

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



自己紹介



- 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



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



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

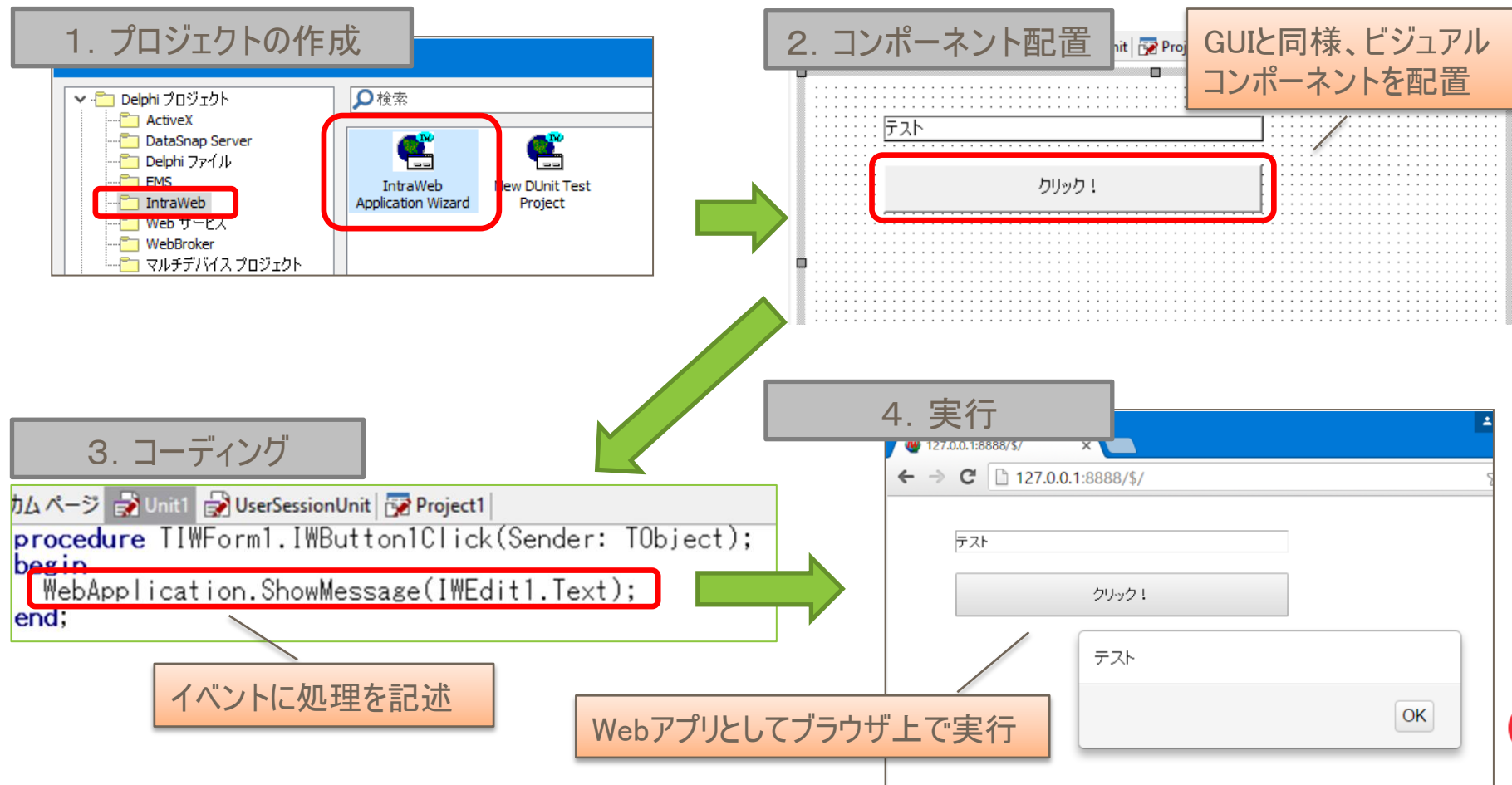
- IntraWebにおけるスマートデバイス向けWebアプリ作成テクニック
- ユニットテストフレームワークについて
- モバイルアプリケーション開発のコツ

■ 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

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



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

The image shows a side-by-side comparison of the IntraWeb development process. On the left, the '開発画面' (Development Screen) displays a visual IDE with a grid-based layout. A form titled '株式会社ミガロ' (Migaro Co., Ltd.) is being designed. The interface includes a '構造' (Structure) pane on the left showing a tree of components like 'FormMain', 'IWFrameLeft1', and 'IWImage1'. A 'プロパティ' (Properties) pane at the bottom left shows settings for 'IWRectangle' such as alignment and color. A green arrow points from this design phase to the right. On the right, the '実行画面' (Execution Screen) shows the rendered web application in a browser. The application features a search bar, a list of recommended products (NOTE PC EX, A, VI), a 'お知らせ' (Notice) section with dates, and a '今週の売上トップ3' (This week's top 3 sales) section. A callout box above the execution screen states 'フォームデザインのレイアウトでアプリが実行' (The application is executed with the layout of the form design).

開発画面

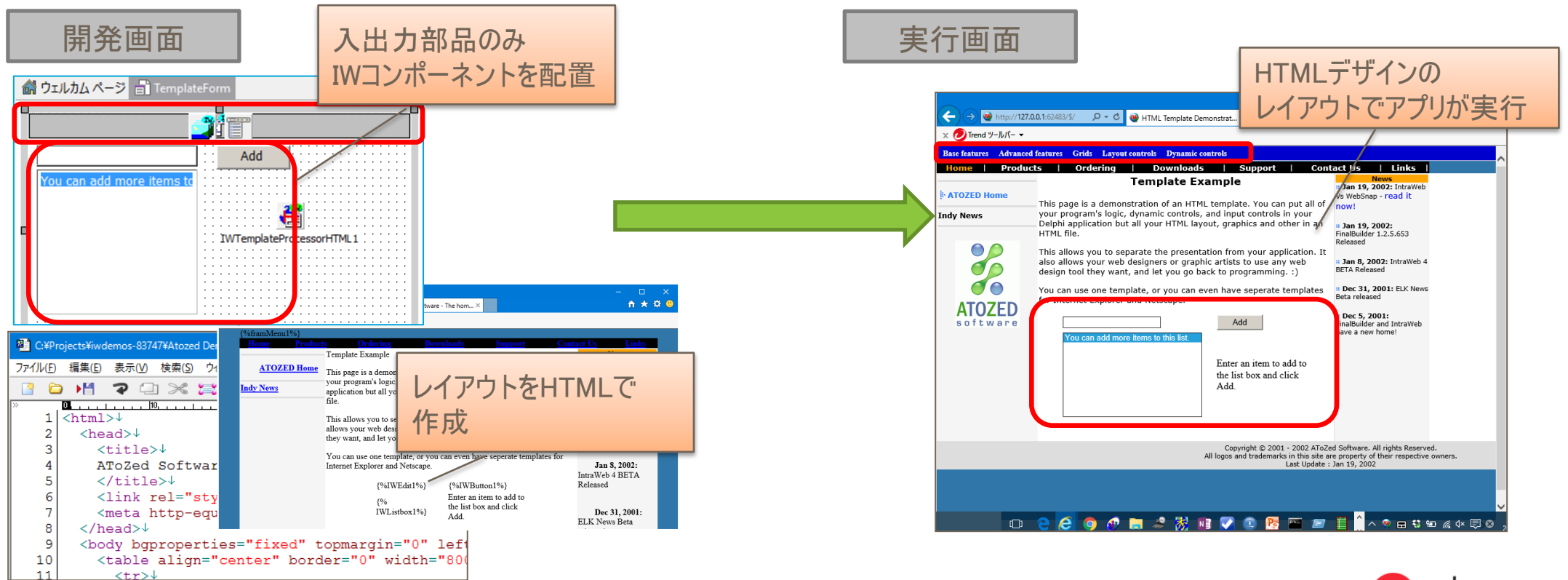
実行画面

フォームデザインのレイアウトでアプリが実行

デザイン、レイアウトをすべてIWコンポーネント群を使用して作成

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）

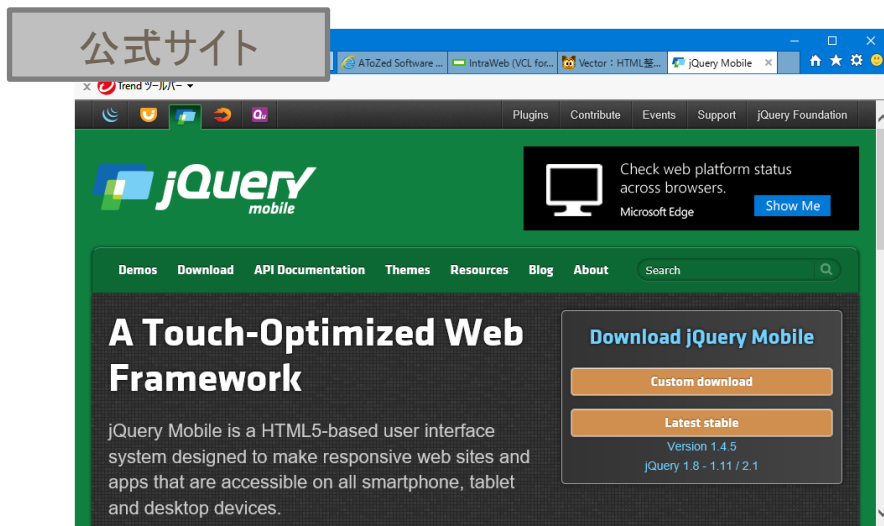
→ スマートデバイスに最適な画面とは？

- 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



<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

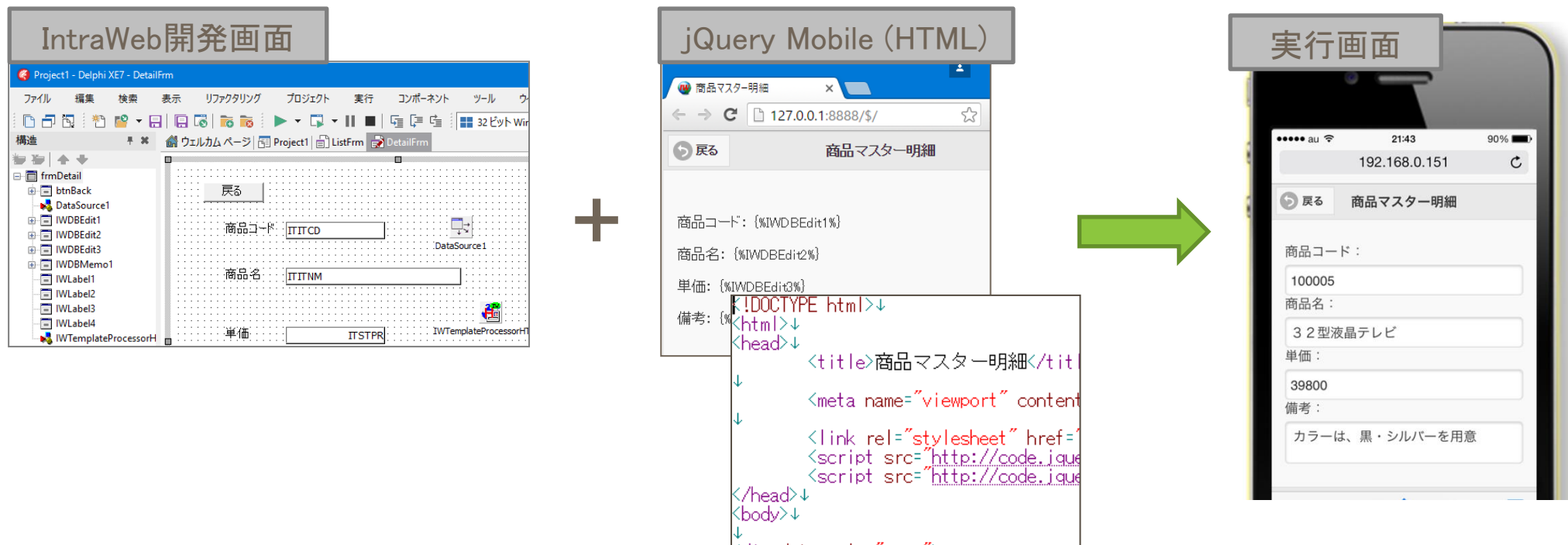


ASCII.jp 「jQuery Mobileを使った国内スマホサイトまとめ」より

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

商品コード	商品名
100001	婦人用ワンピース
100002	婦人用カーディガン
100003	紳士用靴下 クロ
100004	紳士用ネクタイ
100005	32型液晶テレビ
100006	5ドア冷蔵庫
100007	全自動洗濯乾燥機

明細画面

戻る

商品コード: 100005

商品名: 32型液晶テレビ

単価: 39800

備考: カラーは、黒・シルバーを留意



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

192.168.0.151

商品マスター一覧

- 婦人用ワンピース
- 婦人用カーディガン
- 紳士用靴下 クロ
- 紳士用ネクタイ
- 32型液晶テレビ
- 5ドア冷蔵庫
- 全自動洗濯乾燥機

明細画面

192.168.0.151

戻る 商品マスター明細

商品コード: 100005

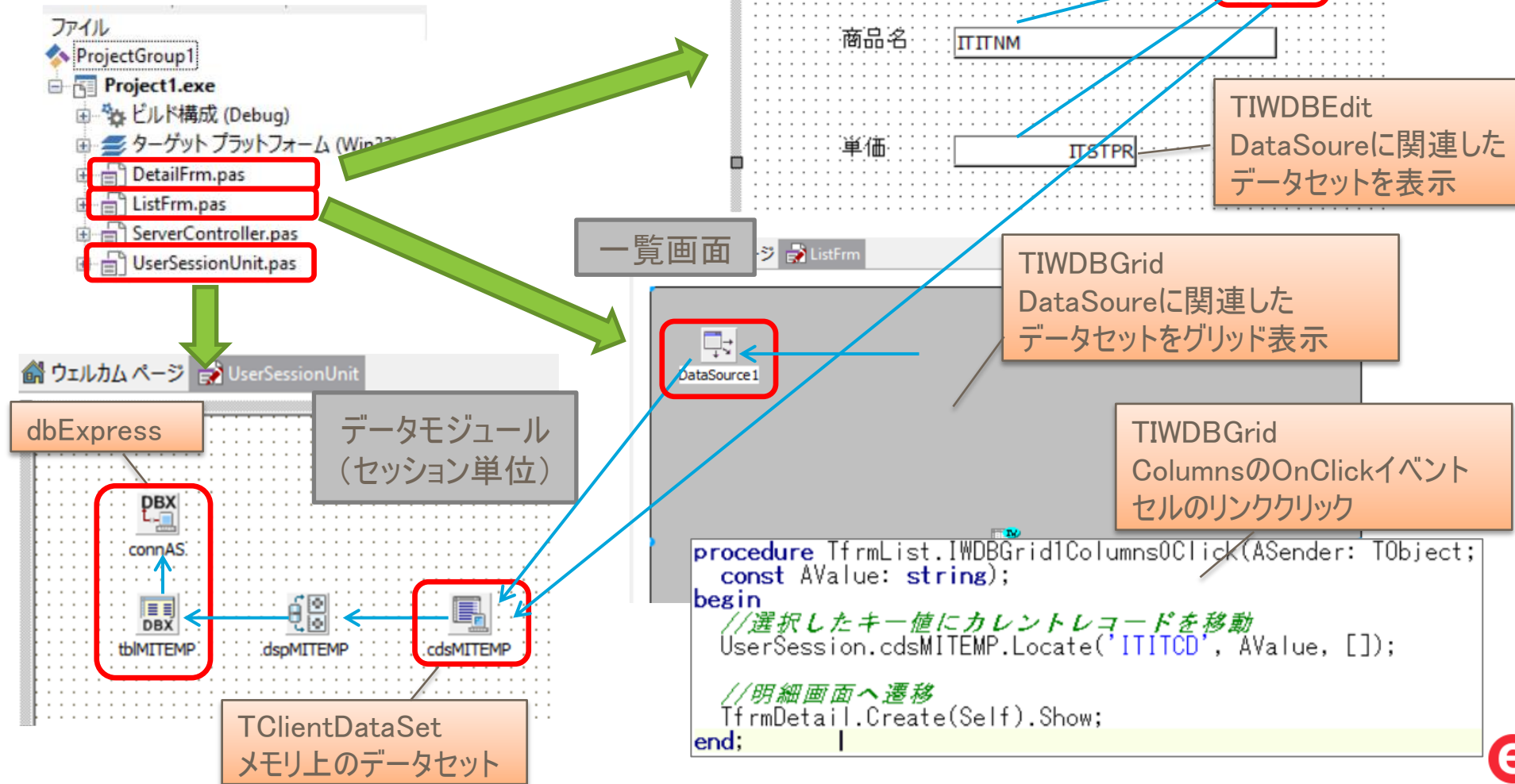
商品名: 32型液晶テレビ

単価: 39800

備考: カラーは、黒・シルバーを留意

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公式サイトの「Demos」リンクが強調されています。

「Putting it together: Basic single page template」リンクが強調されています。

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/[version]/jquery.mobile-[version].css">
<script src="http://code.jquery.com/jquery-[version].min.js"></script>
<script src="http://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.js"></script>
```

[version]部分を"1.4.5"に置換

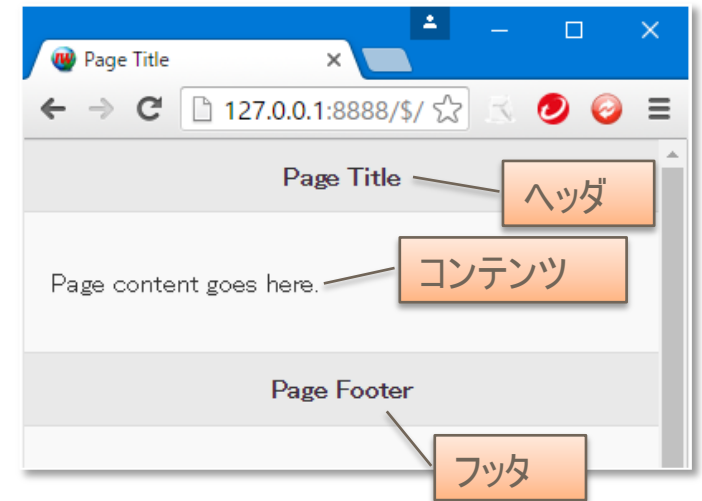
リンク先にあるHTMLソースをメモ帳等に貼り付け

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

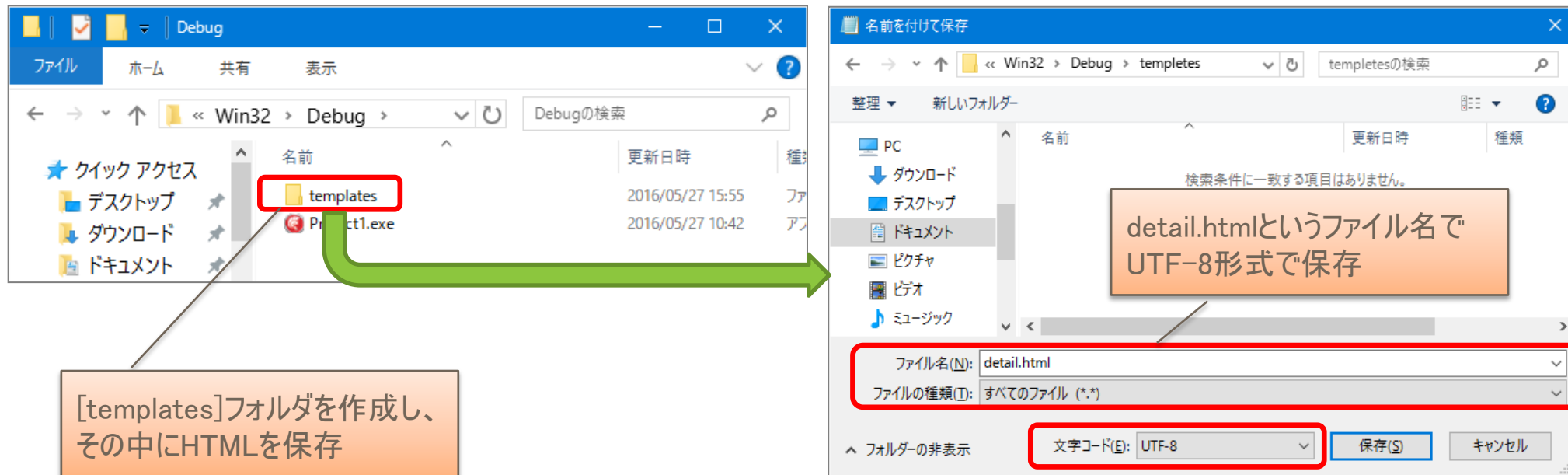
Basic single page template

```
!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]
```



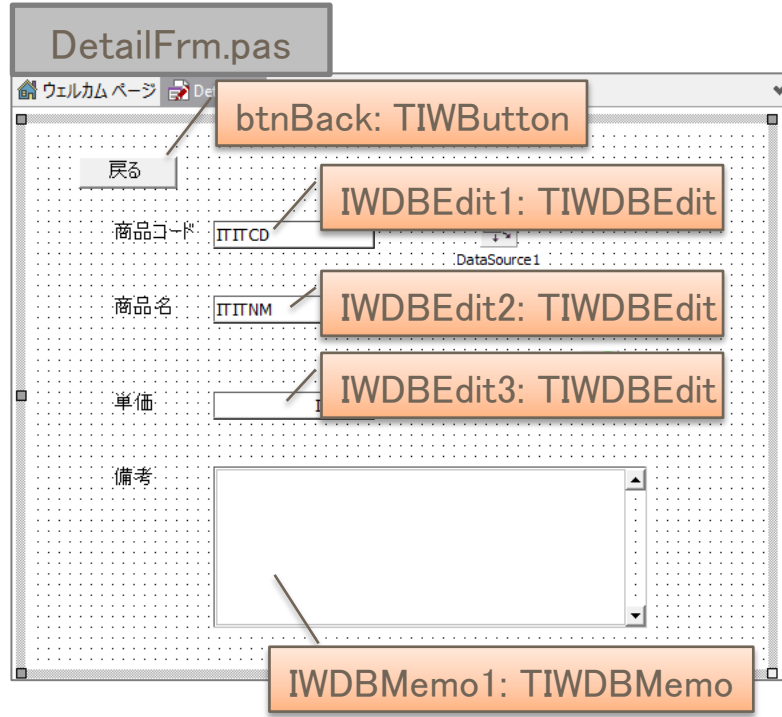
明細画面の作成

- 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。



```
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%]</div>↓<label for="IWDBEDIT2">商品名 : </label>↓[%IWDBEdit2%]</div>↓<label for="IWDBEDIT3">単価 : </label>↓[%IWDBEdit3%]</div>↓<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

The screenshot illustrates the configuration of the IWTemplateProcessorHTML1 component in a Delphi IDE. The main window shows a form with a grid layout containing text labels like '戻る', '商品コード', 'ITITCD', 'ITITNM', and 'ITSTPR'. A component 'IWTemplateProcessorHTML1' is placed on the grid. Three callout boxes with arrows point to specific configurations:

- frmDetailとIWTemplateProcessorHTML1とを関連付け**: Points to the 'LayoutMgr' property in the 'frmDetail' Object Inspector, which is set to 'IWTemplateProcessorHTML1'.
- RenderStylesをFalseに指定**: Points to the 'RenderStyles' property in the 'IWTemplateProcessorHTML1' Object Inspector, which is set to 'False'.
- 作成したhtmlファイルを指定**: Points to the 'Default' sub-property under the 'Templates' property in the 'IWTemplateProcessorHTML1' Object Inspector, which is set to 'detail.html'.

Additional elements in the screenshot include the 'Project1.dproj - プロジェクト マネージャ' window with 'DetailFrm.pas' selected, and the 'Object Inspector' for 'frmDetail' showing other properties like 'HelpType', 'Hidden', 'Hint', 'HorzScrollBar', 'JavaScript', 'KeepAlive', and 'Left'.

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



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



様々なタイプのリストビュー
サンプルが記載



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

```
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</h4>↓
</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

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

The image illustrates the configuration of the IWTemplateProcessorHTML1 component for a list view. It shows the component in the IDE, the HTML template code, and the corresponding event handler procedure.

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.</h4>↓
</div><!-- /footer -->↓
</div><!-- /page -->↓
```

タグを埋め込みたい箇所に {% 名前 %} と定義

オブジェクトインスペクタ

プロパティ	イベント
OnAfterProcess	
OnBeforeProcess	
OnUnknownTag	IWTemplateProcessorHTML1Unk

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(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



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)

<p>商品コード</p>
<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

ListFrm.pas

一覧画面

EnabledをFalseにすると
IntraWebのフォームで実行

PCブラウザで実行し、
HTMLソースを表示

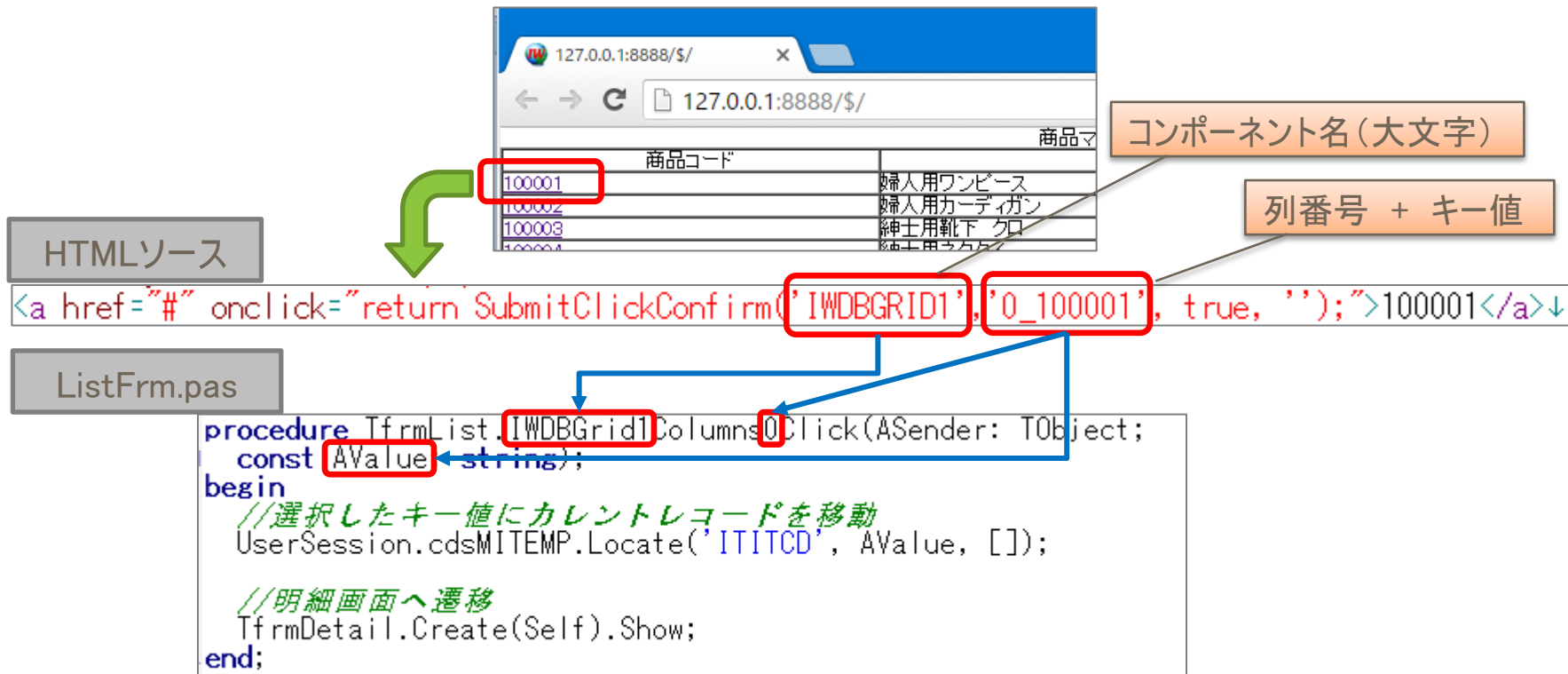
HTMLソース(抜粋、整形済)

```
<form onsubmit="return FormDefaultSubmit();">
  <div class="IWDBGRID1CSS" id="IWDBGRID1">
    <table id="TBLIWDBGRID1" border="1" cellpadding="0" cellspacing="0" width="799">
      <caption>
        商品マスター一覧
      </caption>
      ~~~ (中略) ~~~
      <tr id="627862241">
        <td valign="middle" align="left" style="white-space: nowrap;position:static;">
          <font style="font-size:13px;">
            <a href="#" onclick="return SubmitClickConfirm('IWDBGRID1','0_100001', true, '');">100001</a>
          </font>
        </td>
        <td valign="middle" align="left" style="white-space: nowrap;position:static;">
          <font style="font-size:13px;">婦人用ワンピース</font>
        </td>
      </tr>
    </table>
  </div>
</form>
```

<a>要素にonclick属性が
定義されており、javascript
のサブルーチンが呼び出されている

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

一覧画面の作成

■ 実行（完成）



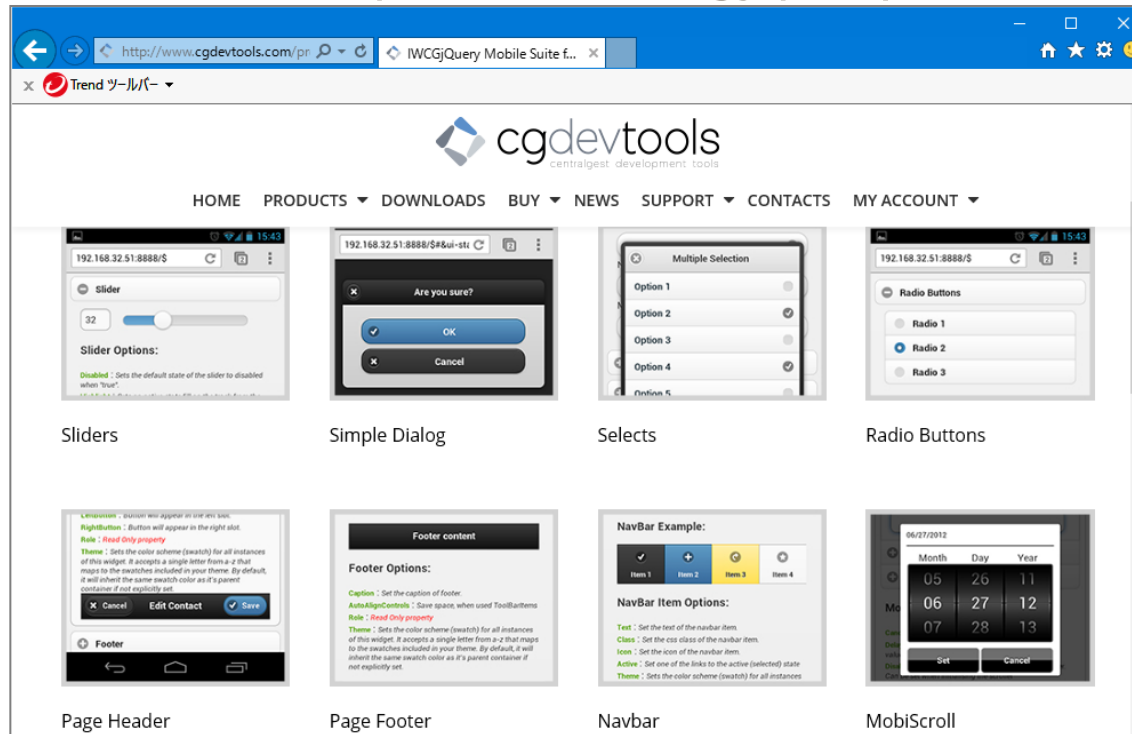
jQuery Mobileの活用で、スマートデバイスに適した画面が作成可能！

(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

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 単元にテスト実施

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)単元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。

```
30 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;
```

ブレークポイント

Result | 28

■ 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。

The screenshot shows a code editor with a C# snippet and the Windows Event Viewer. The code snippet is:

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

A callout box labeled "デバッグ出力ロジック" points to the code. Below the code, a red box highlights the following line:

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

The Event Viewer window shows a list of events. A red box highlights the following event:

```
デバッグ出力: GetLastDay : 28 プロセス ProjectTec18_1.exe (12080)
```

A callout box labeled "デバッグ実行時 イベントログに結果が出力" points to this event.

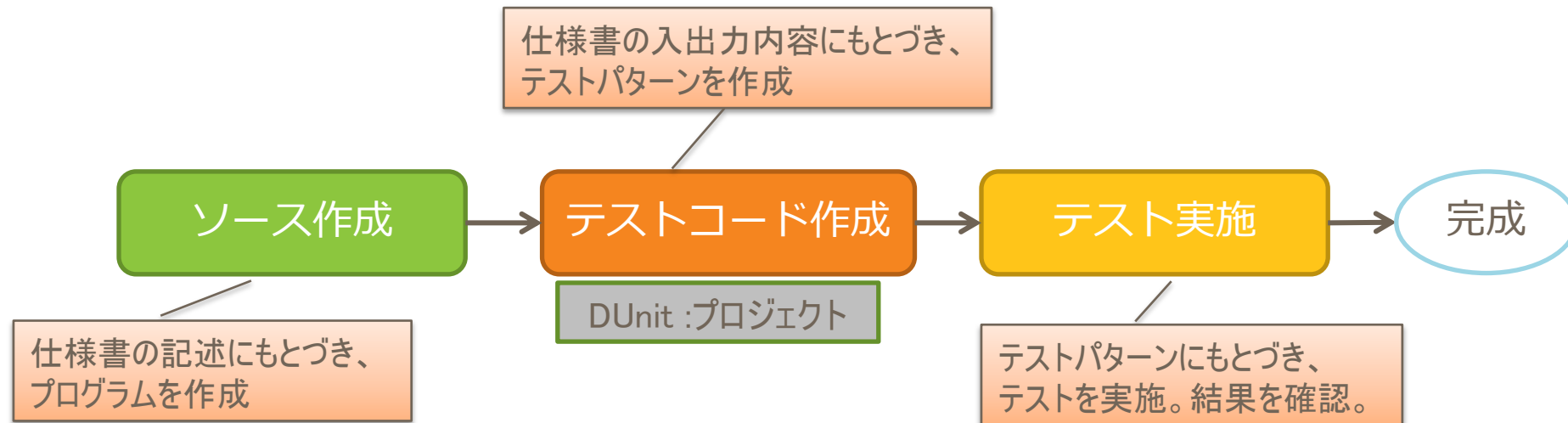
■ デバッグプリントの課題

- 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？

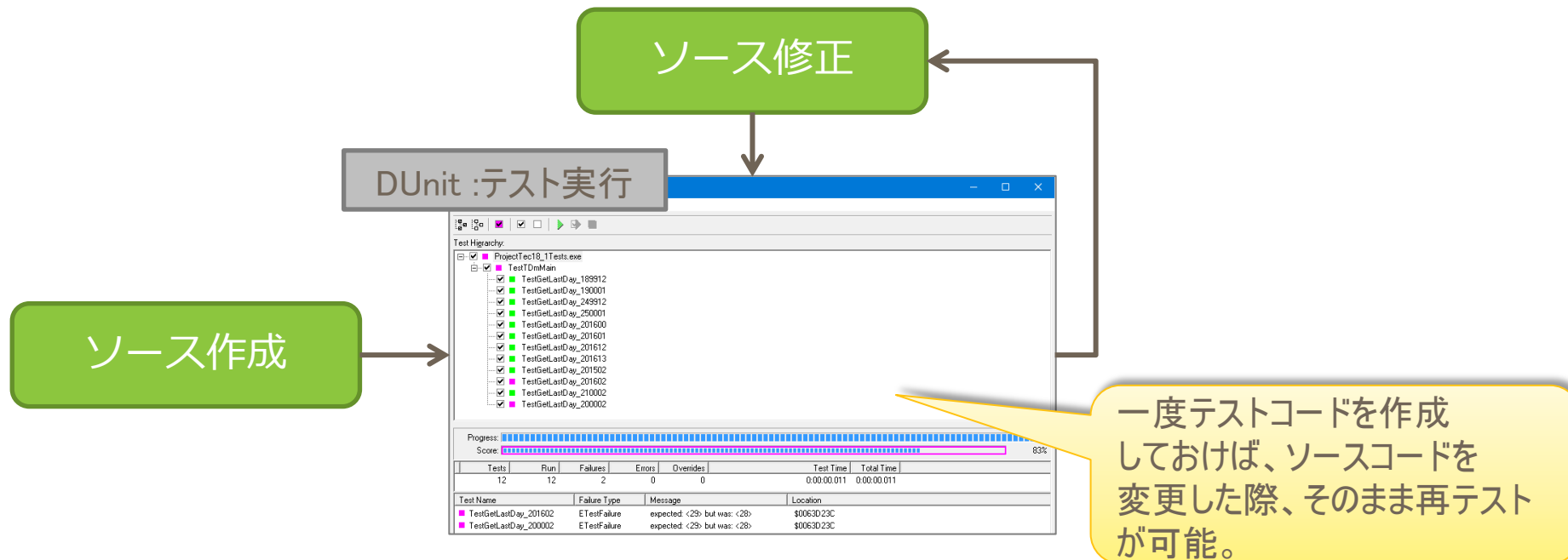
テストフレームワーク DUnit

- 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
 - 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



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

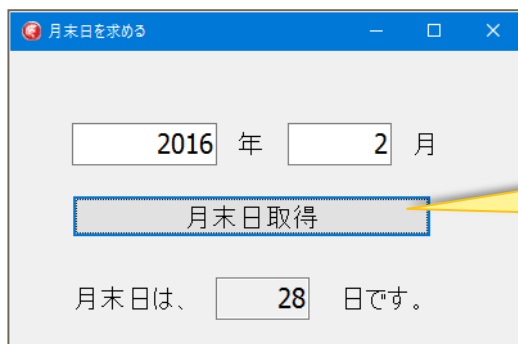
■ GetLastDayメソッドのコーディング

```
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

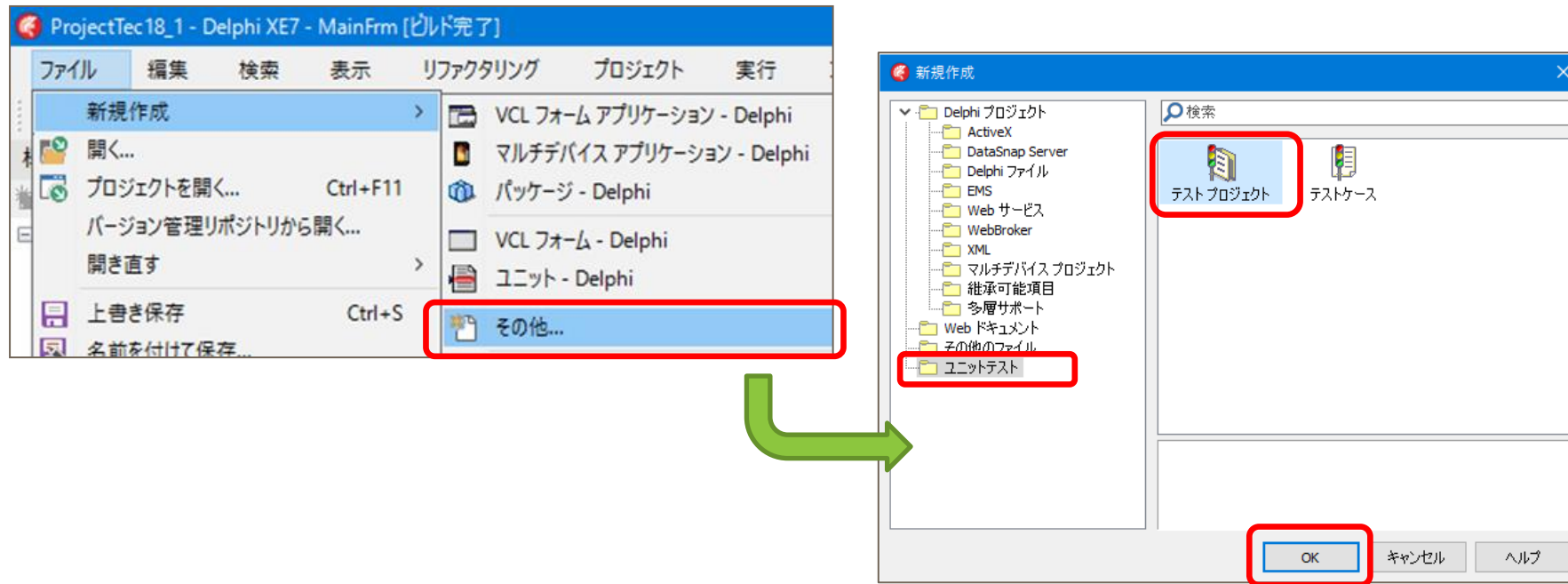


うるう年の考慮がないので、今年(2016年)は、2月は29日のはずが、28日と表示。(メソッドの修正が必要)

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

- 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

テストプロジェクトウィザード - ステップ 1 / 2

テストプロジェクトの詳細を指定
下記フィールドに入力してテストプロジェクト名、作成するプロジェクトのタイプを指定してください

ソースプロジェクト(S): ProjectTec18_1

プロジェクト名(N): ProjectTec18_1Tests

位置(L): C:\Users\OZAKI\Documents\個人フォルダ\01_現在活...

パーソナリティ(P): Delphi

プロジェクトグループ

テストプロジェクトウィザード - ステップ 2 / 2

テストフレームワークオプションの指定
テストフレームワークとテストランナーの選択

テストフレームワーク(A): DUnit / Delphi Win32

テストランナー(R): GUI

ProjectTec18_1Tests.dproj - プロジェクトマネージャ

ファイル

- ProjectGroup1
 - ProjectTec18_1.exe
 - ビルド構成 (Debug)
 - ターゲットプラットフォーム (Win32)
 - MainDM.pas
 - MainFrm.pas
 - ProjectTec18_1Tests.exe**
 - ビルド構成 (Debug)
 - ターゲットプラットフォーム (Win32)

C:\Users\OZAKI\Documents\個人フォルダ\01_現在活...
ProjectTec18_1... モデルビュー | データエクスプロ...

テスト対象のプロジェクトがデフォルト表示される。

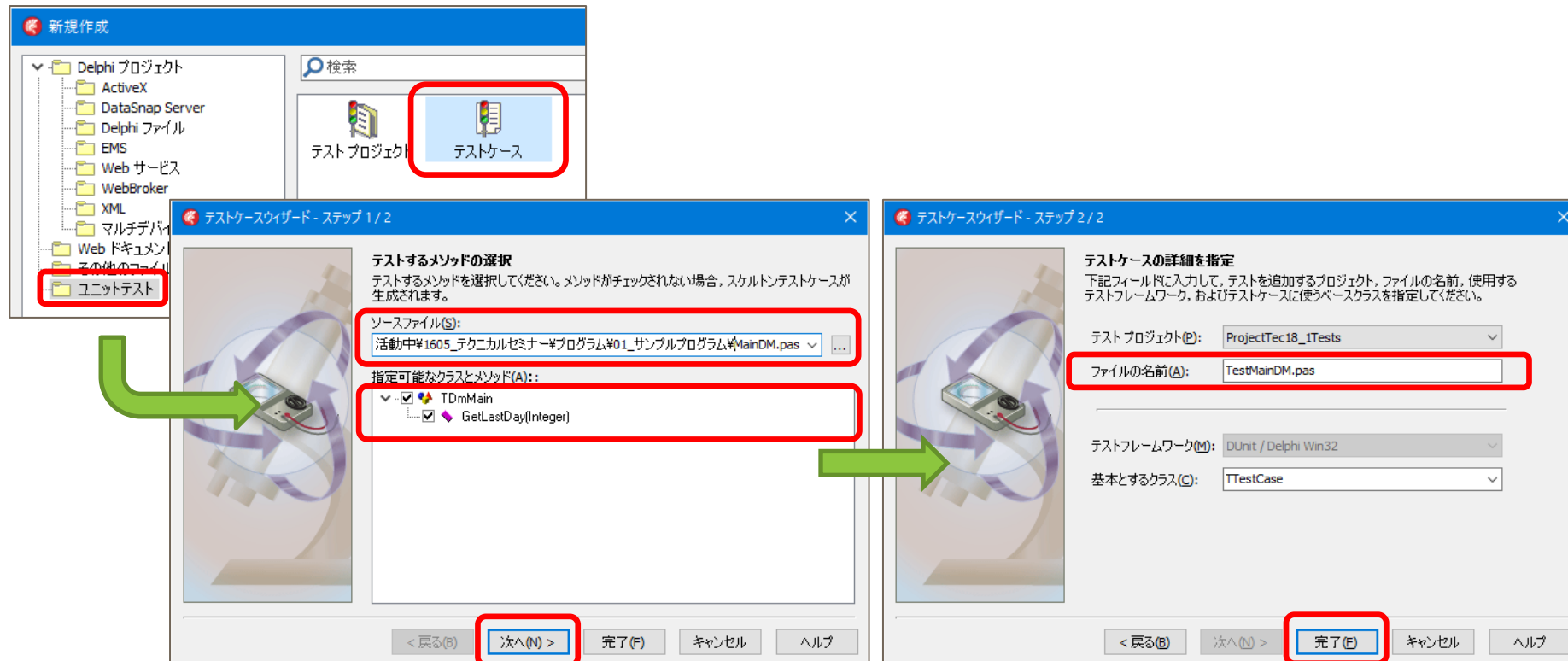
GUIでテストするか、コンソールでテストするかを選択。

テスト用プロジェクトが追加作成。

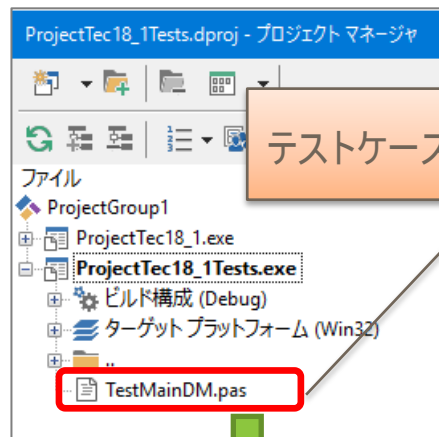
テストケースの作成

■ 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



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

- 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

- テストプロジェクトを[実行]

The image shows two screenshots of the DUnit testing framework. The left screenshot shows the 'Test Hierarchy' with 'TestGetLastDay' selected. A green arrow points to the right screenshot, which shows the test results. A legend indicates that green represents 'テスト成功' (Test Success) and pink represents 'テスト失敗' (Test Failure). The results table shows one test failed. The failure message is 'expected: <29> but was: <28>'. A callout box explains that the failure reason is displayed: '期待値29に対し、28が結果であった' (Expected value 29, but result was 28).

テスト項目を選択し、実行

実行結果が色分けで表示
■ : テスト成功
■ : テスト失敗

Tests	Run	Failures	Errors	Overrides	Test Time	Total Time
1	1	1	0	0	0:00:00.001	0:00:00.003

Test Name	Failure Type	Message	Location
TestGetLastDay	ETestFailure	expected: <29> but was: <28>	\$0063CDFF

TestGetLastDay: ETestFailure
at \$0063CDFF
expected: <29> but was: <28>

テスト失敗の原因が表示
期待値29に対し、28が結果であった

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

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

Tests	Run	Failures	Errors	Overruns	Test Time	Total Time
12	12	2	0	0	0:00:00.011	0:00:00.011

Test Name	Failure Type	Message	Location
TestGetLastDay_201602	ETestFailure	expected: <29> but was: <28>	\$0063D23C
TestGetLastDay_200002	ETestFailure	expected: <29> but was: <28>	\$0063D23C

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;
```

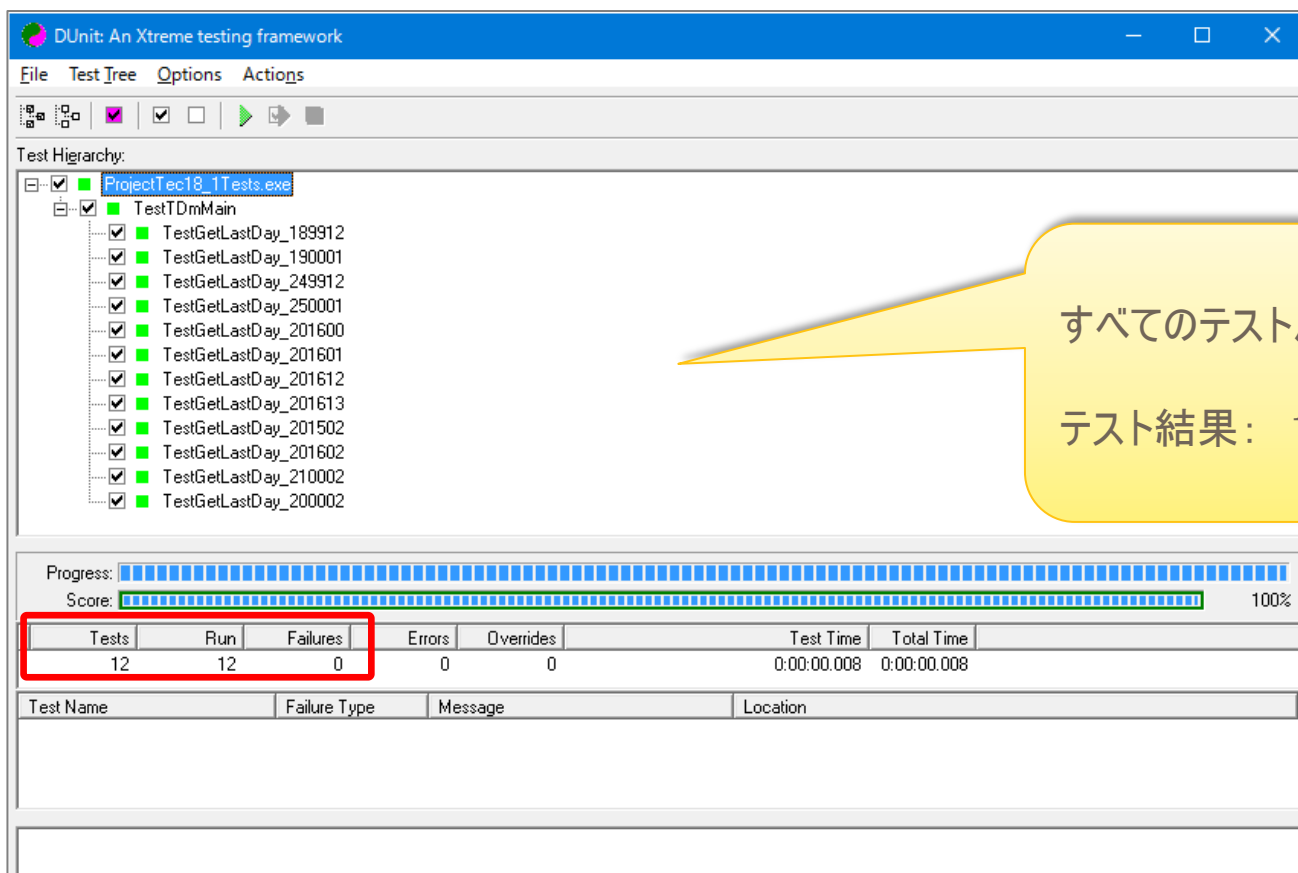
うるう年の判定ロジック

(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



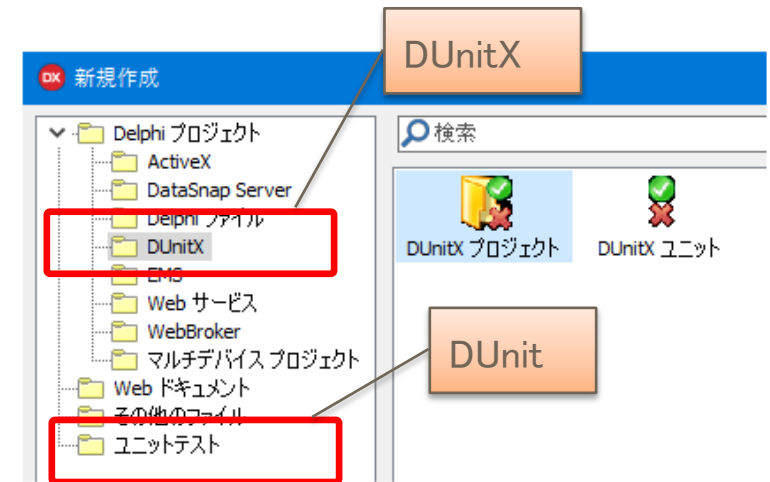
すべてのテストパターンにOK

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！

(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

■ モバイルアプリケーション開発のコツ

PCアプリとモバイルアプリの違い

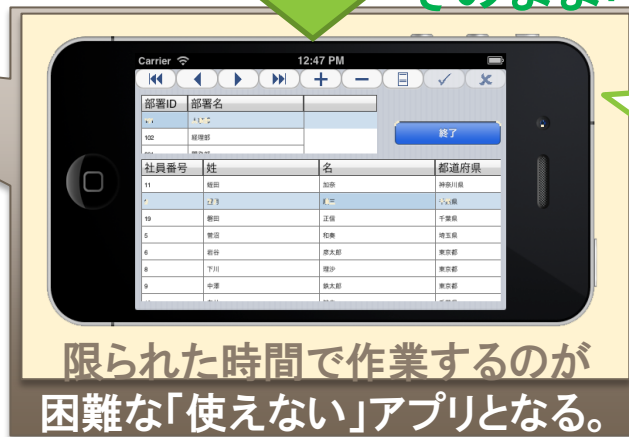
PC



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



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！

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！



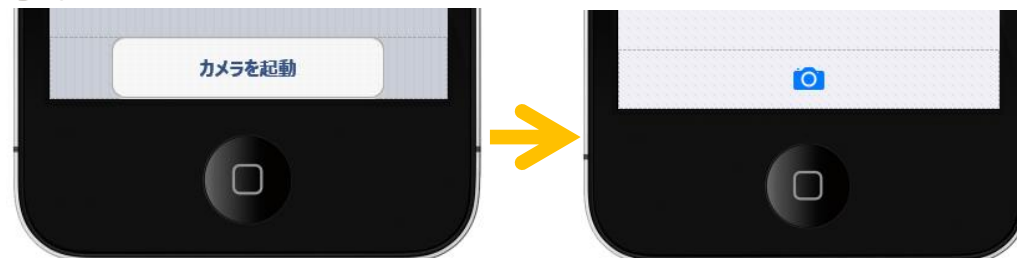
モバイル画面設計のポイント

- 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ピクセル×768ピクセル
iPad3以降 : 2048ピクセル×1536ピクセル
 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。

iPad 専用アプリ



- 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

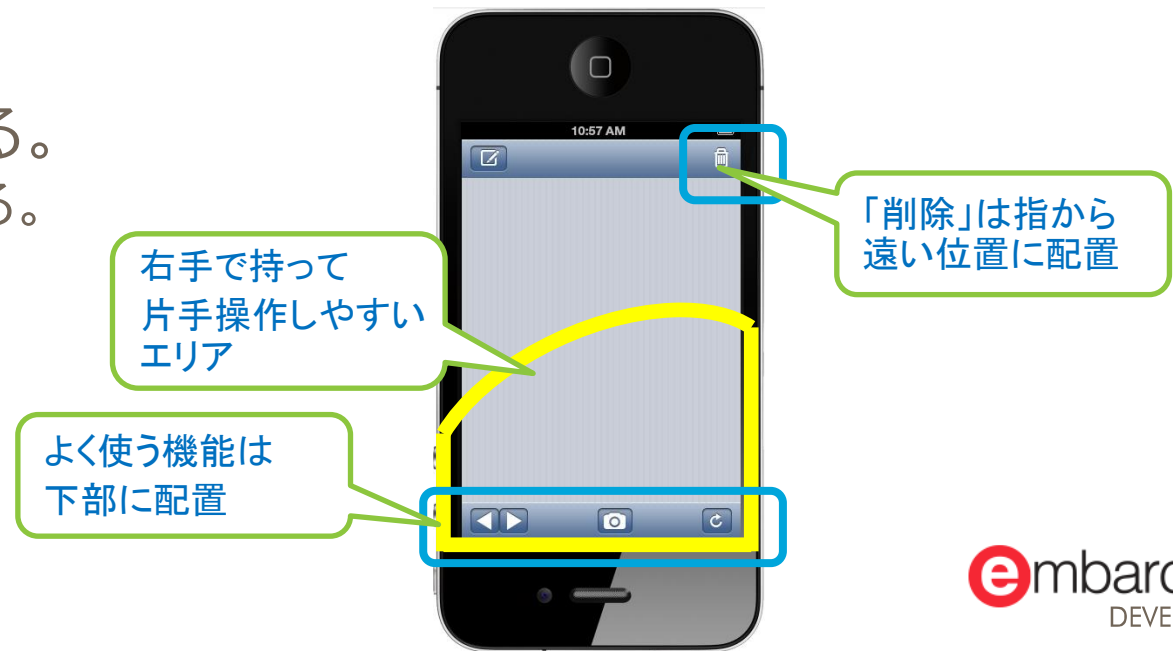
- デバイスの違いを意識する

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。



モバイルアプリ開発のコツ

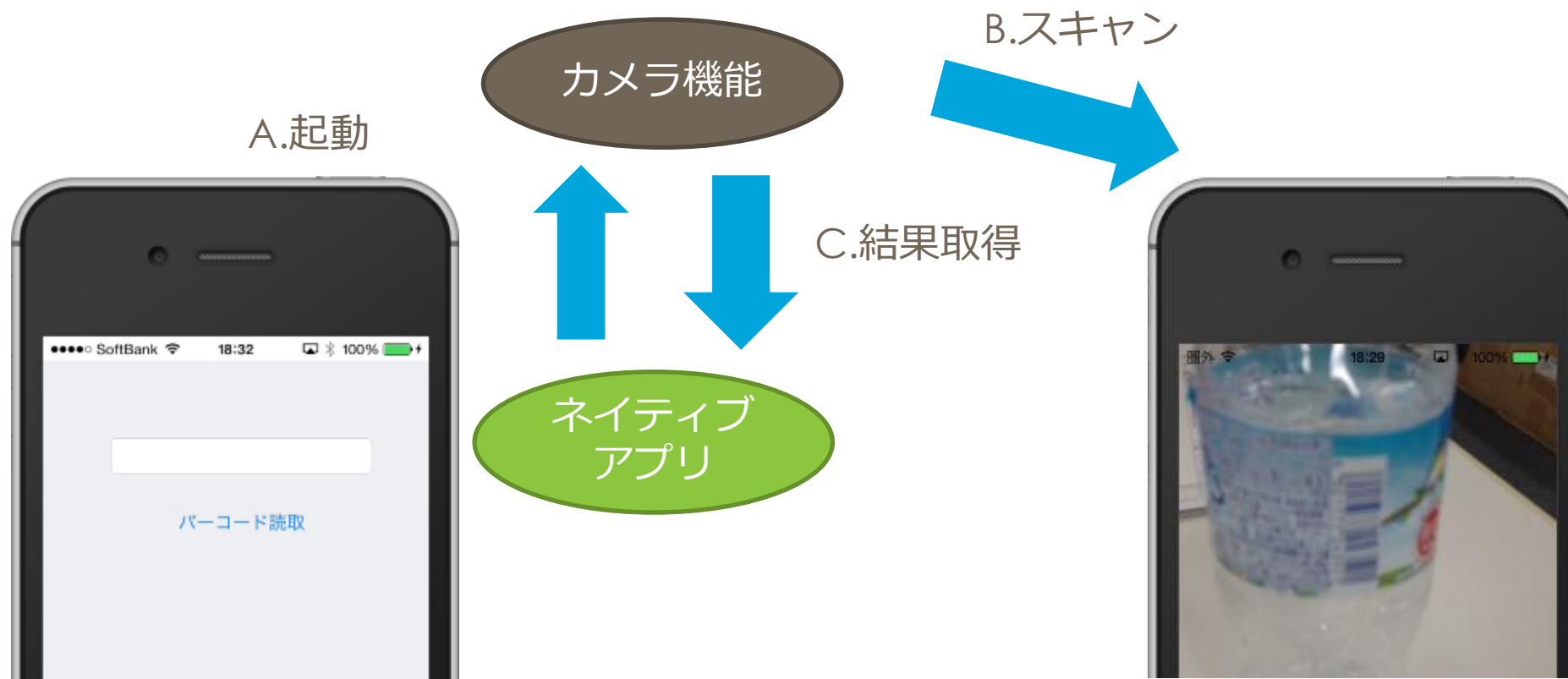
- モバイル開発では、デバイス機能の活用が重要。

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）

(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TKRBarcodeScannerコンポーネントが便利。

【TKRBarcodeScanner】

※iOS / Android可能

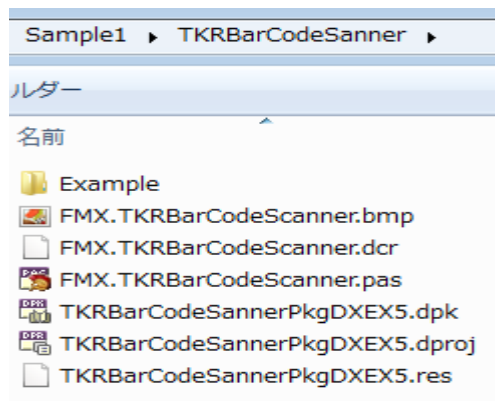
(iOS使用時はZBarSDKもインストールが必要)

<http://www.fmxexpress.com/qr-code-scanner-source-code-for-delphi-xe5-firemonkey-on-android-and-ios/>

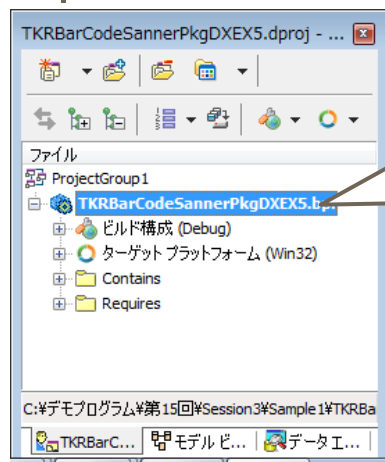
→ 今回はこのTKRBarcodeScannerコンポーネントを使用

(1)カメラを使ったバーコード読み取り機能

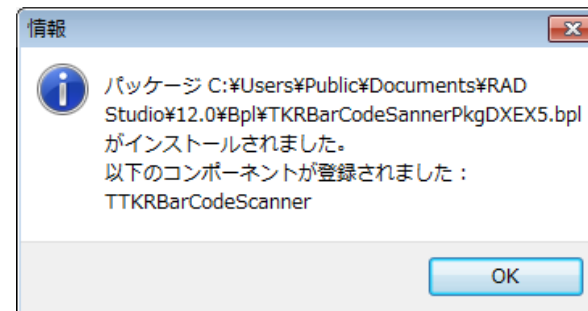
- TKRBarcodeScannerコンポーネントのインストール①
 - TKRBarcodeScanner.zipをダウンロードして展開



- [ファイル|プロジェクトを開く]よりTKRBarcodeScannerPkgDXEX5.dpkを開きます。

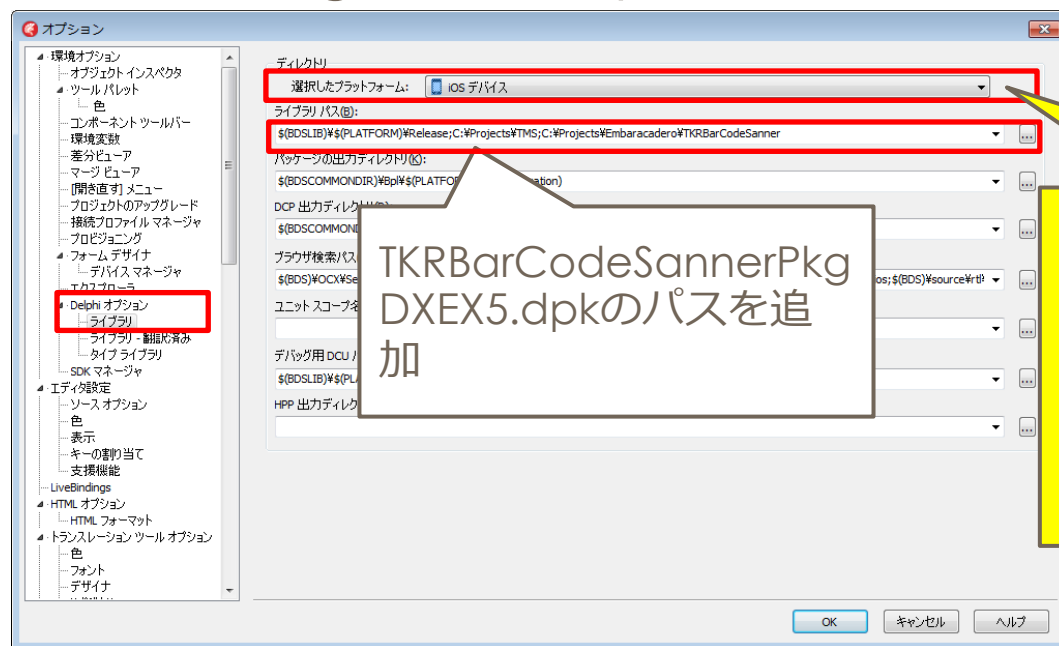


プロジェクトマネージャ
で右クリックからコンパ
イルとインストールを実
行



(1)カメラを使ったバーコード読み取り機能

- TKRBarcodeScanner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TKRBarcodeScannerPkgDXEX5.dpkを開いたパスを追加



【ポイント】
使用するプラットフォームを選択しておく必要があります

ディレクトリ
選択したプラットフォーム: iOS デバイス

- コンポーネントの登録が完了！



TKRBarcodeScanner

embarcadero
DEVELOPER CAMP

(1)カメラを使ったバーコード読み取り機能

- TKRBarcodeScanner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 TKRBarcodeScanner.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}
```

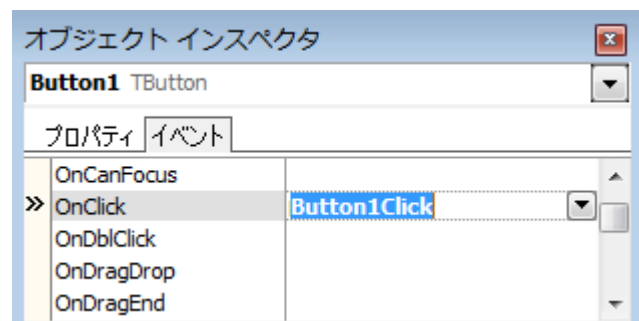
(1)カメラを使ったバーコード読み取り機能

- 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
TKRBarcodeScanner、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



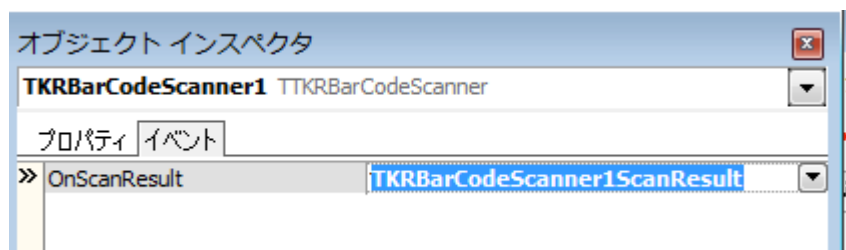
A.カメラ起動 B.スキャン

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
 - TKRBarcodeScanner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



(2) Beacon機能の活用

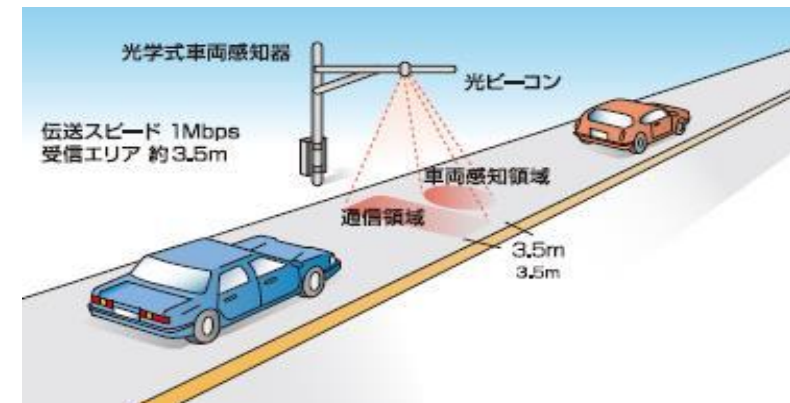
- Beacon（ビーコン）とは？
 - IoTが身近なところで代表される機能のひとつがBeacon（ビーコン）。BeaconはBluetoothを使い、半径十数メートル信号を発信する。用途としては、スマートフォンなどの機器が信号範囲に入ってきたら情報を通知したり（お店のクーポン配信等）、信号範囲から位置を特定して位置情報を使った機能として使うこともできる。

【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。



(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

(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

(2) Beacon機能の活用

- Beacon受信サンプルアプリ開発
 - 機能概要



② Beaconの信号エリアに入るとアプリで広告表示！



① Beacon機器※から信号を発信

Beacon発信エリア



Beacon機器

※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

The image shows a sequence of three screenshots from Xcode illustrating the configuration of the TBeaconRegionCollection property. The first screenshot shows the 'MonitorizedRegions' property in the interface, with a callout box stating 'MonitorizedRegionsプロパティをダブルクリック' (Double-click the MonitorizedRegions property). The second screenshot shows the '0 - TBeaconRegionItem' in a list, with a callout box stating 'アイテムを追加' (Add item). The third screenshot shows the 'オブジェクトインスペクタ' (Object Inspector) for 'Beacon1.MonitorizedRegions[0] TBeaconRegionItem', with a callout box stating 'Beaconに合わせたUUIDをセット' (Set the UUID matching the Beacon). The 'UUID' field in the inspector is highlighted with a red box and contains the value '{00000000-0000-0000-0000-000000000000}'.

(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;
```

OnBeaconExit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;
```

THANKS!

www.embarcadero.com/jp

第32回 エンバカデロ・デベロッパーキャンプ