embarcadero® DEVELOPER CAMP

アジェンダ

システム開発の中で役立つ実践的な技術トピックを厳選し、
デモプログラムやソース解説を交えてご紹介

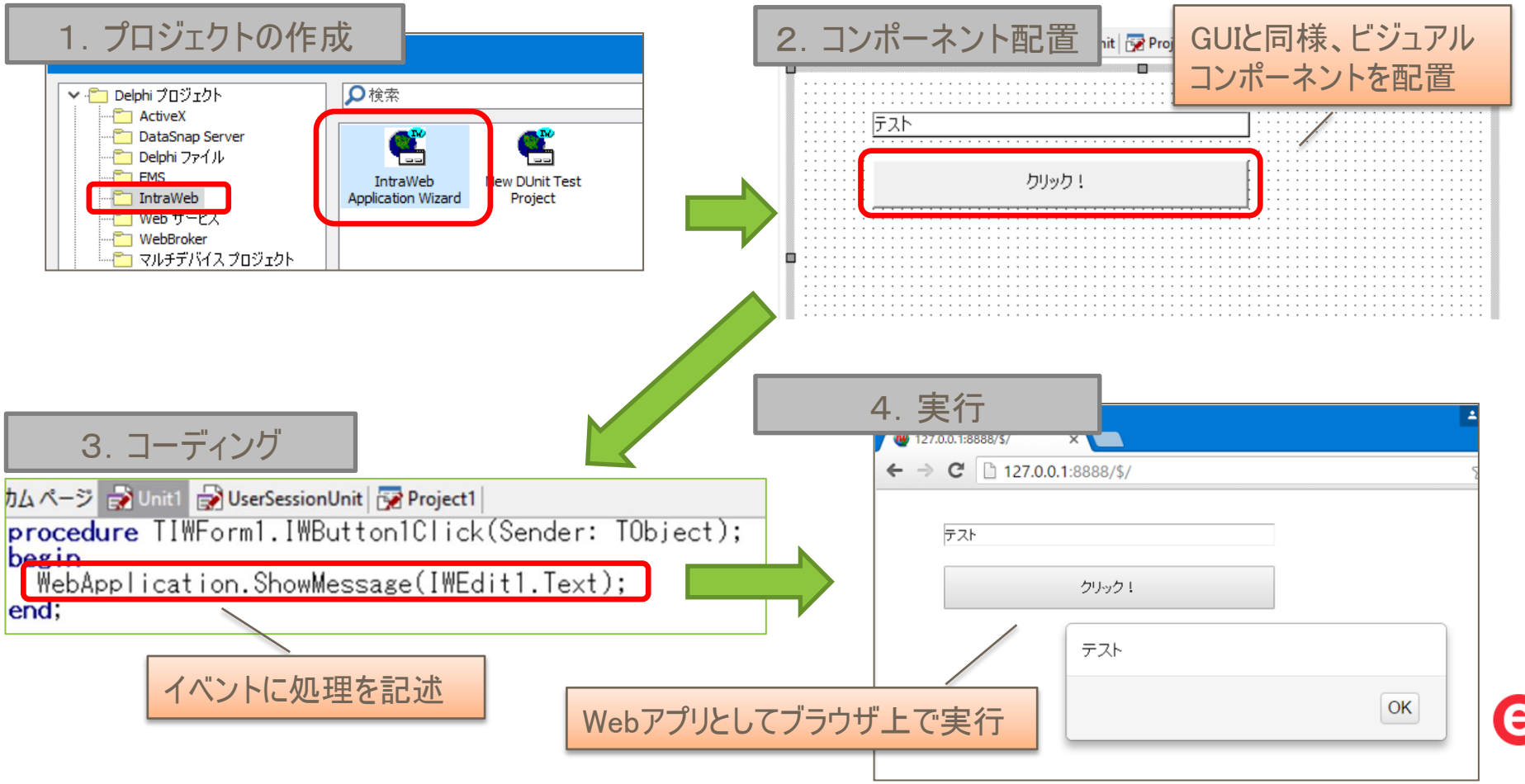
- IntraWebにおけるスマートデバイス向け
  Webアプリ作成テクニック
- ユニットテストフレームワークについて
- モバイルアプリケーション開発のコツ

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

embarcadero DEVELOPER CAMP

DelphiによるWebアプリ開発

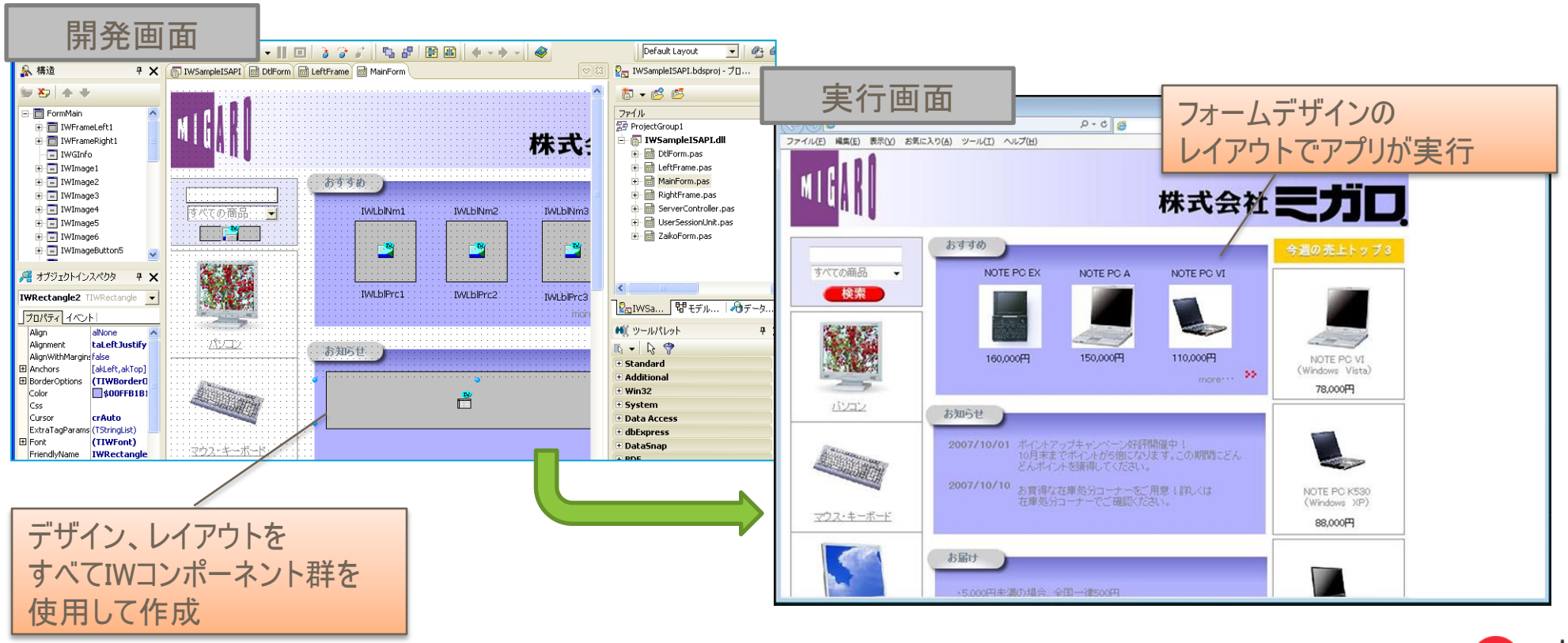
- IntraWeb
  - GUIフォームアプリ同様の手法でWebアプリが構築できるフレームワーク



embarcadero DEVELOPER CAMP

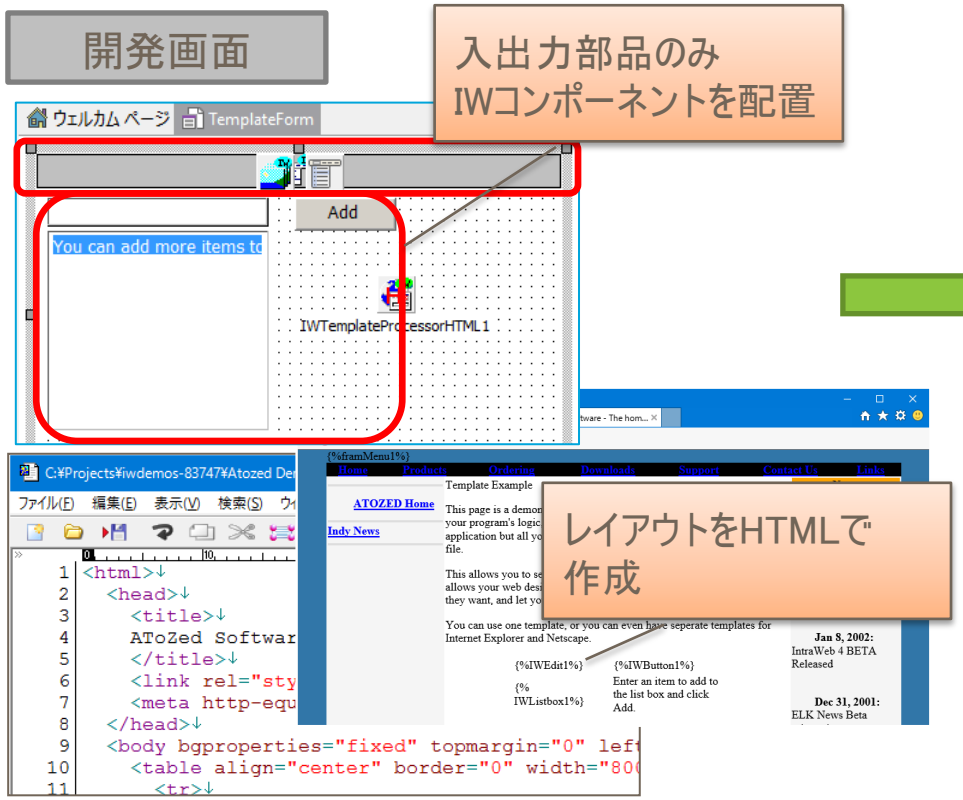
IntraWebでのWeb画面作成方法

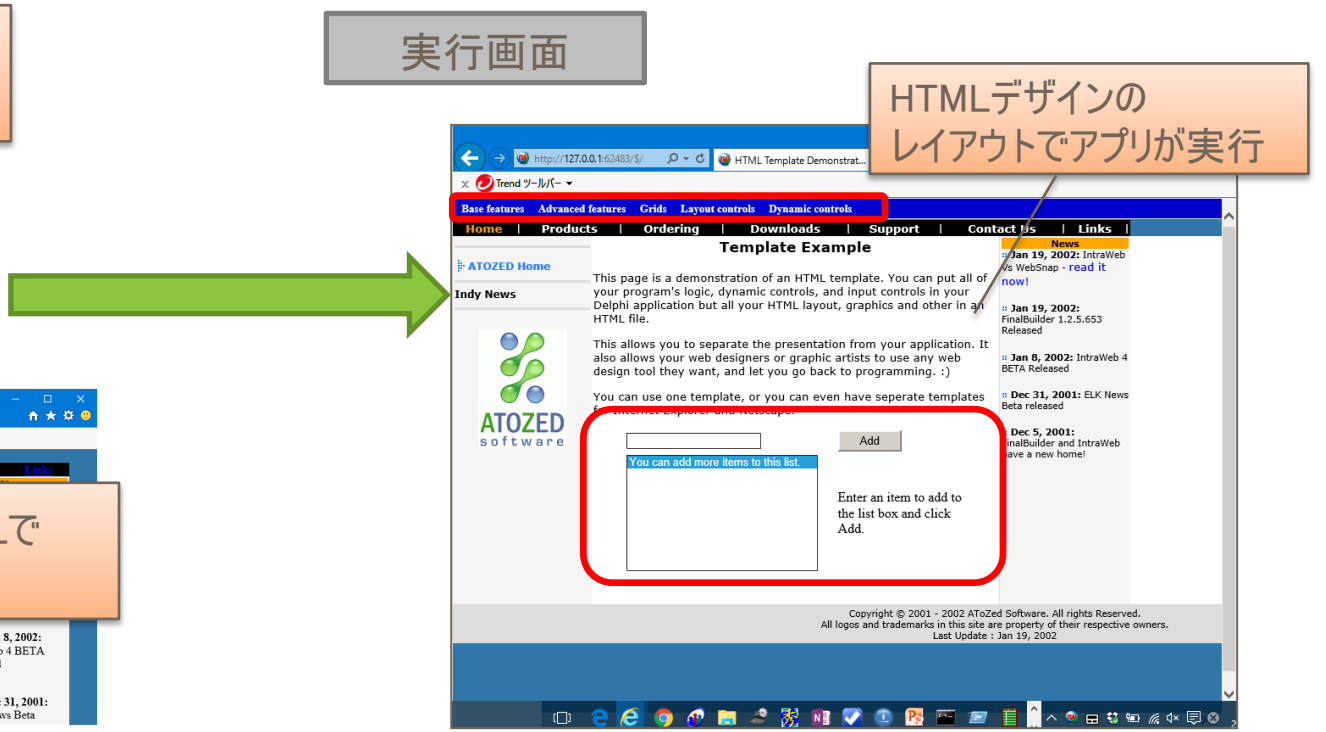
- 方法１：フォームにコンポーネントを配置した画面作成
  - レイアウトをGUIアプリと同様、すべてコンポーネントで構築できる
  - 画面のデザインを変更したい場合、フォームを直接修正する



IntraWebでのWeb画面作成方法

- 方法２：テンプレートHTMLを使用した画面作成
  - デザインをHTMLを使用して作成。実行時にフォーム部品とマージして表示。
  - デザイン変更時はテンプレートHTMLのみ変更すればよい。





スマートデバイスに最適な画面の検討

- IntraWeb
  - 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画面とはどんなものか？

スマートデバイスに最適な画面の検討

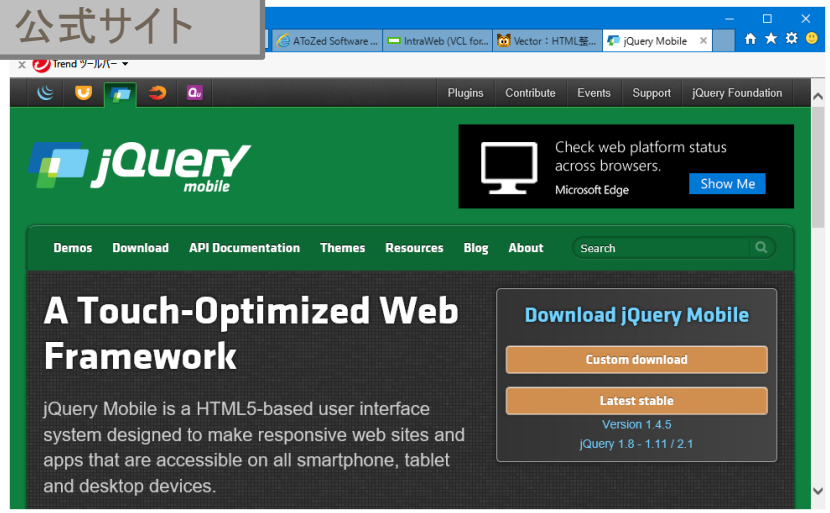
- スマートデバイスの特徴
  - 種類や端末により、サイズや解像度がマチマチ
  - 操作は、タッチやスワイプなど指を使用　（細かな入力などは不向き）

→　スマートデバイスに最適な画面とは？
  - 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

スマートデバイスに最適な画面の検討

- モバイルWeb開発用フレームワーク　“jQuery Mobile”
  - スマートデバイスに最適化した画面デザインや部品を提供するフレームワーク
    - 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/

（2016/05現在　最新版は、Ver.1.4.5）

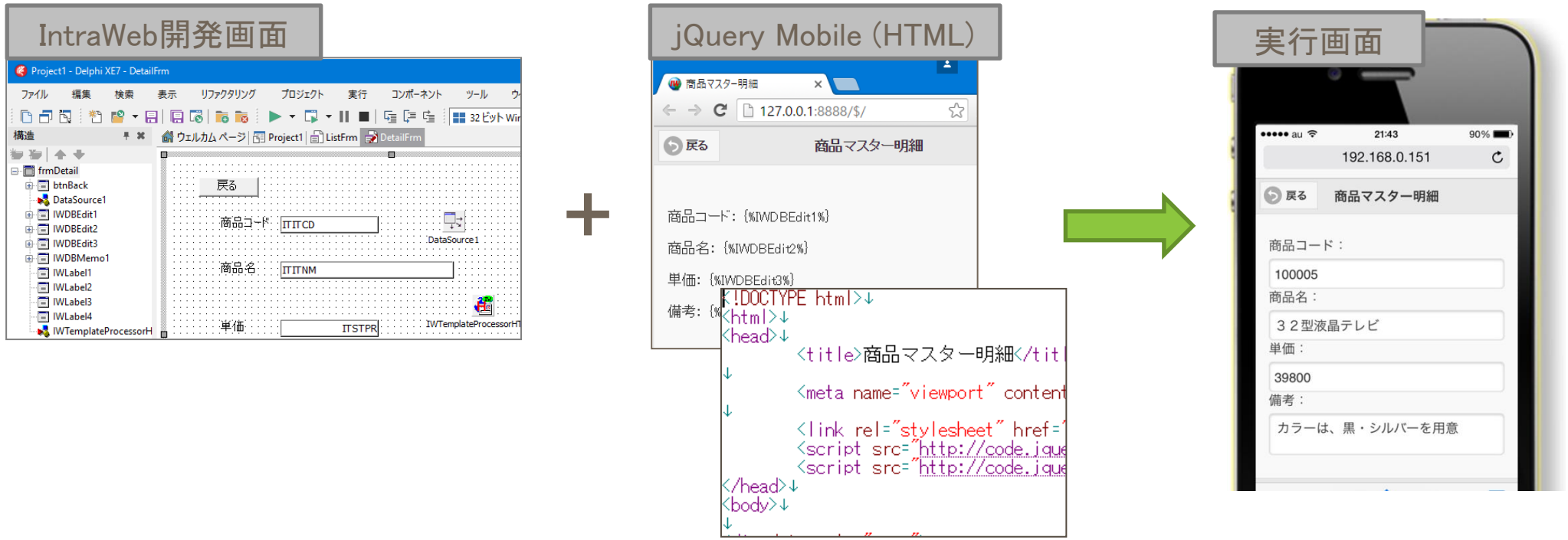
使用されているサイト例

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？

IntraWebでのWeb画面作成方法

- IntraWebとjQuery Mobileとの連携
  - jQueryMobileに用意されている画面デザインや部品は、HTMLで記述



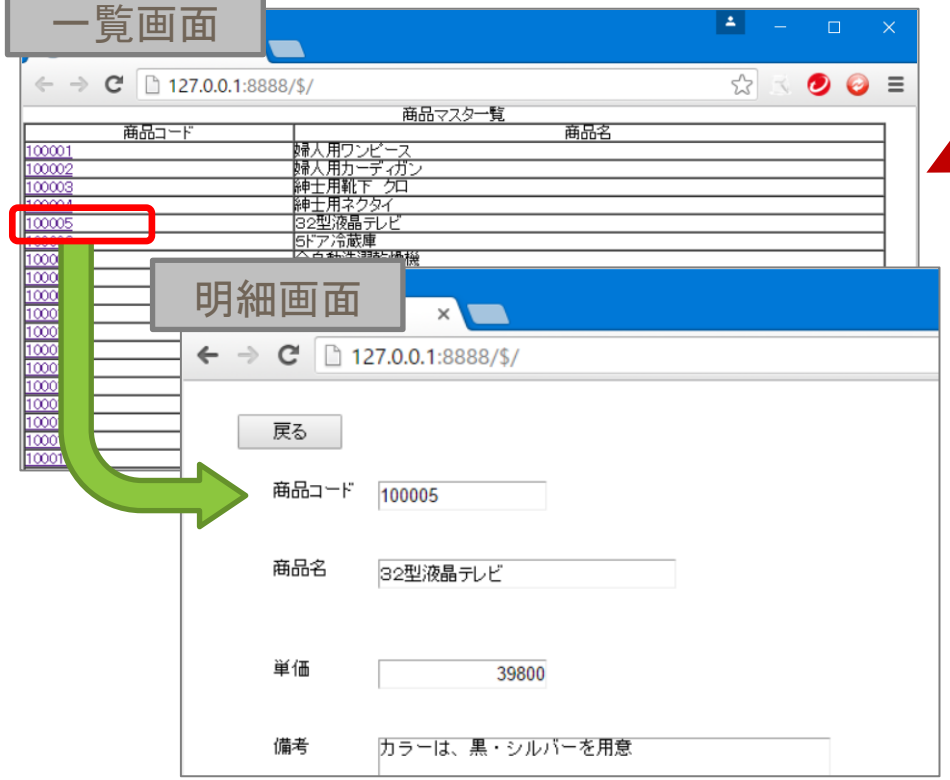
jQuery Mobileとの連携でスマートデバイスに最適な画面を実現可能！

サンプルプログラム

- IntraWebアプリをスマートデバイスに最適化することを検討
  - 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

DB Server

商品マスタ　（MITEMP）	
ITITCD	商品コード
ITITNM	商品名
ITSTPR	標準単価
ITRMRK	商品備考

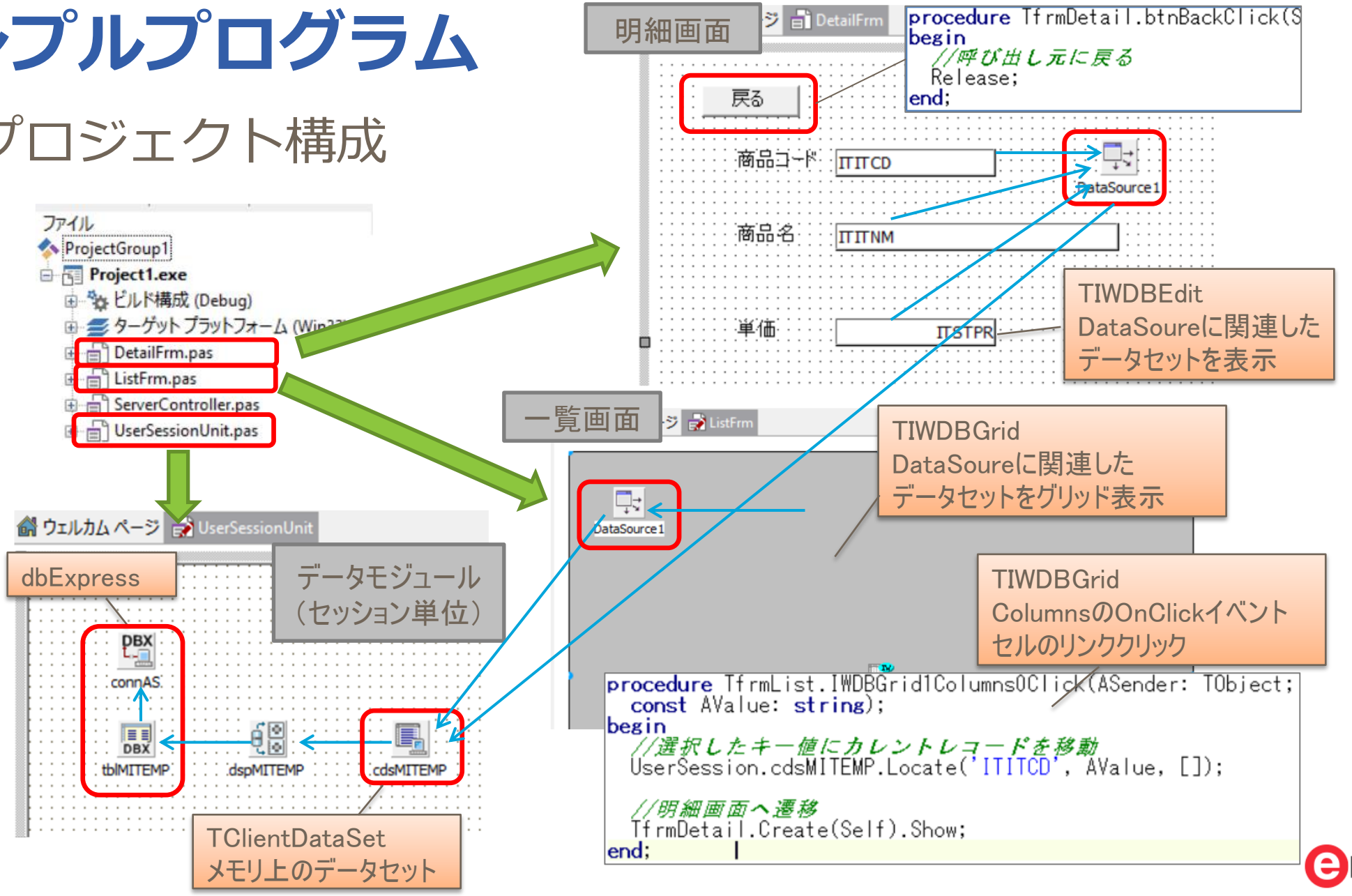
完成イメージ

一覧画面

明細画面

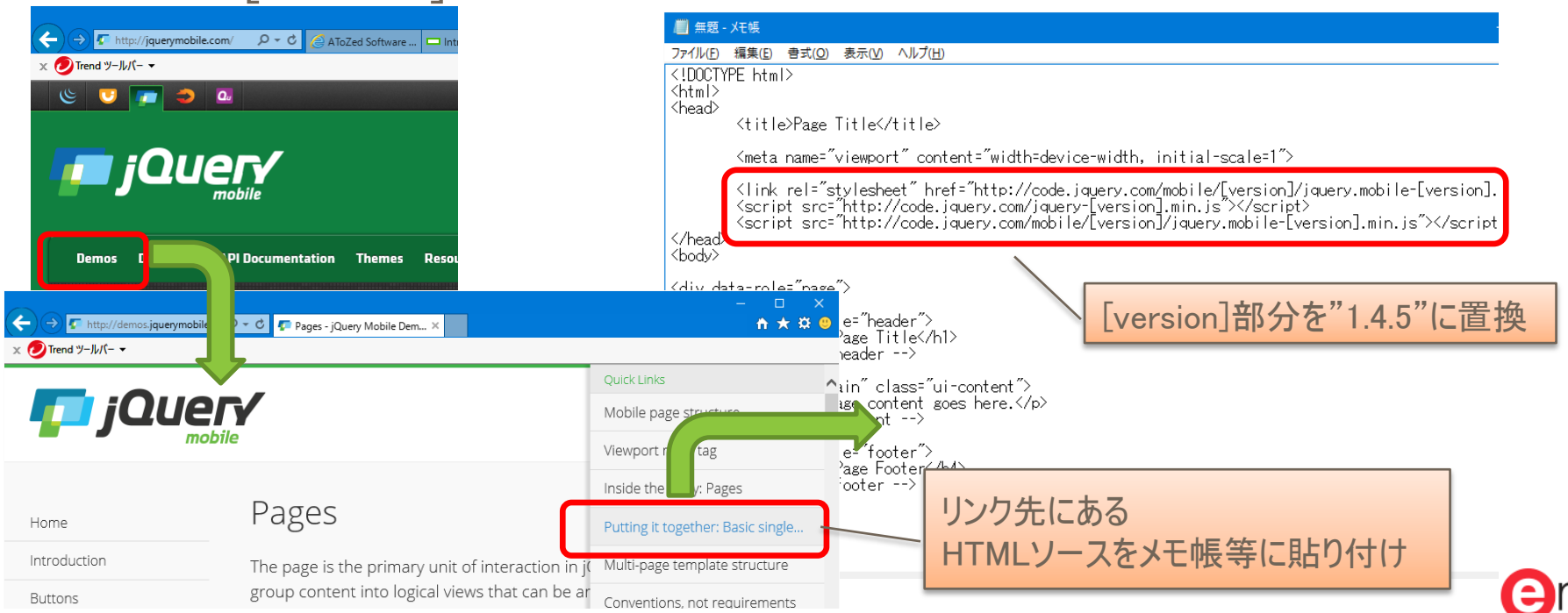
サンプルプログラム

- 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]
```

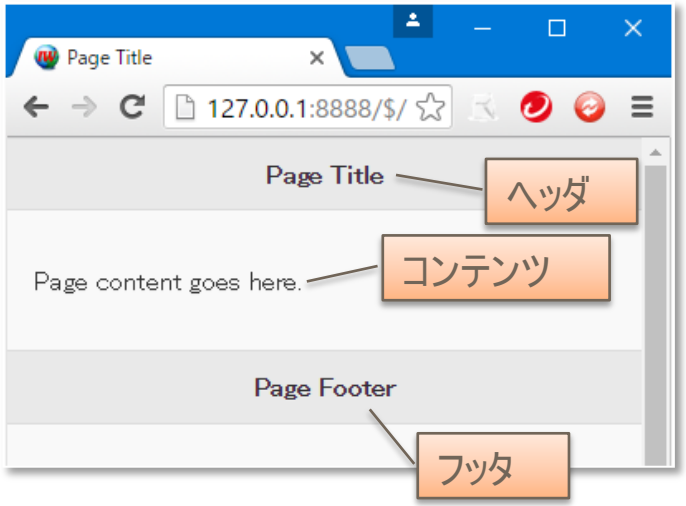
ページ幅、拡大率の設定

ページ全体

ヘッダ領域

コンテンツ領域

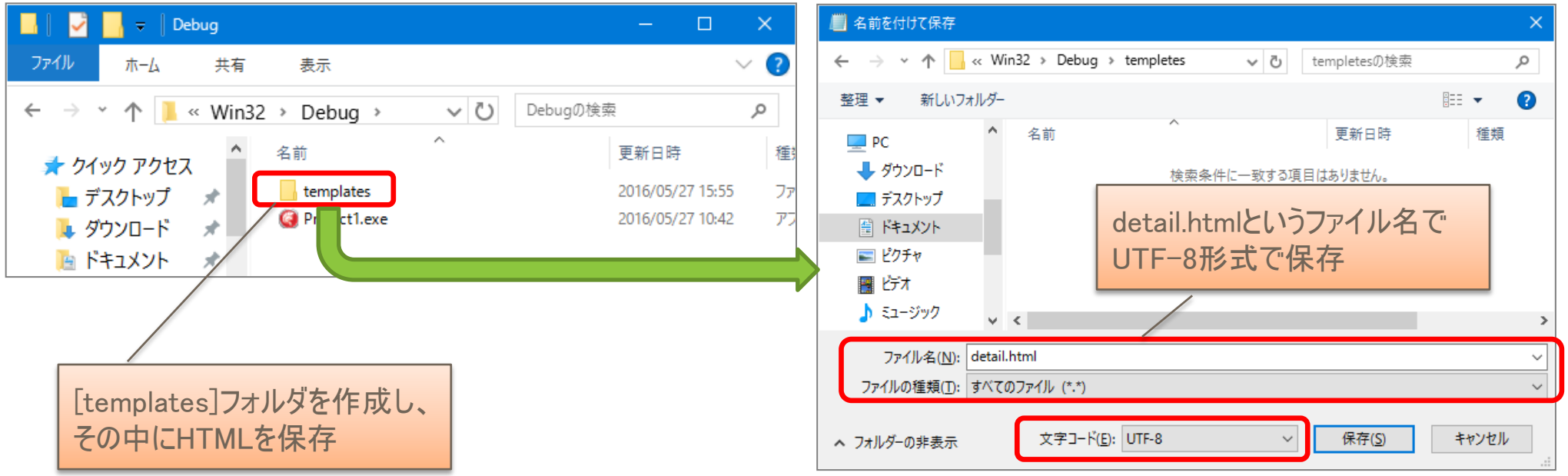
フッタ領域



embarcadero DEVELOPER CAMP

明細画面の作成

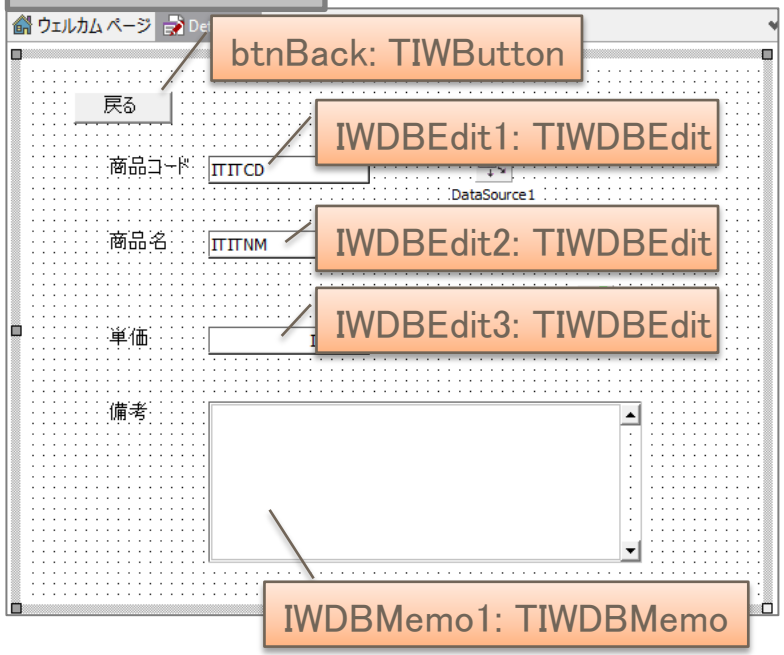
- テンプレート用フォルダにHTMLファイルを保存
  - メモ帳にて[名前を付けて保存]を選択
  - IntraWebアプリ実行フォルダの中に[templates]フォルダを作成
  - [templates]フォルダの中にhtmlを保存
    （明細画面用として、ここではファイル名をdetail.htmlとして保存）
  - 保存の際、文字コードは”UTF-8”とする



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

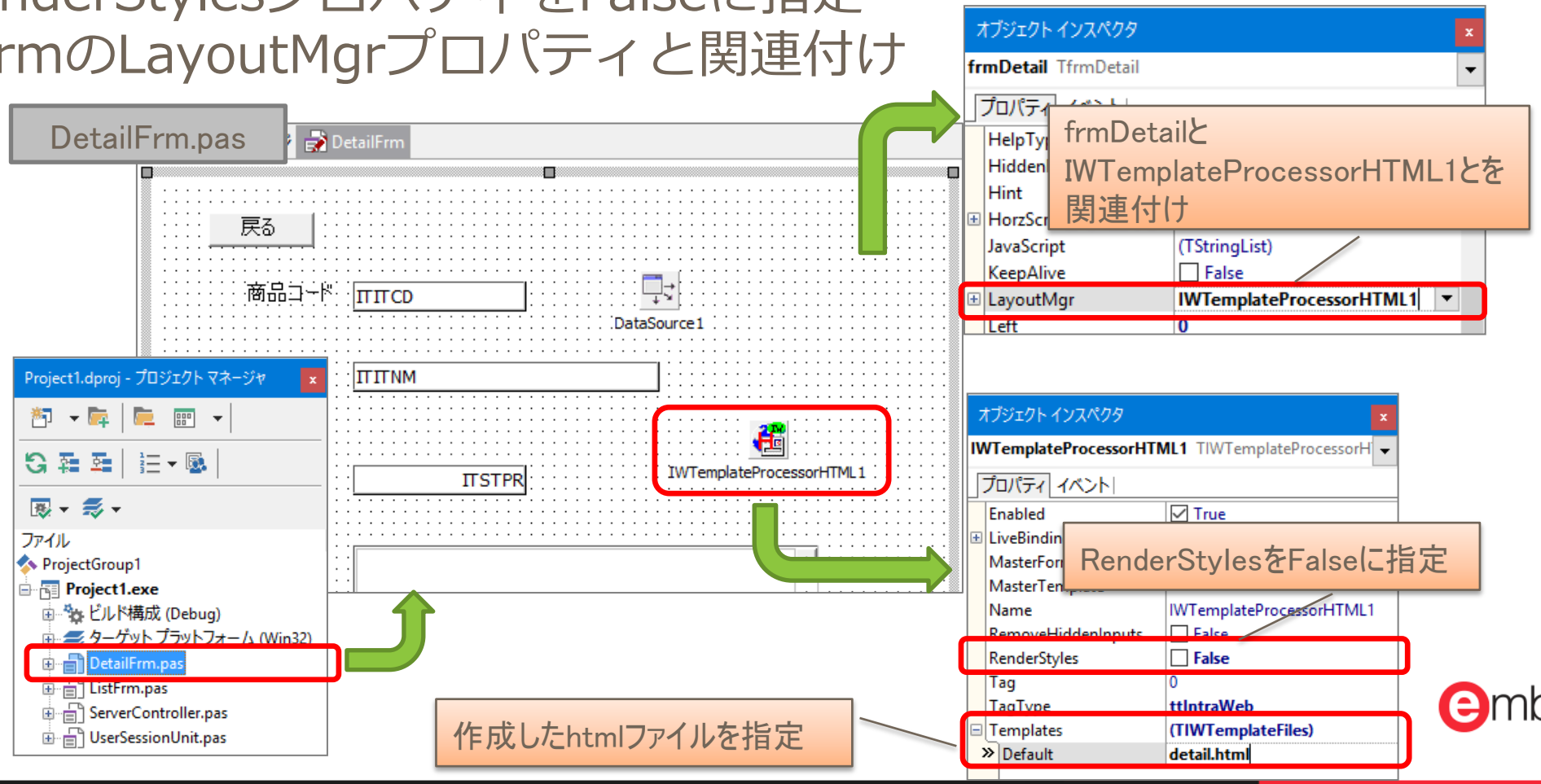
ui-field-contain
ラベルとテキスト部品の
レイアウトを画面サイズに
より調整

IWコンポーネントの関連付け

embarcadero DEVELOPER CAMP

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

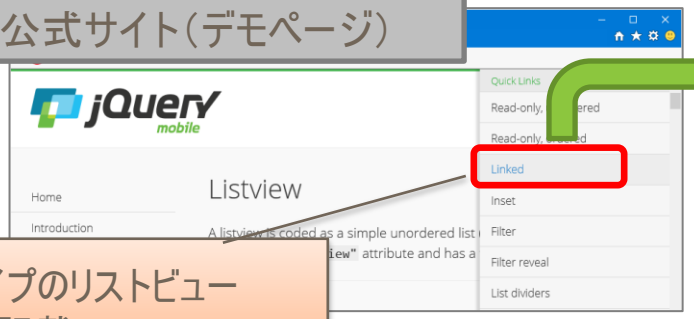
- 一覧画面もテンプレートHTMLで動作するように変更
  - detail.htmlをコピーして、list.htmlを作成
  - ListFrm.pasにIWTemplateProcessorHTMLを追加してHTMLと紐づけを実施
  - list.htmlのコンテンツ領域に {%IWDBGrid1%}を指定
- 実行

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

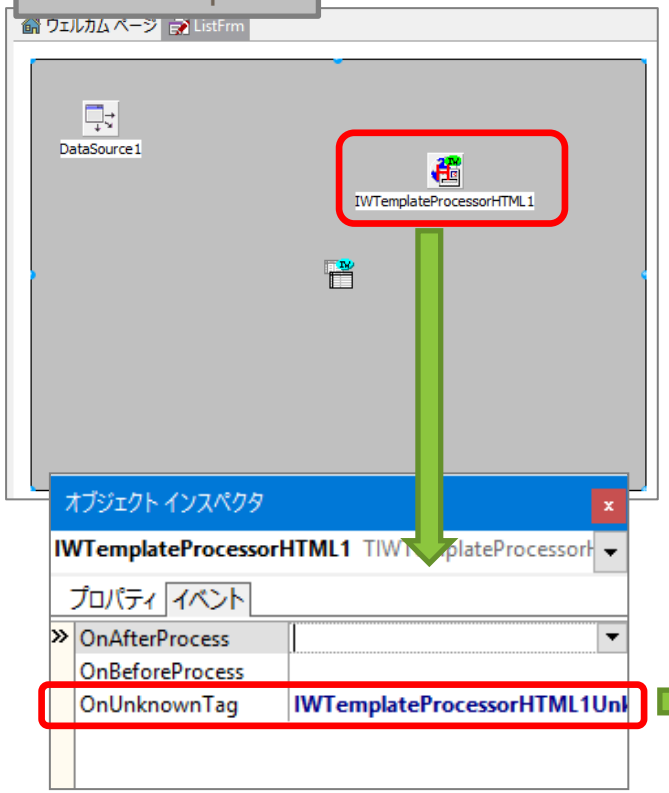
スマートフォンのタッチ操作に
最適なリスト形式で出力

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

embarcadero DEVELOPER CAMP

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

embarcadero DEVELOPER CAMP

一覧画面の作成

- 実行

商品マスタより取得した
商品名をもとに動的に
作成したリストを表示

動的に作成したリストビューには、
クリックイベントが定義されて
いない為、タッチしても画面が
遷移しない

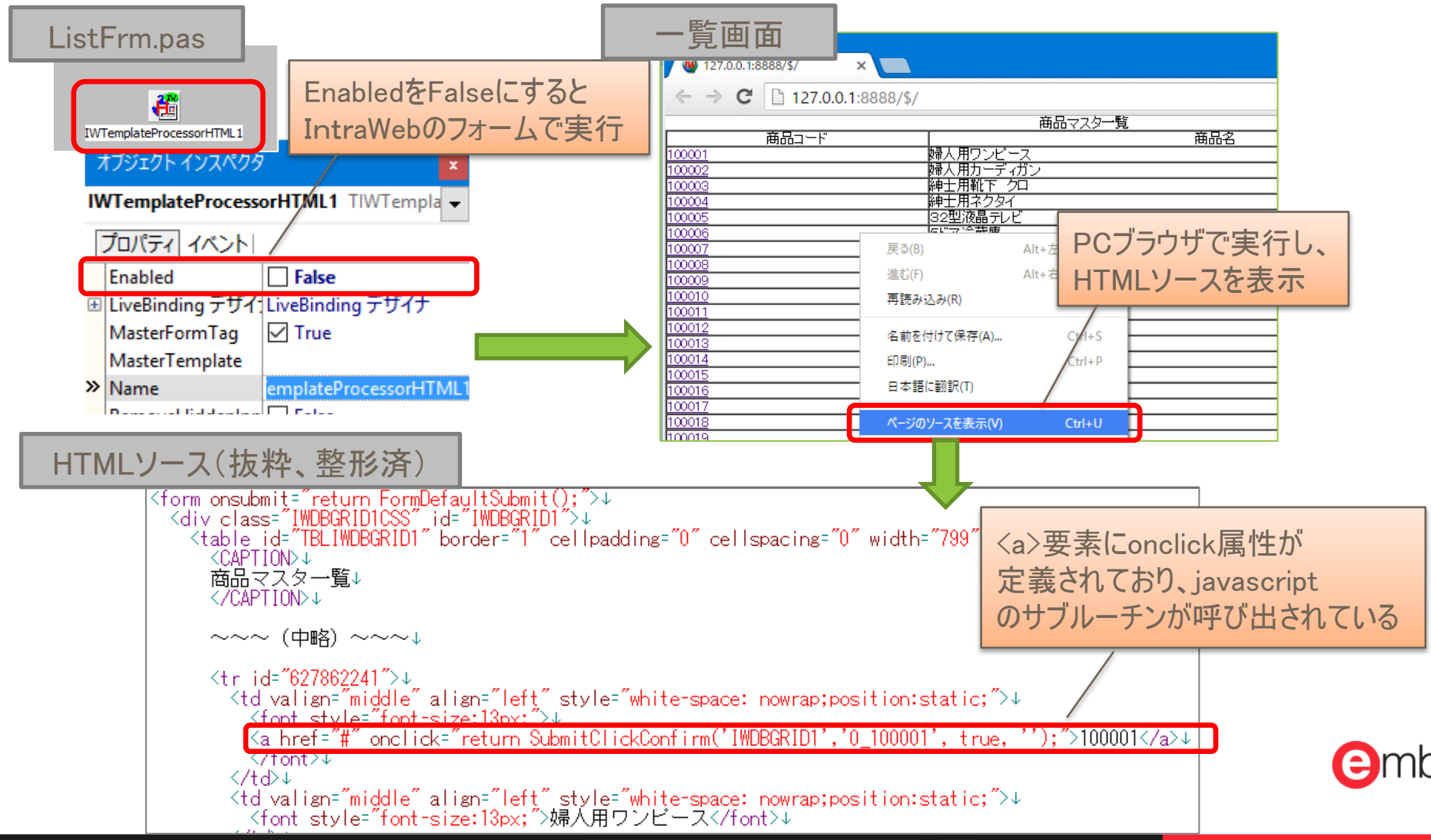
次のような形式でタグを作成
すれば、より使い勝手の良い
リストも作成可能
（例）
<img　src=“画像ファイル” >
商品コード
<h1>商品名</h1>
<h4>単価</h4>

応用例

Delphiのイベントと関連付けることはできないか？

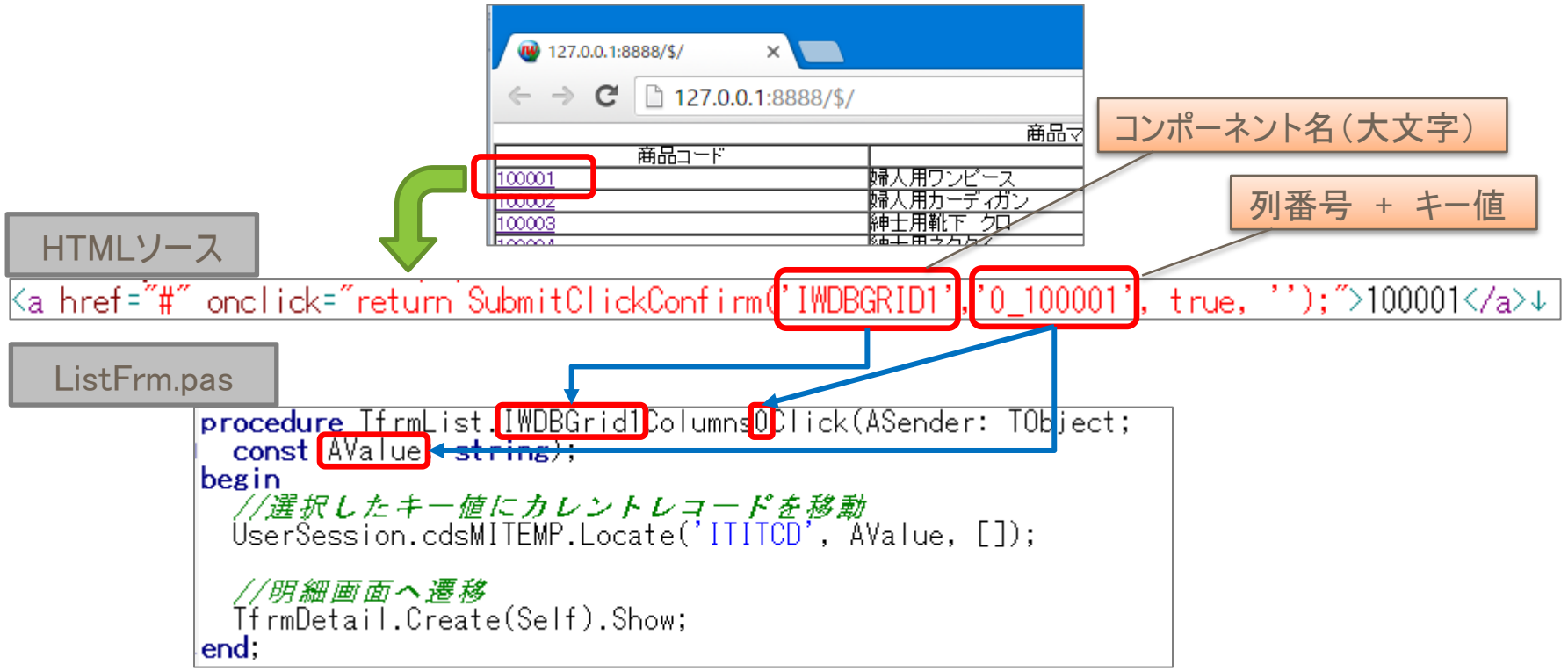
一覧画面の作成

- 再度、IntraWebのフォームで一覧画面を実行



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



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

embarcadero DEVELOPER CAMP

一覧画面の作成

- 実行（完成）

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！

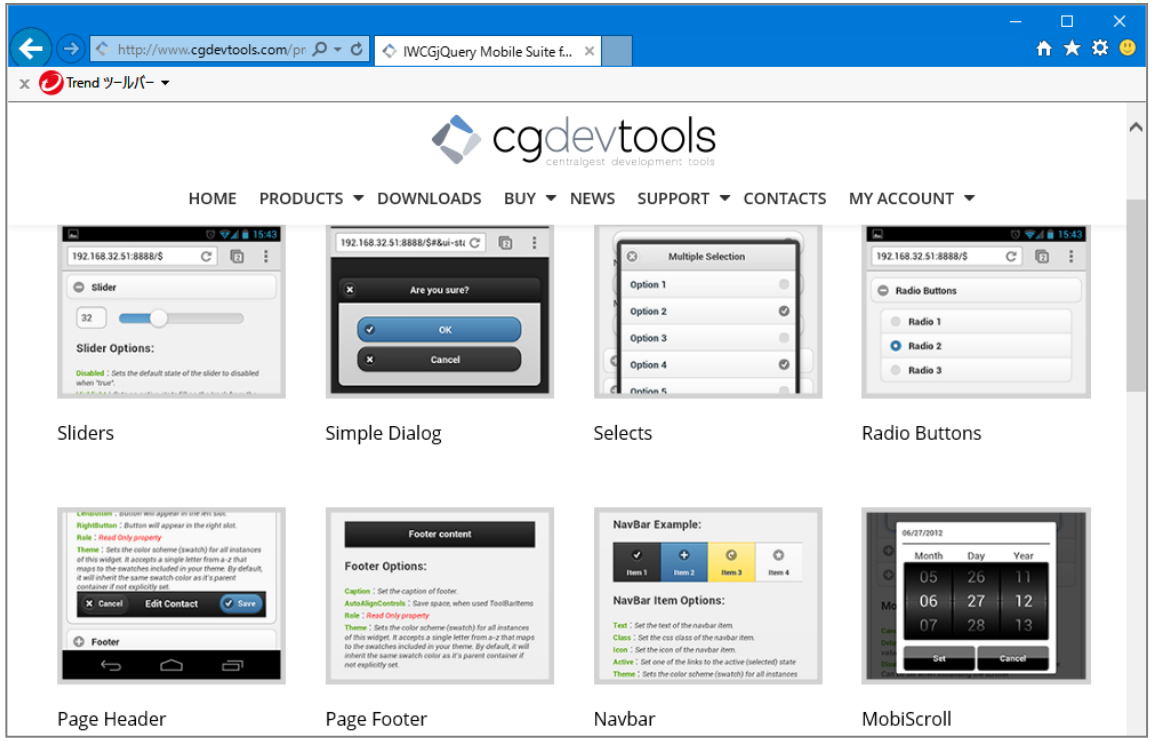
embarcadero DEVELOPER CAMP

（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画面を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/



- ユニットテストフレームワークについて

embarcadero DEVELOPER CAMP

ユニットテストとは？

- ユニットテストとは、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

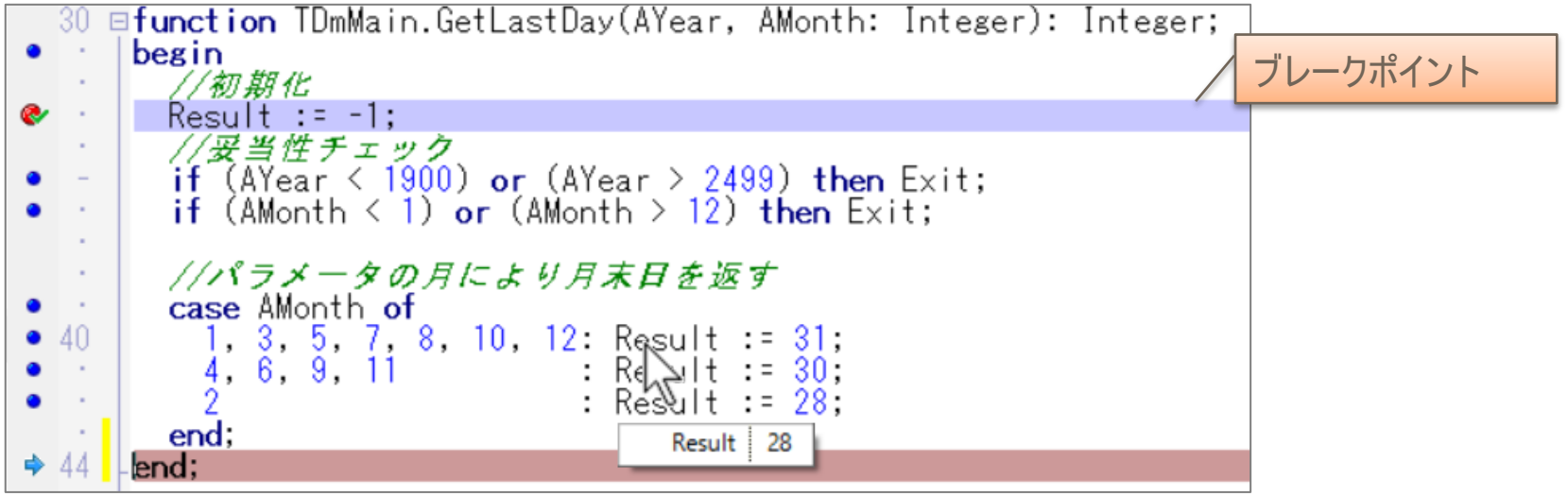
```
type
  TDmMain = class(TDataModule)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    function GetLastDay(AYear, AMonth: Integer): Integer;
  end;
```

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

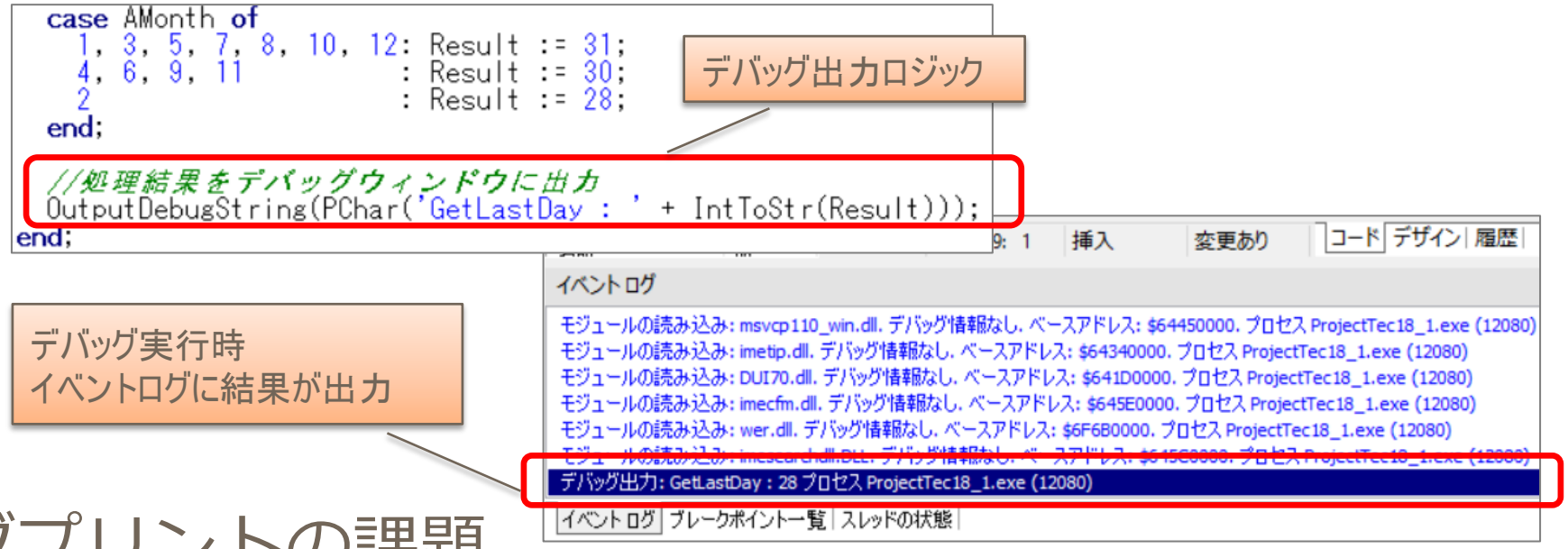
- デバッグ実行
  - ブレークポイントを設定して、デバッグすることで実際に動作させながら値の確認などが行える。



- 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。



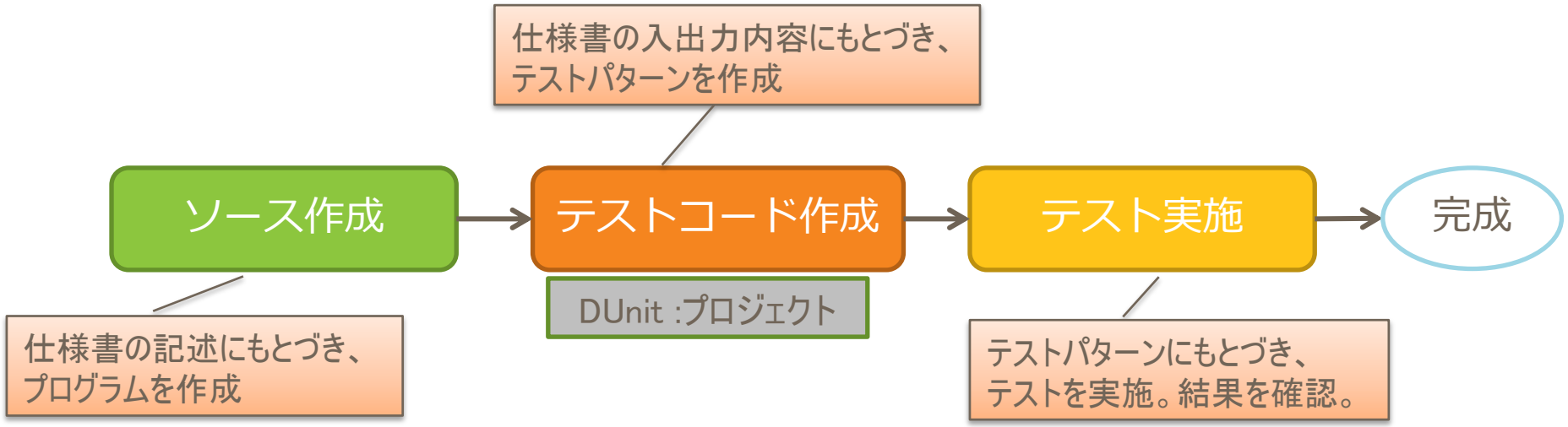
- デバッグプリントの課題
  - 実装コードの中に直接デバッグ用コードを埋め込む必要がある

ユニットテストの課題を解決する方法はないか？

embarcadero DEVELOPER CAMP

テストフレームワーク DUnit

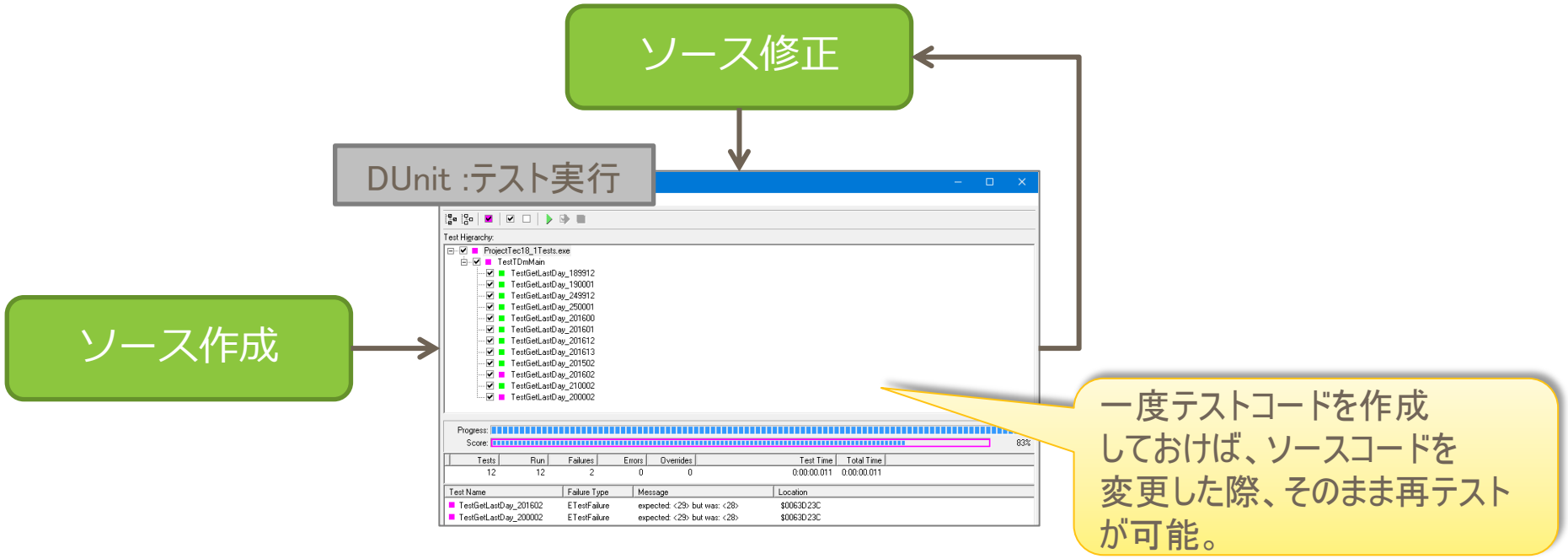
- ユニットテストを行うためのフレームワーク
  - 実装コードとは、別にテスト用コードを用意し、テスト実施を自動化可能
  - テストパターンを用意しておけば、何回でもテストを繰り返すことができる



embarcadero DEVELOPER CAMP

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;

embarcadero DEVELOPER CAMP

サンプルプログラム

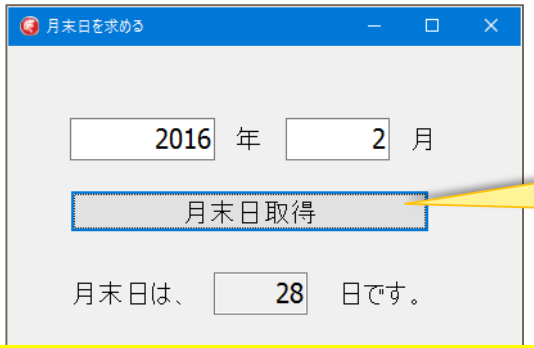
- GetLastDayメソッドのコーディング

```
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



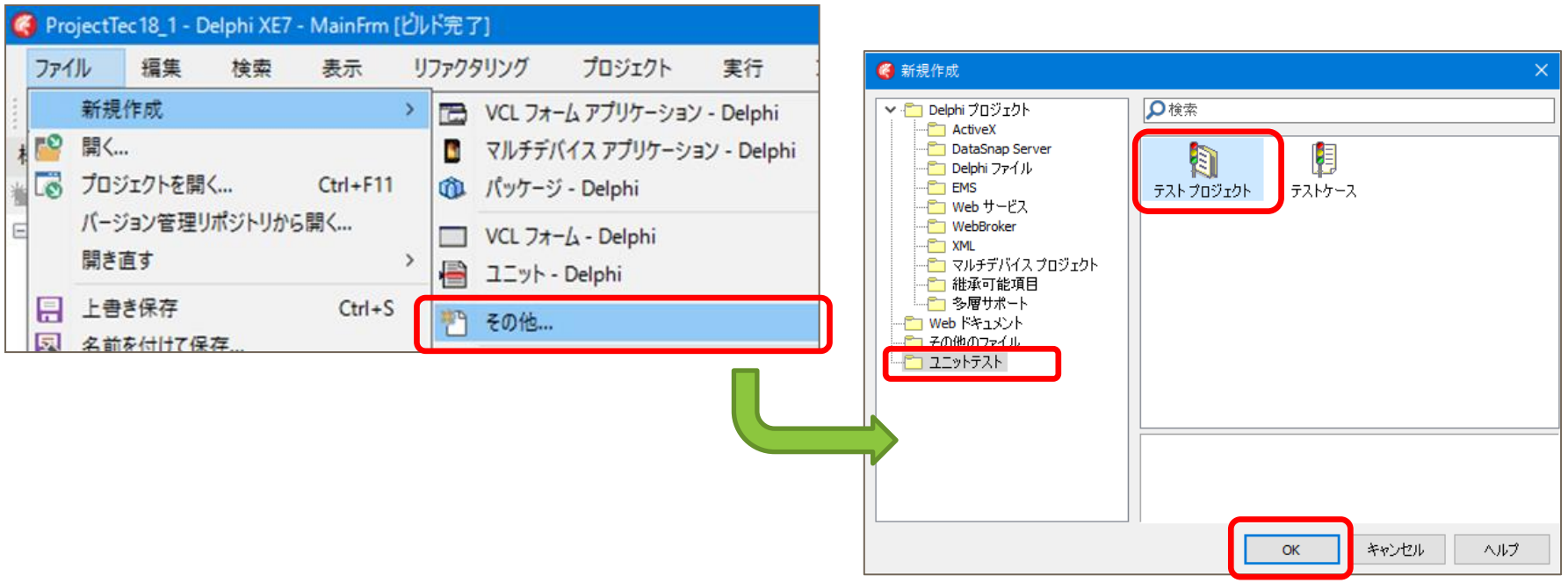
うるう年の考慮がないので、
今年(2016年)は、2月は29日
のはずが、28日と表示。
（メソッドの修正が必要）

GetLastDayメソッドをテストする方法を検討！

embarcadero DEVELOPER CAMP

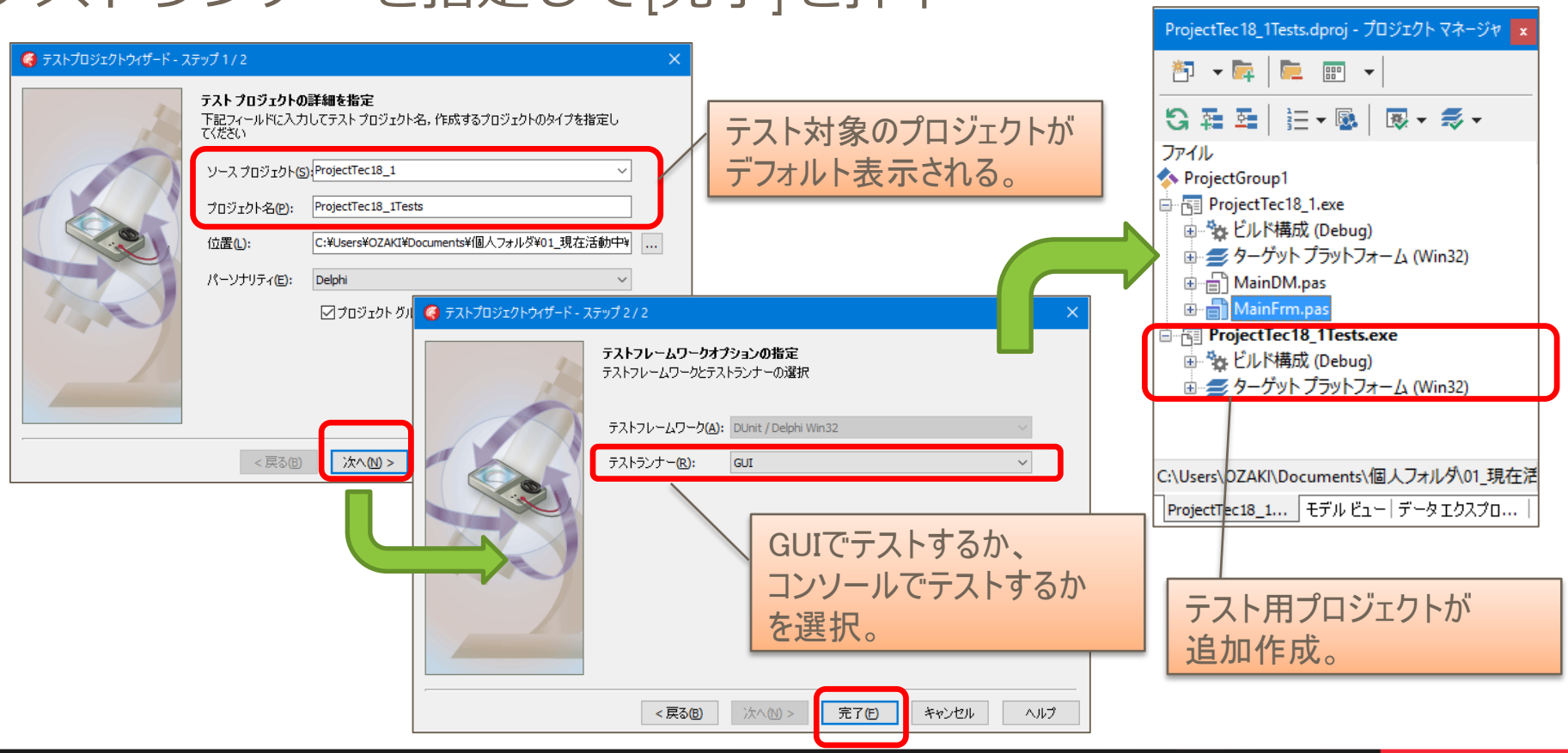
ユニットテストプロジェクトの作成

- 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



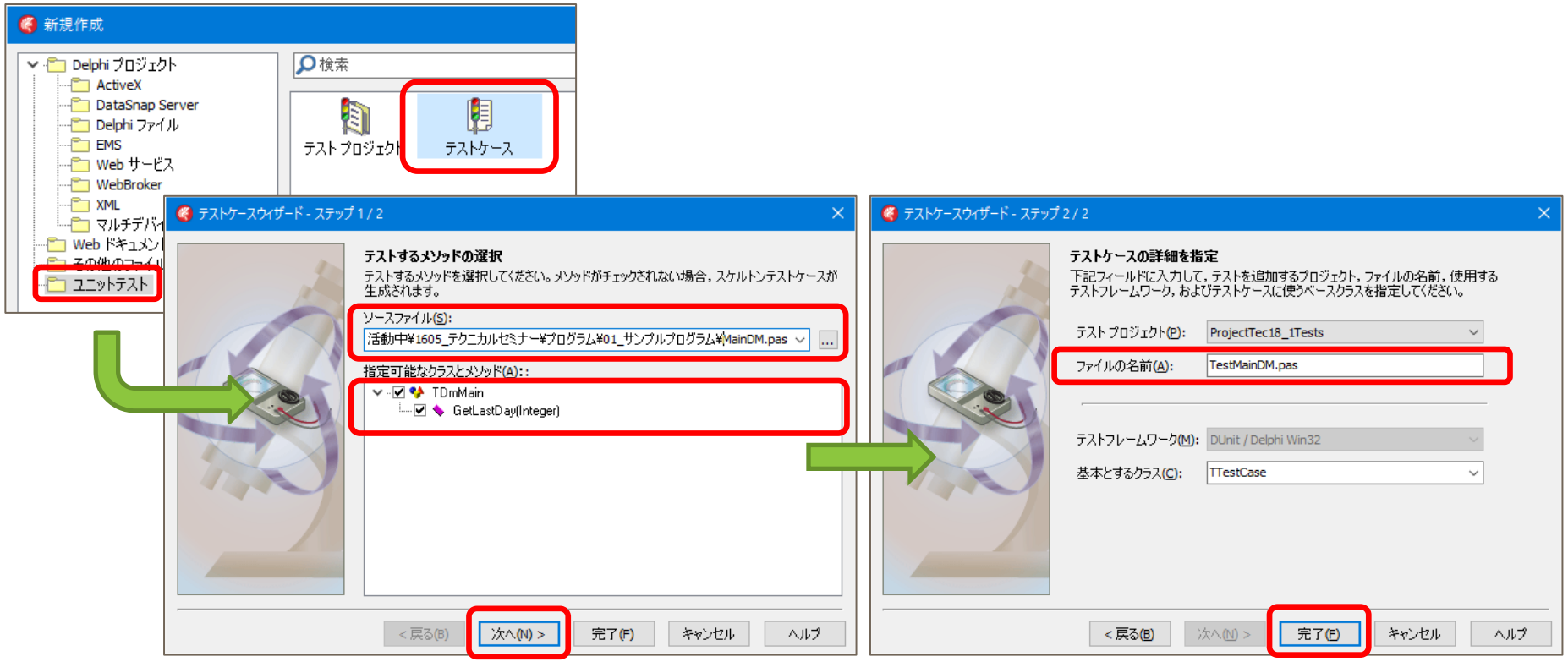
ユニットテストプロジェクトの作成

- 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

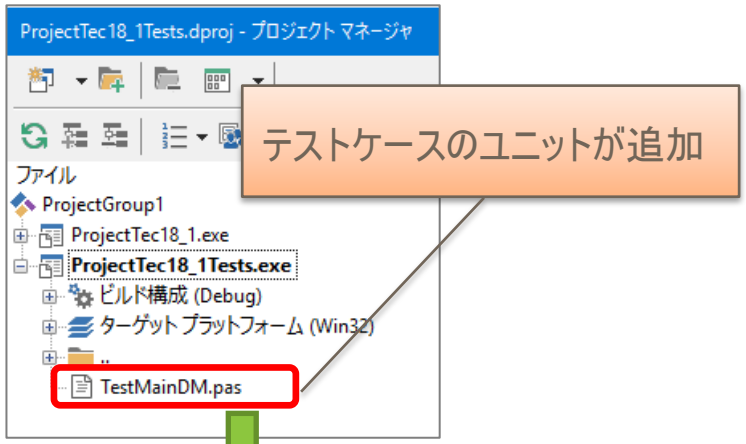


テストケースの作成

- 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



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！

embarcadero DEVELOPER CAMP

テストケースの作成

- 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

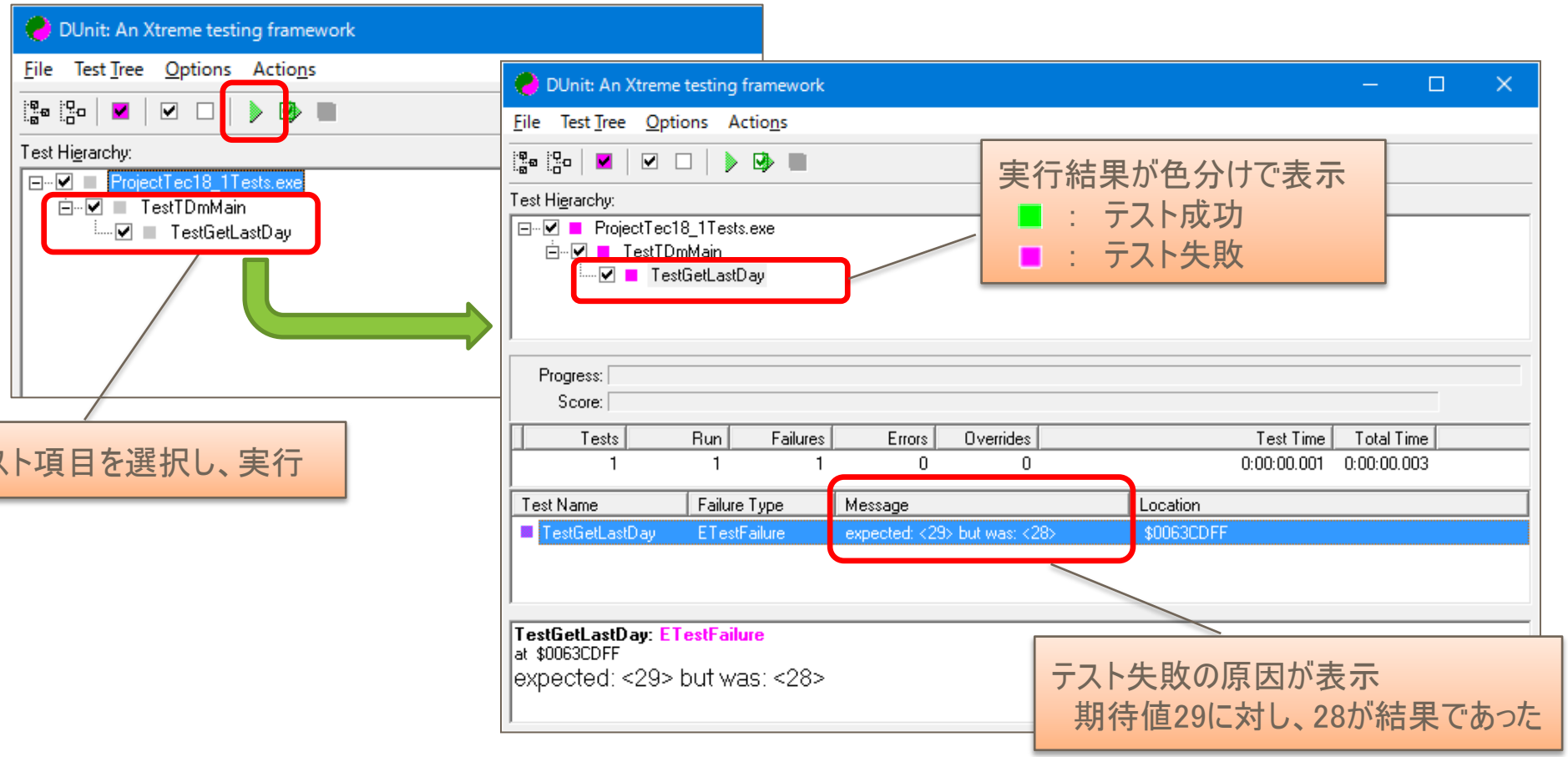
メソッド実行

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

embarcadero DEVELOPER CAMP

テストケースの作成

- テストプロジェクトを[実行]



仕様書に基づき、必要なテストパターンを同様に追加すればよい！

テストパターンの作成

- テストパターンの作成
  - 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

  - テストパターンの検討
    - 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）

テストパターンの作成

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.41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

テストパターンにもとづくメソッドを追加で宣言

実装部（P.42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

embarcadero DEVELOPER CAMP

テストパターンの作成

テストパターン実装部（一部抜粋）

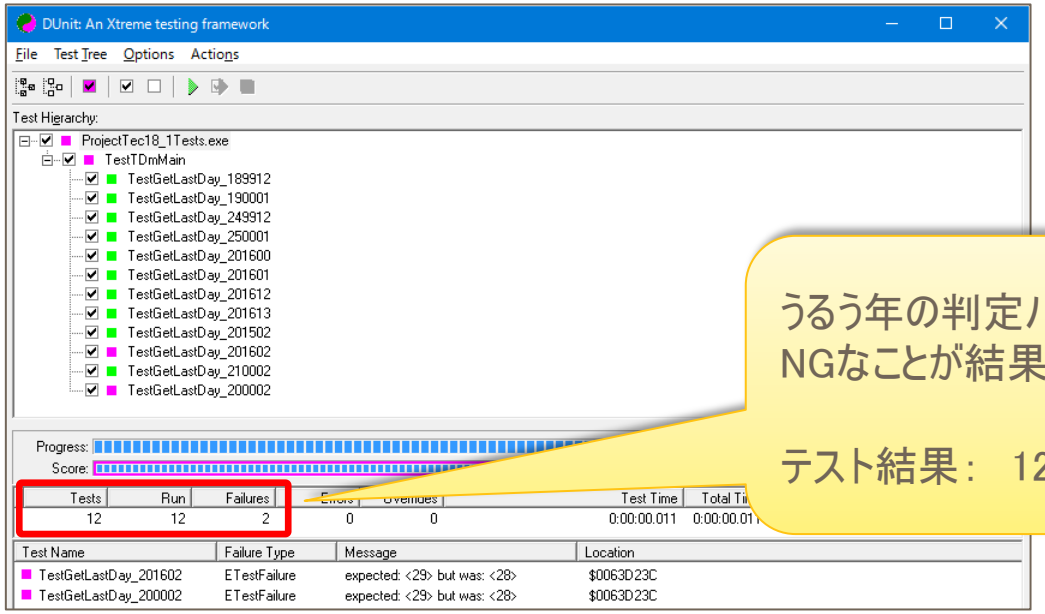
```
procedure TestTDmMain.TestGetLastDay_189912;
begin
  TestGetLastDay(1899, 12, -1);
end;

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;
```

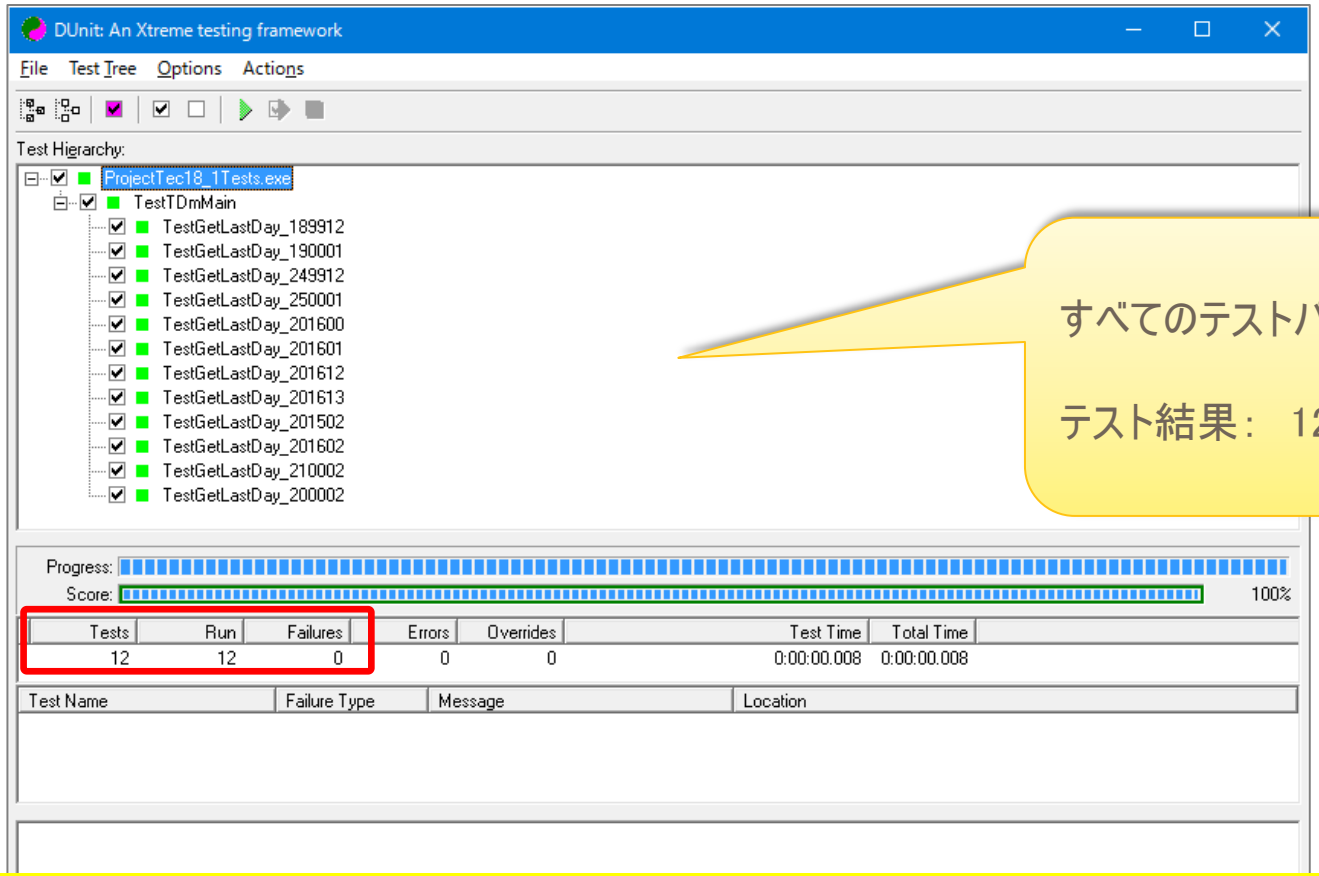
うるう年の判定ロジック

（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



すべてのテストパターンにOK

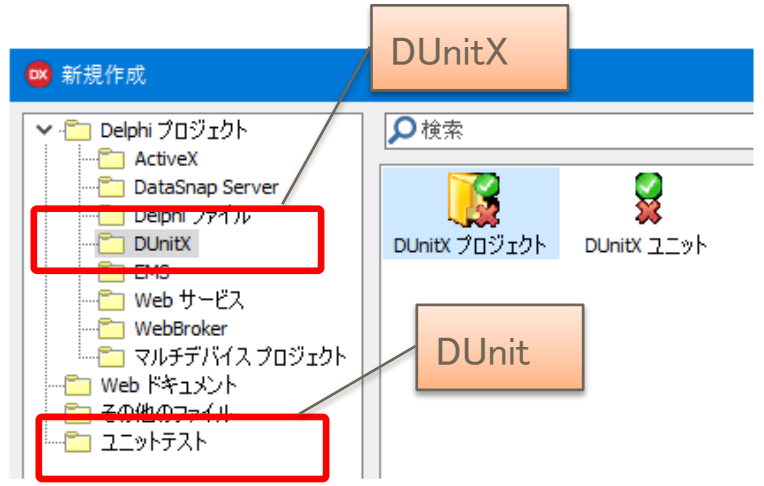
テスト結果： 12件中12件OK

テストパターンがあれば、何度でも再テスト可能でデグレードを防止！

embarcadero DEVELOPER CAMP

（参考）DUnitXについて

- 新しいテストフレームワークDUnitX
  - Delphi XE8以降からは、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

embarcadero DEVELOPER CAMP

- モバイルアプリケーション開発のコツ

embarcadero DEVELOPER CAMP

PCアプリとモバイルアプリの違い

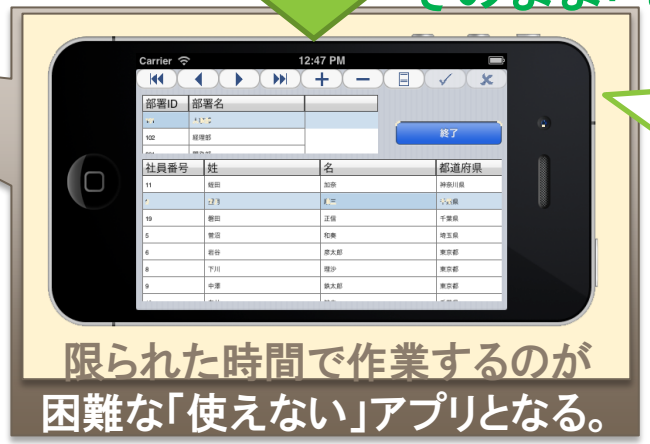
PC

時間をかけて作業することを
前提とした多目的なアプリ。

オフィスでの使用が前提
- キーボード
- マウス
- 十分な画面サイズ
- リッチな通信環境

モバイル

そのまま「モバイル化」しても...



限られた時間で作業するのが
困難な「使えない」アプリとなる。

いつでも・どこでもが前提
- ソフトキーボード
- タッチ
- 限られた画面サイズ
- 不安定な通信環境

PCアプリの全ての機能をそのままモバイルで実現しても、結局使い勝手が悪い！

embarcadero DEVELOPER CAMP

モバイルに最適なアプリとは？

- 限られた画面サイズ/入力方法
  - 1画面に情報を詰め込みすぎない（1機能＝1画面）
    - 部門選択　→　社員選択　→　社員編集
  - 必要最小限の入力にする工夫
    - モバイルから編集が必要な物を取捨選択
- デバイス機能の活用
  - カメラによる画像撮影
  - マップやGPS、音声による入力省力化
  - 電話やメールとの連携

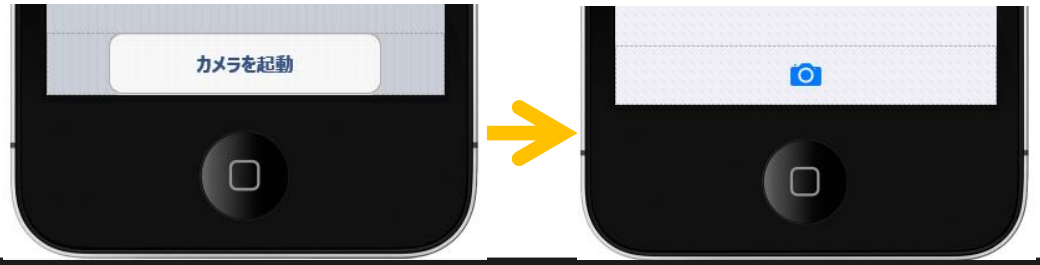
モバイルアプリでは、
何を**選び**、何を**捨てる**か、
何を**新しい価値**として採用するか？を
明確にすることが重要！

embarcadero DEVELOPER CAMP

モバイル画面設計のポイント

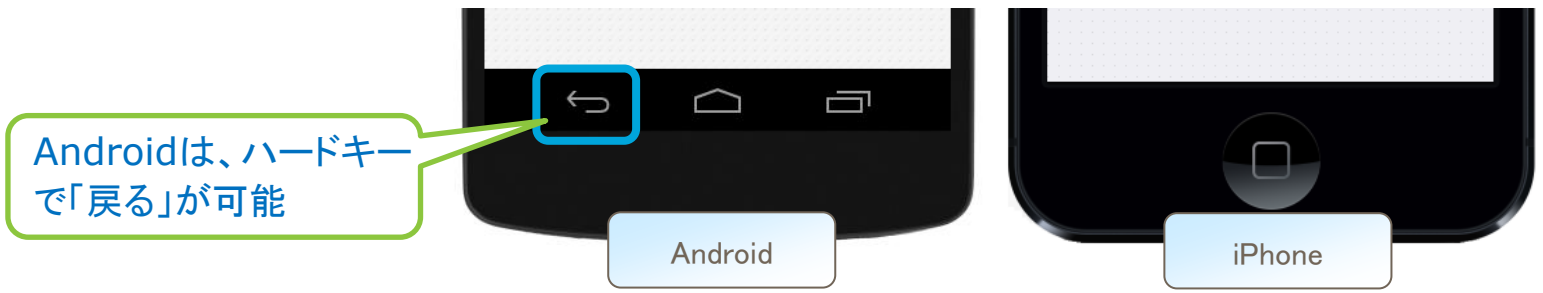
- 使用するデバイスや画面サイズが限定できるか？
  1. 業務で使用するデバイスが限定できれば画面設計はシンプル。
     例）iPad専用アプリであれば 論理サイズ 768dp × 1024dpで固定できる。
     iPad/iPad2　：　1024ﾋﾟｸｾﾙ×768ﾋﾟｸｾﾙ
     iPad3以降　：　2048ﾋﾟｸｾﾙ×1536ﾋﾟｸｾﾙ
  2. 機種が限定できない場合でも、論理サイズで設計する事で
     ある程度の解像度の違いは吸収可能。
  3. 実行時の縦横切り替えを許可するか、固定化するかを決定する。

- シンプルな表現を意識する。
  1. テキスト/ボタンのサイズは大きくする。
  2. 表示する情報量は最低限にする。
     例）メール一覧：受信日時　　【当日】時分のみ　【当年】月日のみ
  3. 表記をシンプルにする。
     例）アイコン表示

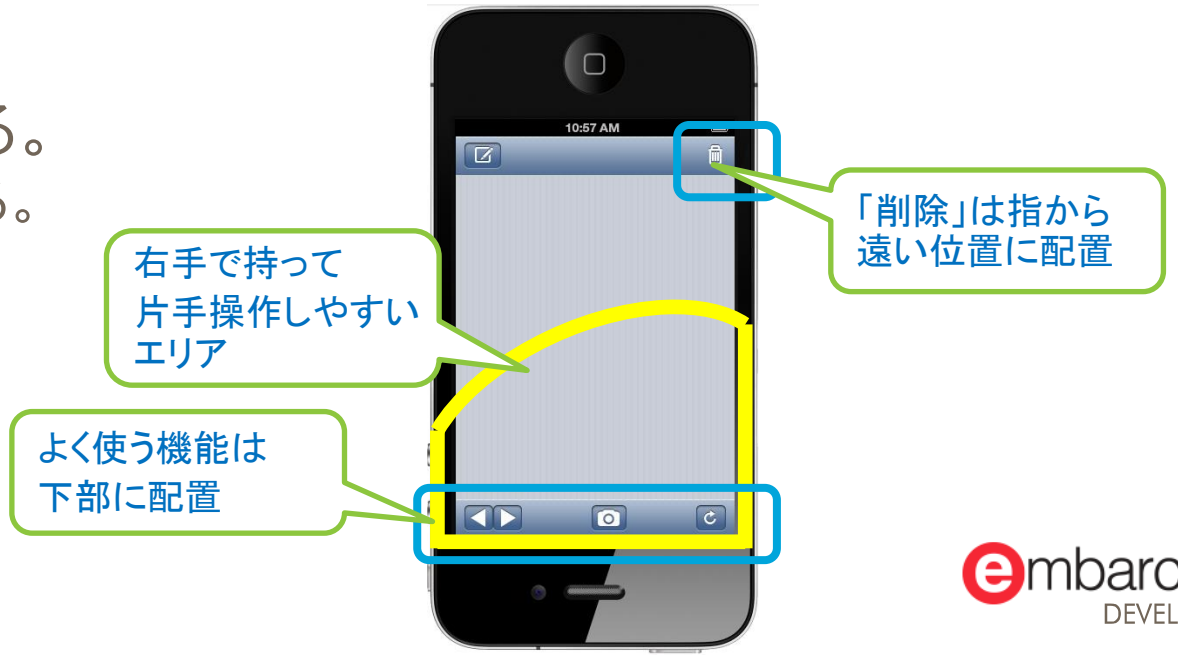


モバイル画面設計のポイント

- デバイスの違いを意識する
  1. iOSには「戻る」ボタンが無い。iOSの場合には画面上に「戻る」ボタンを配置する。



- コンポーネントの配置に留意する。
  1. よく使用する機能は、下に配置する。
  2. 逆に「削除」等、誤タップを防ぎたいものは上は配置。
  3. 片手で操作しやすいエリアを意識する。
  4. ソフトキーボードの出現を考慮する。



embarcadero DEVELOPER CAMP

モバイルアプリ開発のコツ

- モバイル開発では、デバイス機能の活用が重要。

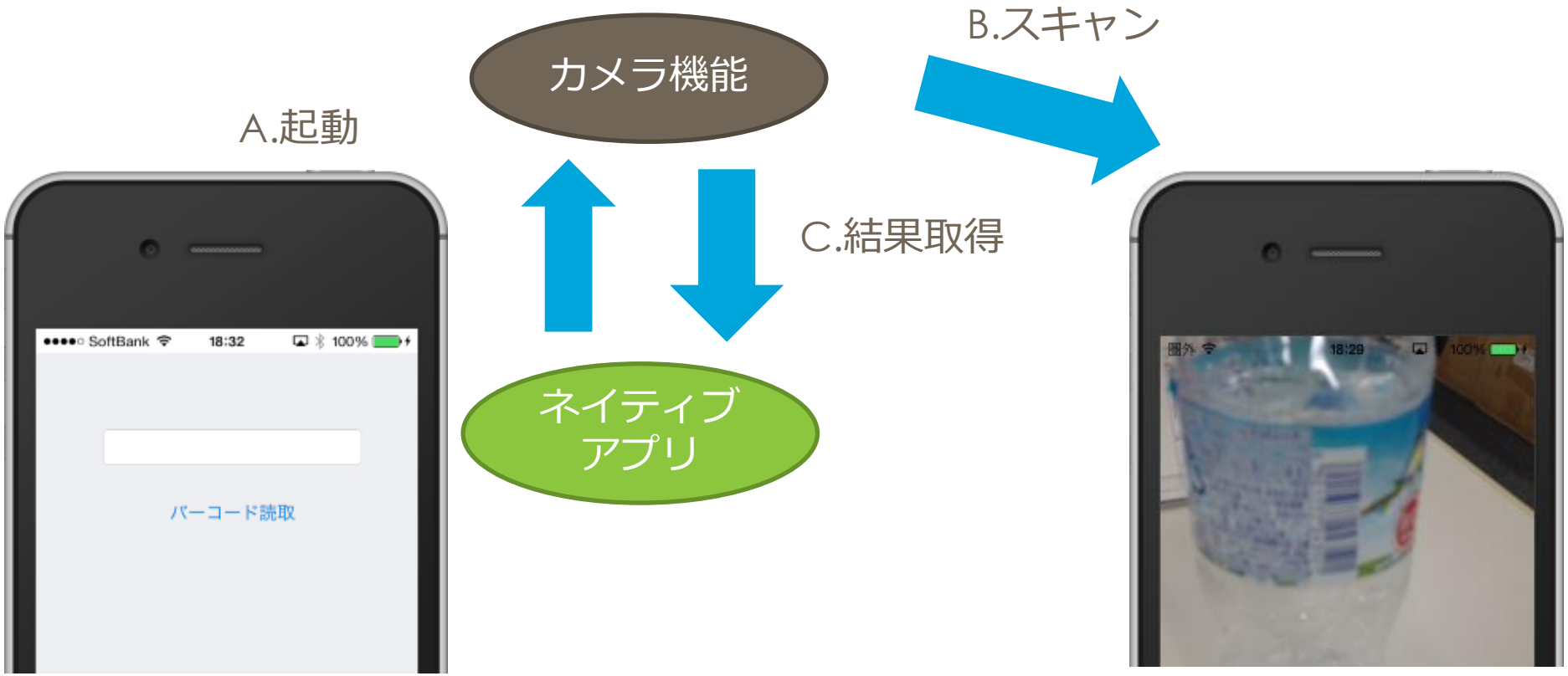
今回は、デバイス機能を活用した下記の活用方法を紹介！

①　バーコード読み込み　（カメラ機能）
②　Beacon　（IoT:Bluetooth機能）

embarcadero DEVELOPER CAMP

(1)カメラを使ったバーコード読み取り機能

- カメラ機能を使ったバーコード読み取りの仕組み
  - スマートデバイスではカメラ機能を利用して、バーコードやQRコードを読み取り、値を取得することが可能。
    （PCのように、バーコードリーダーの外部接続は不要）



(1)カメラを使ったバーコード読み取り機能

- 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/blog.asp?post=280

tmssoftware.com

ただしZBarSDKコンポーネントはiOS専用。
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コンポーネントが便利。

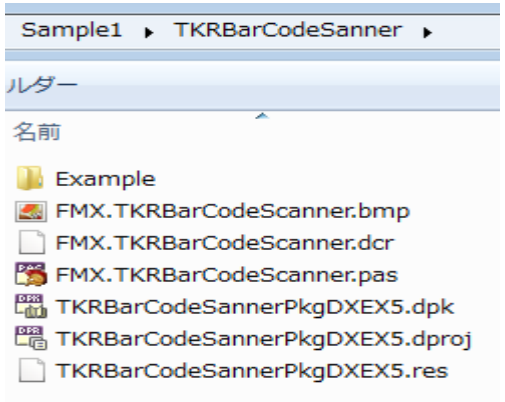
【TKRBarCodeSanner】 ※iOS / Android可能
（iOS使用時はZBarSDKもインストールが必要 ）
http://www.fmxexpress.com/qr-code-scanner-source-code-for-delphi-xe5-firemonkey-on-android-and-ios/

→ 今回はこのTKRBarCodeSannerコンポーネントを使用

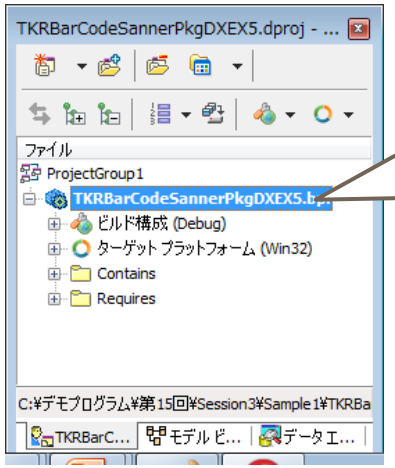
embarcadero DEVELOPER CAMP

(1)カメラを使ったバーコード読み取り機能

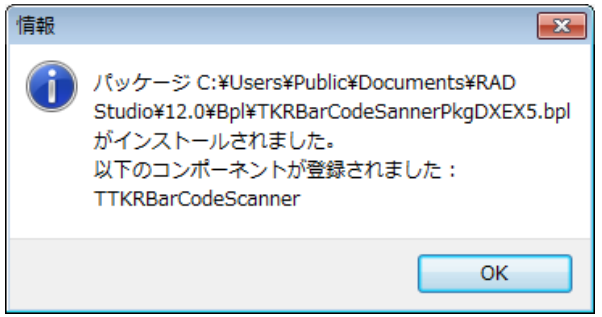
- TKRBarCodeSannerコンポーネントのインストール①
  - TKRBarCodeSanner.zipをダウンロードして展開



  - [ファイル|プロジェクトを開く]よりTKRBarCodeSannerPkgDXEX5.dpkを開きます。

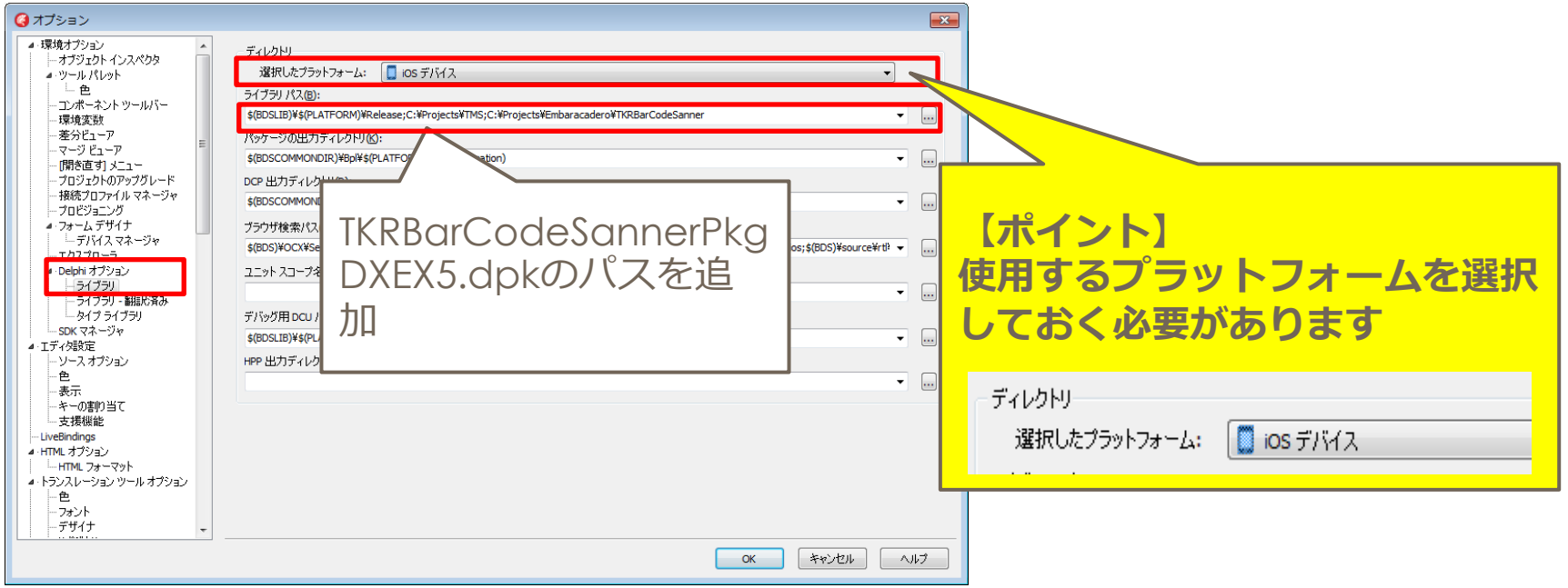


プロジェクトマネージャで右クリックからコンパイルとインストールを実行

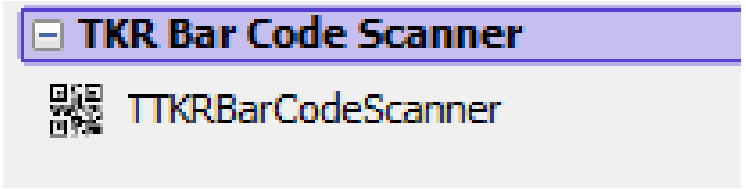


(1)カメラを使ったバーコード読み取り機能

- TKRBarCodeSannerコンポーネントのインストール②
  - [ツール | オプション]のライブラリでライブラリパスにTKRBarCodeSannerPkgDXEX5.dpkを開いたパスを追加



- コンポーネントの登録が完了！



embarcadero DEVELOPER CAMP

(1)カメラを使ったバーコード読み取り機能

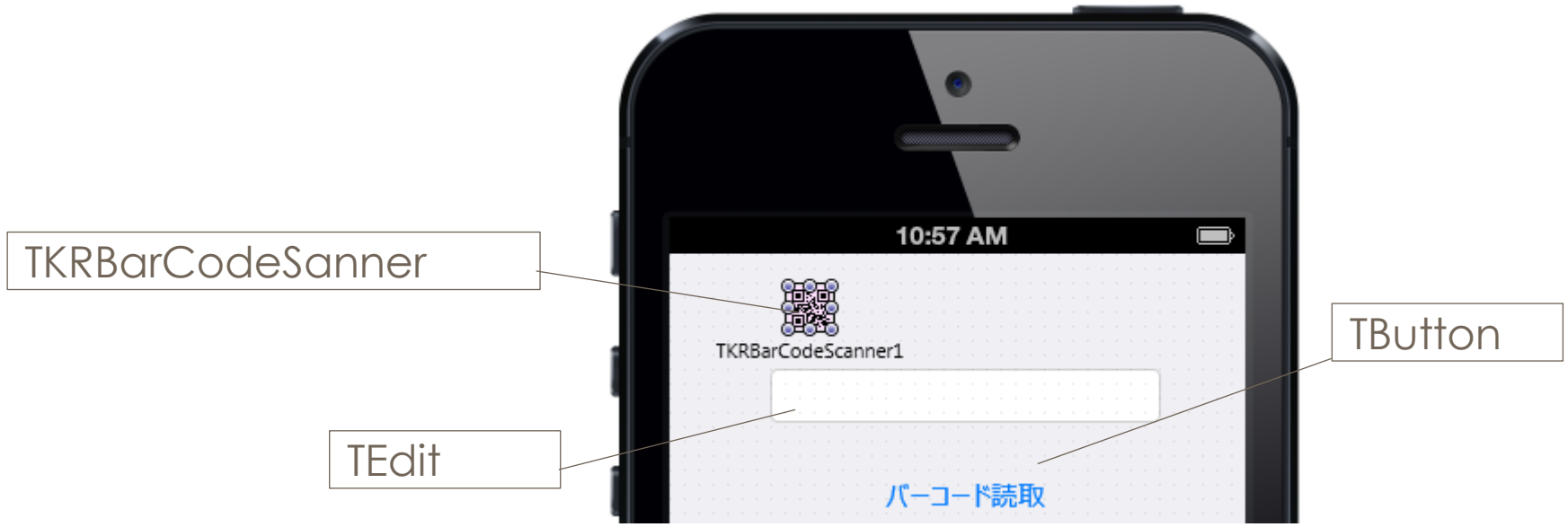
- TKRBarCodeSannerコンポーネントソースの修正が必要
  - Delphi XE5をベースにしたコンポーネントの為、XE7以降は若干修正が必要。

```
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

  Result := False;
  if FMonitorClipboard and (AAppEvent = aeBecameActive) then
  begin
    Result := GetBarcodeValue;
  end;
end;
{$ENDIF}
```

(1)カメラを使ったバーコード読み取り機能

- バーコード機能の実装手順①
  - フォームに次のコンポーネントを配置
    TKRBarCodeSanner 、TEdit、TButton



(1)カメラを使ったバーコード読み取り機能

- バーコード機能の実装手順②
  - TButtonのクリックイベントに処理を記述

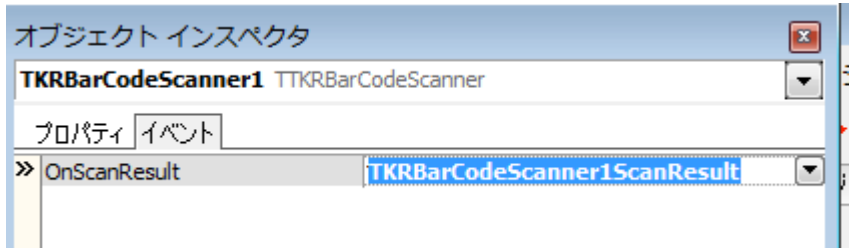
A.カメラ起動 B.スキャン

OnClick処理（バーコードスキャン）

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TKRBarCodeScanner1.Scan;    //バーコードスキャンを実行
end;
```

(1)カメラを使ったバーコード読み取り機能

- バーコード機能の実装手順③
  - TKRBarCodeSannerのスキャン結果イベントにプログラムを実装



C.結果取得

OnScanResult処理（スキャン結果）

```
procedure TForm1.TKRBarCodeScanner1ScanResult(Sender: TObject; AResult: string);
begin
  Edit1.Text := AResult;      //読み取ったコードをEditにセット
end;
```

(1)カメラを使ったバーコード読み取り機能

- バーコード機能の実行

A.カメラ起動　　B.スキャン　　C.結果取得

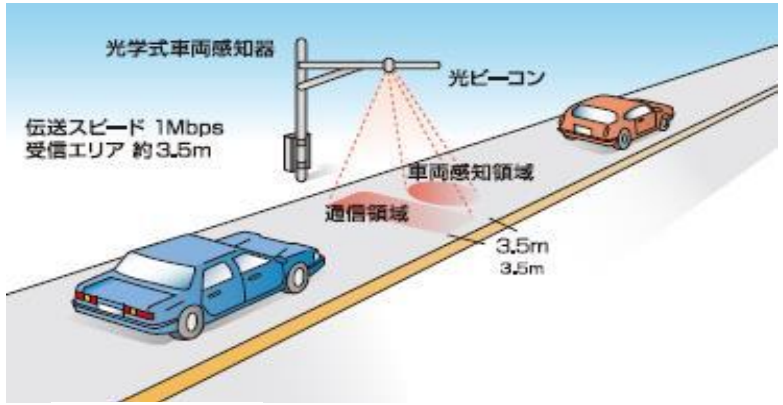
embarcadero DEVELOPER CAMP

(2)Beacon機能の活用

- Beacon（ビーコン）とは？
  - IoTが身近なところで代表される機能のひとつがBeacon（ビーコン）。
    BeaconはBluetoothを使い、半径十数メートル信号を発信する。
    用途としては、スマートフォンなどの機器が信号範囲に
    入ってきたら情報を通知したり（お店のクーポン配信等）、
    信号範囲から位置を特定して位置情報を使った機能として使うこともできる。

【利用例】
道路交通情報通信システム（VICS）では、
渋滞の状況や所要時間、通行止めなどの
情報を電波ビーコンや光ビーコンで
車に情報を伝えている。
（一般財団法人 道路交通情報通信システムセンター）

また登山で携帯する「**雪崩ビーコン**」も
その名の通り、非常時に場所を発信する
Beacon機器。



embarcadero DEVELOPER CAMP

(2)Beacon機能の活用

- TBluetooth、TBluetoothLEコンポーネント
  - 画面上にドラッグ＆ドロップするだけでアプリケーションを Bluetooth、 BluetoothLE(LowEnergy)に対応できる

- TBeaconコンポーネント
  - Beacon情報を感知する

MonitorizedRegionsプロパティ内のアイテムでUUIDを指定

MonitorizedRegions	(TBeaconRegionCollection)
UUID	{00000000-0000-0000-0000-000000000000}

- TBeaconDeviceコンポーネント
  - Beacon情報を発信する

UUIDプロパティを指定

UUID	{00000000-0000-0000-0000-000000000000}

※UUIDとは、全世界で2つ以上のアイテムが同じ値を持つことがない一意な識別子
BeaconはUUID、MajorID、MinorIDなどで信号を判別する

embarcadero DEVELOPER CAMP

(2)Beacon機能の活用

- TBluetooth、TBluetoothLEコンポーネント
  - 画面上にドラッグ＆ドロップするだけでアプリケーションを Bluetooth、 BluetoothLE(LowEnergy)に対応できる

- TBeaconコンポーネント
  - Beacon情報を感知する

MonitorizedRegionsプロパティ内のアイテムでUUIDを指定

MonitorizedRegions	(TBeaconRegionCollection)
UUID	{00000000-0000-0000-0000-000000000000}

- TBeaconDeviceコンポーネント
  - Beacon情報を発信する

UUIDプロパティを指定

UUID	{00000000-0000-0000-0000-000000000000}

※UUIDとは、全世界で2つ以上のアイテムが同じ値を持つことがない一意な識別子
BeaconはUUID、MajorID、MinorIDなどで信号を判別する

embarcadero DEVELOPER CAMP

(2)Beacon機能の活用

- Beacon受信サンプルアプリ開発
  - 機能概要

※Beacon機器は業界の標準規格であるiBeacon と AltBeacon に対応

(2)Beacon機能の活用

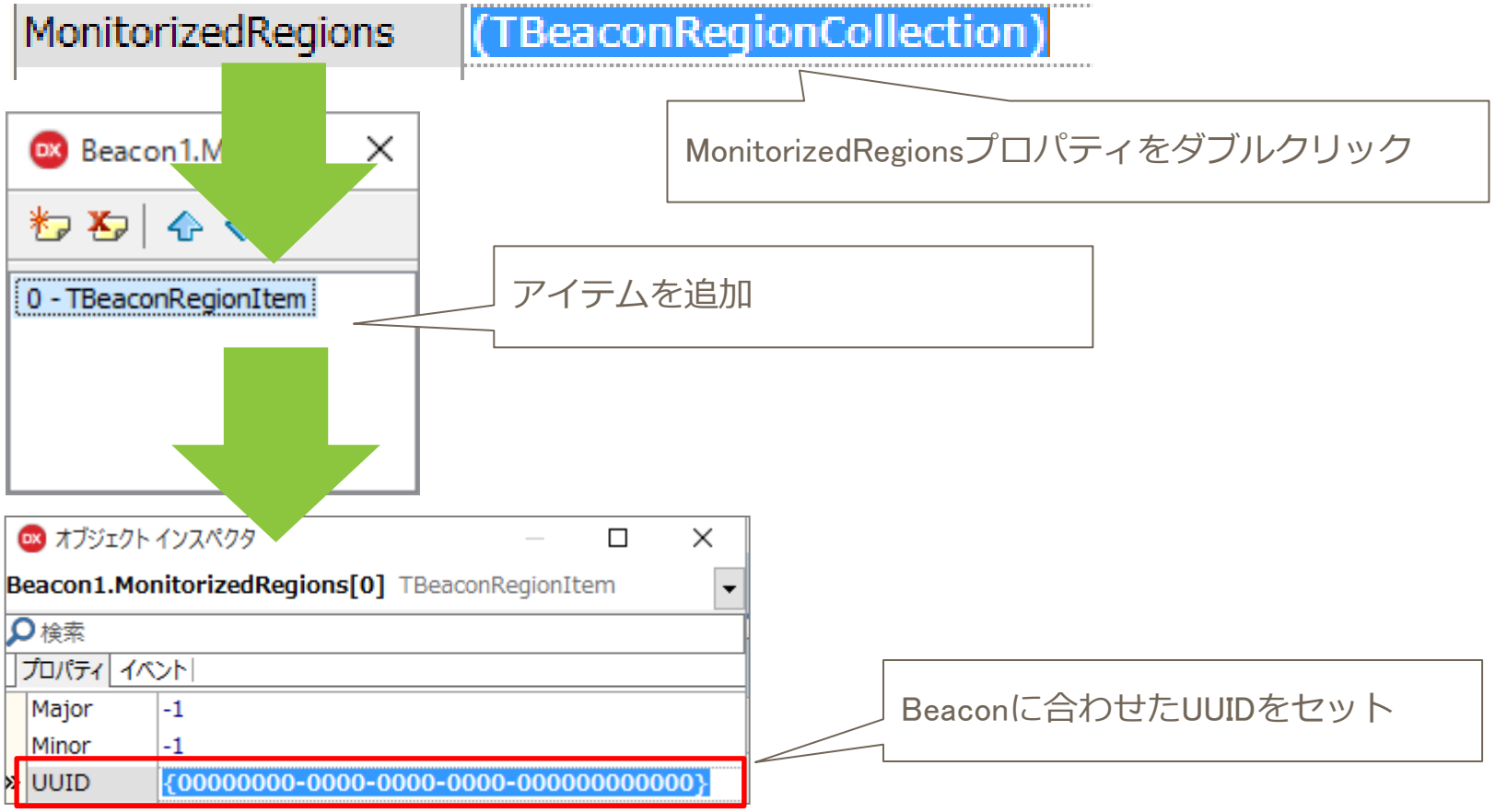
- Beacon受信サンプルアプリ開発手順①
  - コンポーネントの配置

TBeacon
（Beacon処理用）

TImage
（画像をセットしておく）

(3)Beacon機能の活用

- Beacon受信サンプルアプリ開発手順②
  - TBeaconプロパティの設定



(2)Beacon機能の活用

- Beacon受信サンプルアプリ開発手順③

OnCreateイベント（初期処理）

```
procedure TForm1.FormCreate(Sender: TObject);
Begin
  Image1.Visible := False; //画像を非表示
  Beacon1.Enabled := True; //Beaconを有効化
end;
```

OnBeaconEnterイベント（Beaconエリアに入った処理）

```
procedure TForm1.Beacon1BeaconEnter(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  Image1.Visible := True; //画像を表示
end;
```

OnBeaconEnterイベント（Beaconエリアから出た処理）

```
procedure TForm11.Beacon1BeaconExit(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  Image1.Visible := False; //画像を非表示
end;
```

embarcadero DEVELOPER CAMP

THANKS!

www.embarcadero.com/jp

第32回 エンバカデロ・デベロッパーキャンプ