

【セッションNo.2】

# Delphi/400を使用した 最新モバイルアプリ開発入門

株式会社ミガロ.  
プロダクト事業部 技術支援課  
尾崎 浩司



# 【アジェンダ】

## Delphi/400を使用した最新モバイルアプリ開発入門

- はじめに
- モバイルアプリ開発の基本
- IBM i DB情報を活用するモバイルアプリ開発手法
- さいごに

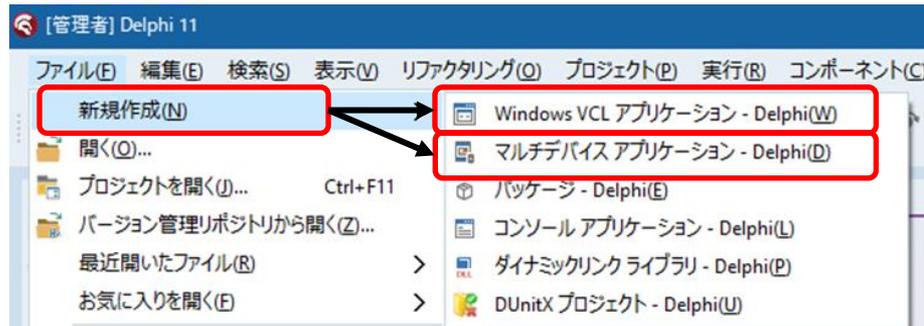


# はじめに

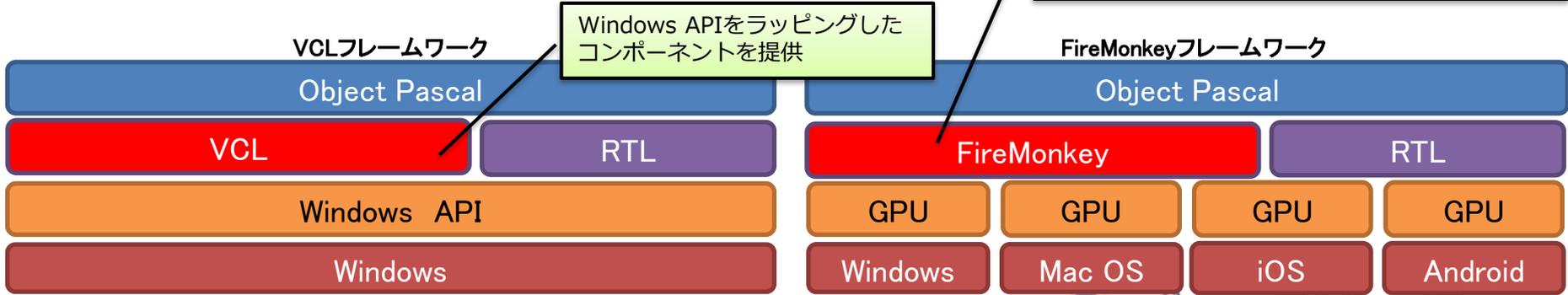


# ■ Delphi/400におけるGUIフレームワーク

- 2つのGUIフレームワークが使用可能
  - Windows VCL アプリ (VCL)**
    - Windows APIをラッピングした従来から使用可能なフレームワーク
    - Windowsアプリ開発専用
  - マルチデバイスアプリ (FireMonkey)**
    - Delphi/400 XE5以降に搭載されたマルチデバイスアプリ開発用フレームワーク
    - Win/Mac/iOS/Androidアプリ開発に対応



各プラットフォームのGPUを利用して、独自の画面を描画するコンポーネントを提供  
視覚効果がより高いアプリが作成可能



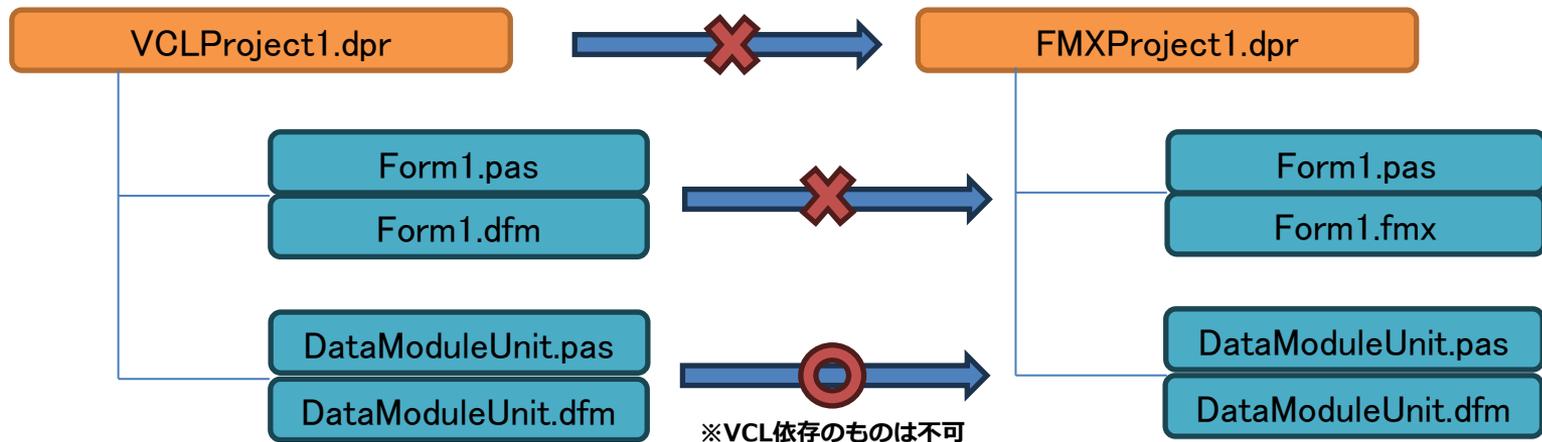
# ■ Delphi/400におけるモバイルアプリ開発

- モバイルアプリ開発には、**FireMonkey**を使用する
  - iOS, Androidのいずれのモバイル用OS向けアプリ開発にも対応可能。
  - ターゲットとなるデバイスに加え、iOSの場合、モバイルアプリをビルド・配置する為**Mac**や**Apple Developer Program**への加入が別途必要。



# ■ Delphi/400におけるモバイルアプリ開発

- VCLとFireMonkeyは、**コンポーネント自体が異なる**ので互換性は無い
  - VCLプロジェクトをそのままFireMonkeyプロジェクトには変換できない
  - データモジュールやユニットは流用できる事が多い
    - 但し、配置されたコンポーネントや処理がVCL専用の物を使用している場合は流用不可。
    - データモジュールの場合、ClassGroupプロパティを[System.Classes.TPersistent]にすると、フレームワークに依存しないコンポーネントのみデータモジュール上に配置できるようになる。



# ■ Delphi/400におけるモバイルアプリ開発

- 言語は、いずれのフレームワークもObject Pascalを使用
- 文字列リスト(TStringList) のようなRTLは共通に使用可能
  - RTL (ランタイム ライブラリ) の利用  
[https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/RTL\\_\(ランタイム\\_ライブラリ\)\\_の利用](https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/RTL_(ランタイム_ライブラリ)_の利用)
- TEditやTButtonのようにコンポーネントは同名のものが多く、VCL同様に直感的な開発は可能だが、**似て非なるものとしてFireMonkeyの流儀を知る事**が重要
  - VCL と FireMonkey のよくある違い  
[https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/VCL\\_と\\_FireMonkey\\_のよくある違い](https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/VCL_と_FireMonkey_のよくある違い)

今回は、**FireMonkeyを使用したモバイルアプリ**開発の基本についてご紹介！



# モバイルアプリ開発の基本



# ■ PCとモバイルの画面の違い

- PCアプリの場合、フォームは実行時にウィンドウとしてスクリーンの一部に表示

- PCは、モニターへの表示が前提であり、設計者が決めたフォームサイズで固定的にレイアウトを行っても問題となる事は少ない。



モニターのサイズに関わらず、通常ウィンドウとして設計したサイズでフォームが表示される。固定的なレイアウト配置でも問題になりにくい。

- モバイルアプリの場合、原則アクティブなフォームが常に全画面表示となる

- スマートフォンだけでなくタブレットもあり、デバイス毎に画面の大きさが異なる為、固定的なレイアウトを行うと問題が発生する可能性が高い



モバイルアプリは通常ディスプレイ全体に全画面表示される。スマホやタブレット等デバイスにより画面サイズが異なる為、PCアプリのような固定的なレイアウト配置では問題が発生しやすい



# ■ モバイルアプリのフォーム

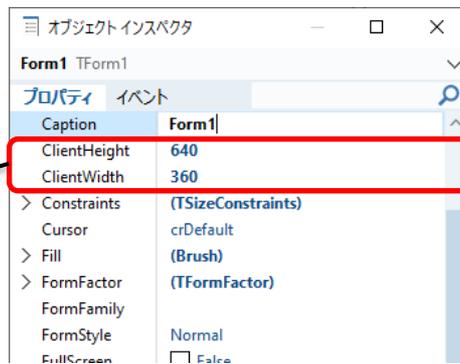
- フォーム(TForm) のサイズ
  - スマートフォンをターゲットとするアプリであれば、縦横比**16:9**となる以下のサイズを基準に設計すればよい。

**ClientHeight = 640**

**ClientWidth = 360**

- ターゲットとするデバイスのOSにあわせた**スタイルを選択**

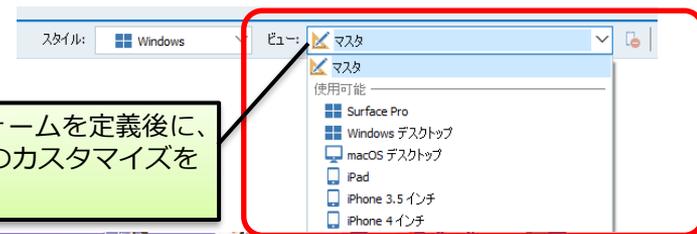
デバイスによって、縦横比は異なるが（近年は、縦に長い傾向）、**16:9**を前提とすれば、凡そのデバイスは網羅可能。



スタイルを指定すると、ターゲットとなるOS (Android/iOS) のUIを模倣したデザインで設計可能

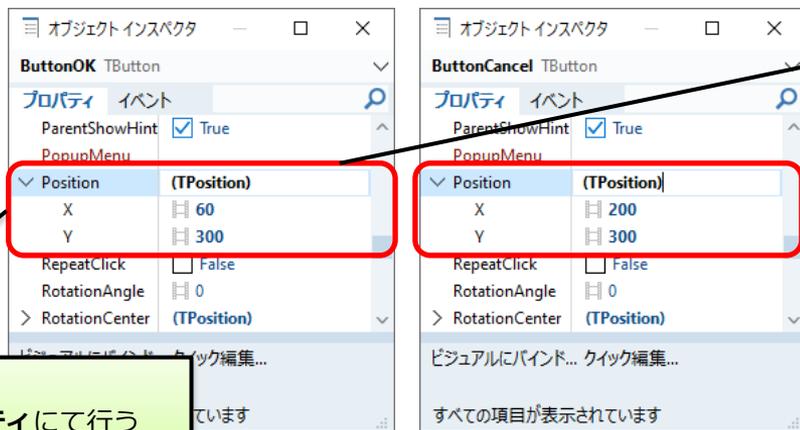
- スマホ/タブレット/PC全てをターゲットとするアプリを作成するのであればマルチデバイスデザイナー (ビュー) を活用する事も可能 (必須ではない)

基準となる[マスタ]フォームを定義後に、デバイス毎のフォームのカスタマイズを行い最適化する仕組み



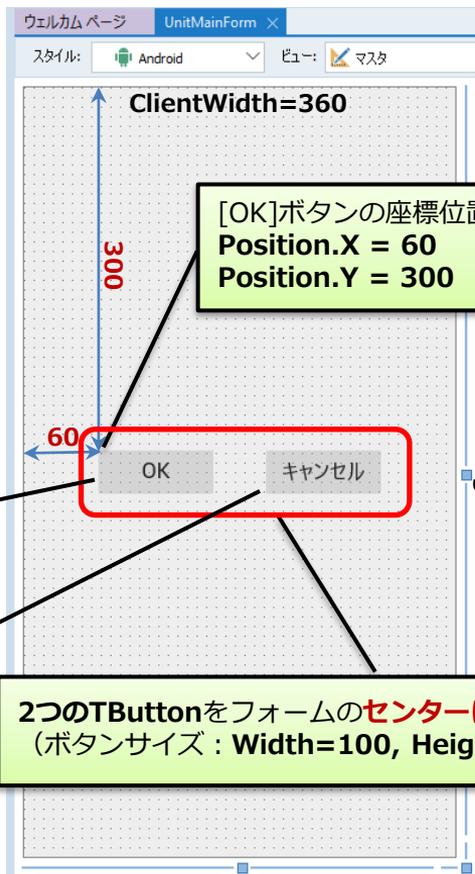
# ■ 固定的なレイアウト配置

- VCLの場合、各コンポーネントの配置は、Top/Leftプロパティによる絶対座標で指定する事が多い。
- FireMonkeyの場合、Position.X/Position.Yプロパティで絶対座標が指定可能。



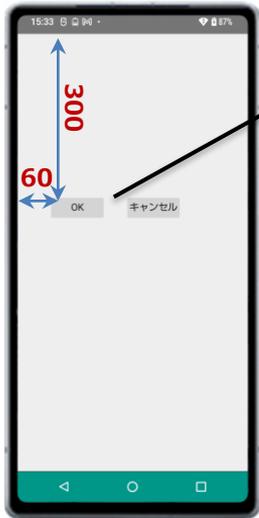
FireMonkeyの場合、絶対座標指定はPositionプロパティにて行う

【フォームデザイナー】



# ■ 固定的なレイアウト配置

- Androidスマートフォンで実行  
5.7インチ(720×1520px)のデバイス



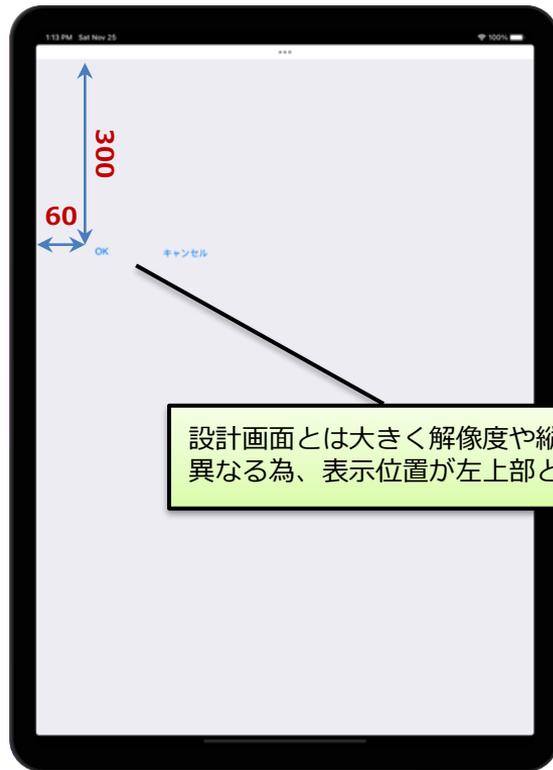
縦横比が設計画面と異なる為、上下/左右ともセンター位置から若干ずれている



スマートフォンを横表示にすると、縦横比が大きく変わる為、表示位置が左下部になる

多様なデバイスで常にセンター部にボタンを配置するには、どうすればよいか？

- iPadで実行  
11インチ(1668×2388px)のデバイス



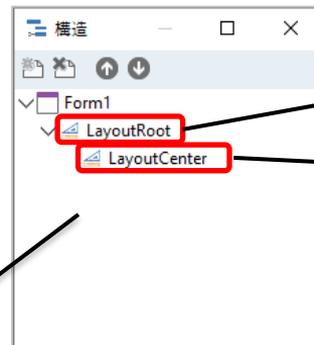
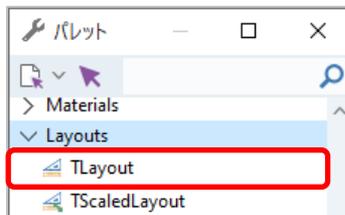
設計画面とは大きく解像度や縦横比が異なる為、表示位置が左上部となる



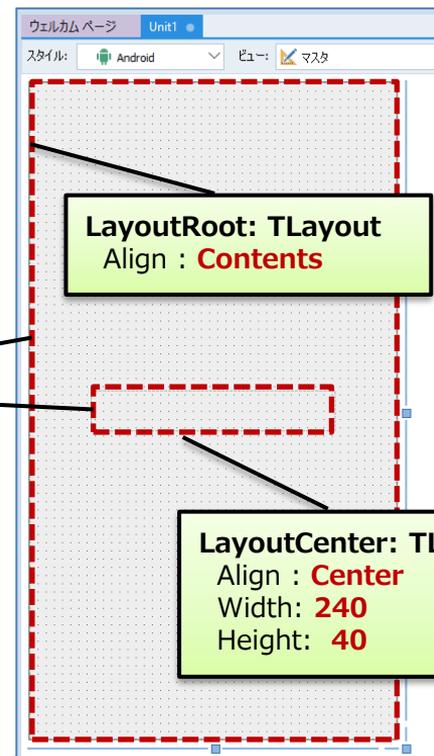
# ■ モバイルに最適なレイアウト作成方法

## • TLayoutコンポーネント

- レイアウトを配置する為のコンテナコンポーネント
- 実行時は何も表示されない
- **Alignプロパティ**で配置を調整
- フォームの上に直接コンポーネントを配置せず、TLayoutの上に配置する
- TLayoutは**入れ子**にして利用する



【フォームデザイナー】



画面全体の土台となるTLayoutコンポーネントをフォームの上に配置。(LayoutRoot)  
配置を設定するAlignは、親(フォーム)コンポーネント全体にあわせてサイズが変更されるContentsを指定。

LayoutRootの子として、さらにTLayoutコンポーネントを配置。(LayoutCenter)  
Alignは、親コンポーネントの中央位置に設定されるCenterを指定。



# ■ モバイルに最適なレイアウト作成方法

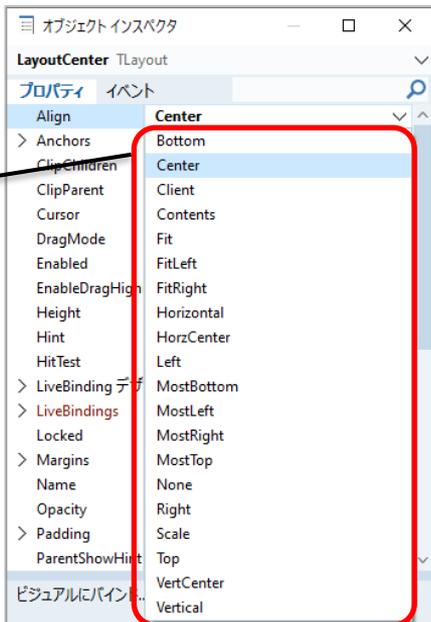
## • Alignプロパティ

- 親コンポーネントに対するコンポーネントの相対的な位置を設定するプロパティ

【設定値一覧】

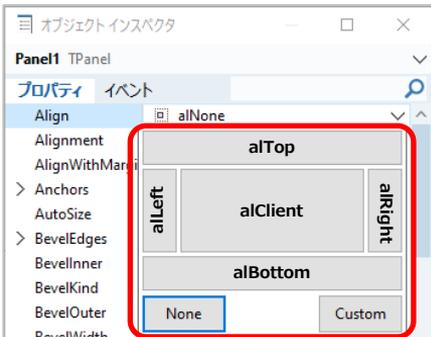
FireMonkey	コンポーネントの配置方法	VCL
Bottom	親コンポーネントの下端で固定。幅は親の幅に広がる。	alBottom
Center	親コンポーネントの中央に配置	—
Client	他要素を除く親領域いっぱい広がる	alClient
Contents	親領域いっぱい広がる	—
Fit	縦横比を保ったまま親いっぱい広がり親に中央に配置	—
FitLeft	縦横比を保ったまま親いっぱい広がり親の左端で固定	—
FitRight	縦横比を保ったまま親いっぱい広がり親の右端で固定	—
Horizontal	親の幅いっぱい広がる（高さは変わらない）	—
HorzCenter	横位置は親の中央となり、高さは親いっぱい広がる。（幅は変わらない）	—
Left	親コンポーネントの左端で固定。高さは親の高さに広がる	alLeft
MostBottom	親コンポーネントの最下端で固定。	—
MostLeft	親コンポーネントの最左端で固定。	—
MostRight	親コンポーネントの最右端で固定。	—
MostTop	親コンポーネントの最上端で固定。	—
Right	親コンポーネントの右端で固定。高さは親の高さに広がる	alRight
Scale	親コンポーネントのサイズ変更にあわせて、位置やサイズを変更し、相対的な位置とサイズを保つ	—
Top	親コンポーネントの上端で固定。幅は親の幅に広がる。	alTop
VertCenter	縦位置は親の中央となり、幅は親いっぱい広がる。（高さは変わらない）	—
Vertical	親の高さいっぱい広がる（幅は変わらない）	—

【FireMonkey】



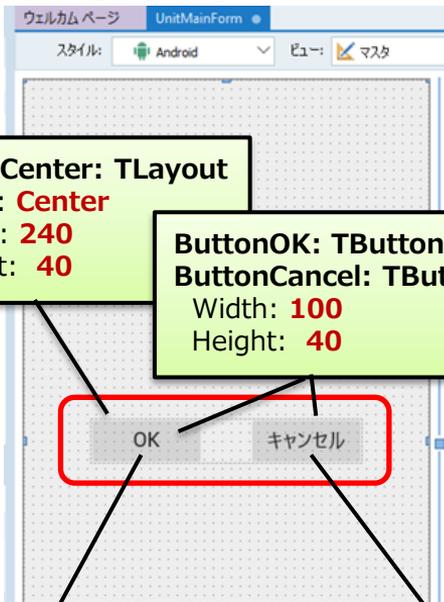
VCLでもAlign設定は可能だが、FireMonkeyの方がより多彩な配置の指定が可能

【VCL】



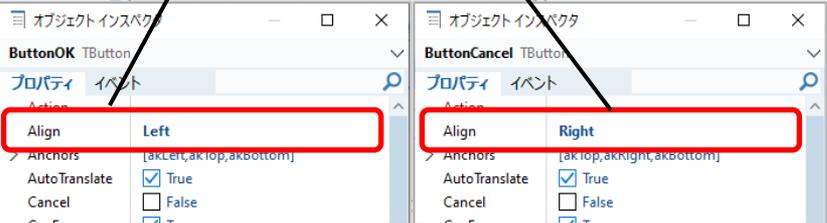
# ■ モバイルに最適なレイアウト作成方法

【フォームデザイナー】



LayoutCenter: TLayout  
Align : **Center**  
Width: **240**  
Height: **40**

ButtonOK: TButton  
ButtonCancel: TButton  
Width: **100**  
Height: **40**



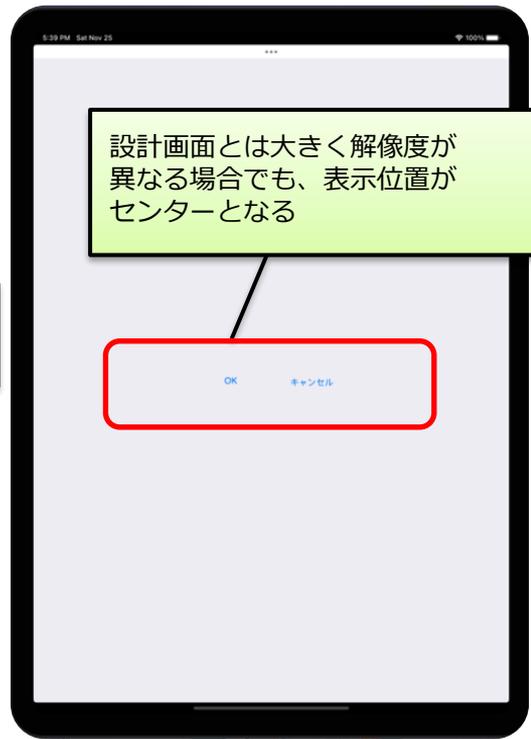
- Android  
スマートフォンで実行



スマートフォンを横表示にしても、常にセンター位置に表示される

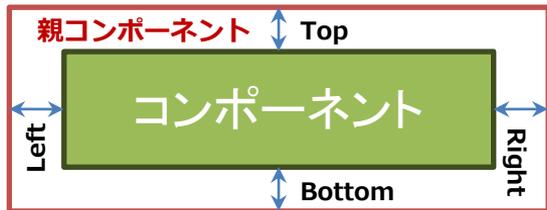
縦横比が設計画面と異なるが、センター位置に正しく表示

- iPadで実行

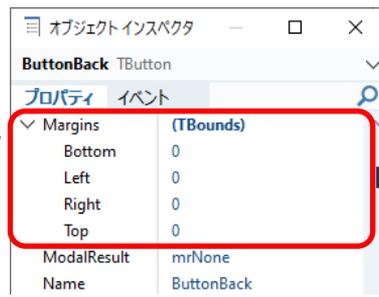


# ■ モバイルに最適なレイアウト作成方法

- Marginsプロパティ
  - コンポーネントの親コンポーネントに対する余白を設定



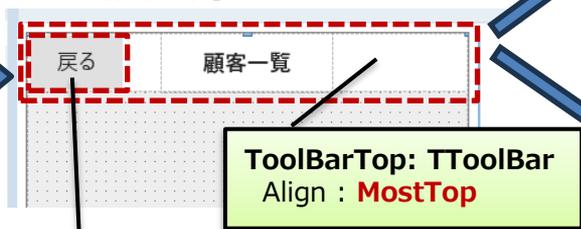
## Marginsプロパティ



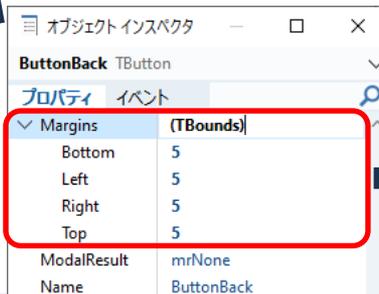
上部ツールバーの中にある  
[戻る]ボタンは、**余白無く**  
左端(Left)に配置



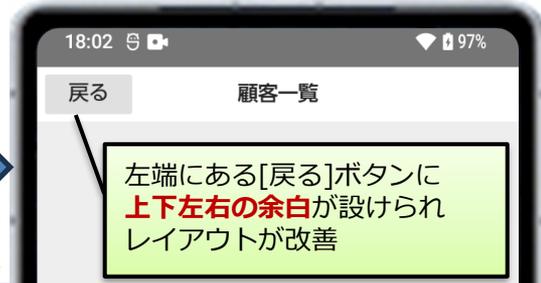
## 【フォームデザイナー】



ButtonBack: TButton  
Align : **Left**  
Width: 80



左端にある[戻る]ボタンに  
**上下左右の余白**が設けられ  
レイアウトが改善



# ■ モバイルに最適なレイアウト作成方法

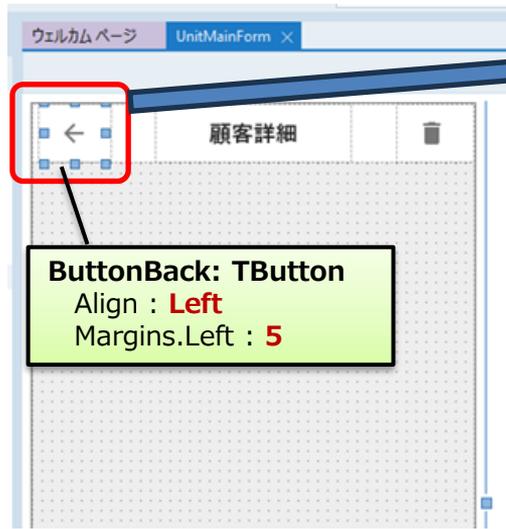
## • ボタンアイコンの使用

- 画面領域の小さいスマートフォンでは、ボタンに表題を設定する代わりにアイコンを設定する事が多い

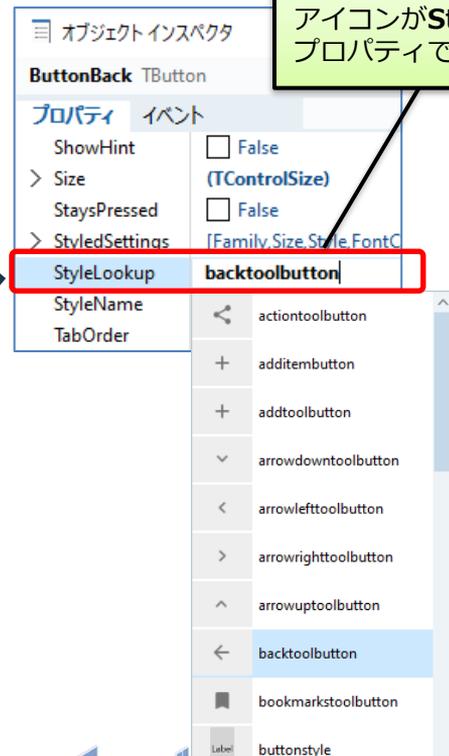


ボタンに表題を設定すると、領域を広く使ってしまふ為アイコンで表現する事が多い

【フォームデザイナー】



ButtonBack: TButton  
Align : Left  
Margins.Left : 5



ボタンで多用する機能アイコンがStyleLookupプロパティで設定可能



# ■ モバイルに最適なレイアウト作成方法

## • ボタンアイコンの使用

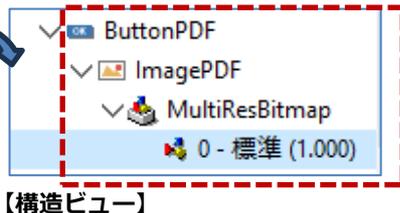
StyleLookUpで設定できない  
アイコンは独自に定義する



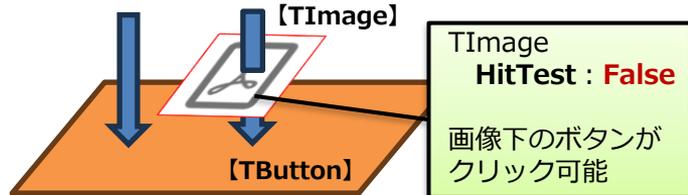
画面下部に機能ボタンを配置。複数機能がある場合、均等にボタンを配置する事が多い。

## • TImageコンポーネント

- TButtonに直接画像は設定できないが、ボタンの子として画像を追加して、アイコンが設定可能



【構造ビュー】



※ Get Itから[Icon8]をインストールすれば多彩なアイコン  
画像が無料で取得が可能

## • TGridPanelLayoutコンポーネント

- Grid上のパネルの上にコンポーネントが配置できるコンテナ

それぞれのパネル上にボタン  
コンポーネントを配置する



4列を定義し、それぞれの列幅を  
25%で設定

GridPanelLayout1.ColumnCollection の編集

メンバ	サイズの種類	値
列 0	パーセント	25.00%
列 1	パーセント	25.00%
列 2	パーセント	25.00%
列 3	パーセント	25.00%

Gridの行列数とそのサイズを  
**ColumnCollection**プロパティ  
**RowCollection**プロパティ  
で定義する

# ■ 一般的な画面構成

## • 一覧画面と詳細画面

### • PCアプリ実装例

データを一覧表示する画面は、グリッド (TDBGrid) を使用する事が多い。

[閉じる]ボタンクリックにより、前画面 (一覧) に戻る

各項目の表示には、エディット (TDBEdit) を使用する事が多い。

グリッド上の行を選択する事により、詳細を表示するフォームに画面遷移する。

著者コード	著者名
0	著者未定
1	加藤 伸一
2	上杉 則夫
3	大本 雄二
4	佐々木 麗子
5	笹原 裕子
6	小松 健一
7	松山 良一
8	小林 次郎
9	谷山 利香

著者詳細

著者コード: 3 著者名: 大本 雄二 生年月日: 1954/02/28

最終学歴: 岡山産業大学校電算科

得意分野: データベース

電話番号: 00-5252-9653 FAX 番号: 00-5252-9655

e-Mail アドレス:

住所1: 岡山県倉敷市玉島 2-5-9

### • モバイルアプリ実装例

モバイルアプリでもグリッドは使用できるが、一般的に横幅が狭い為、横スクロール操作が必要になる事が多い。

操作性を考慮した場合、リスト形式でデータを一覧表示した方が適切な事が多い。

[戻る]ボタンタッチにより、前画面 (一覧) に戻る

リスト上の行をタッチする事により、詳細を表示するフォームに画面遷移する。

モバイルアプリでもグリッドは使用できるが、一般的に横幅が狭い為、横スクロール操作が必要になる事が多い。

操作性を考慮した場合、リスト形式でデータを一覧表示した方が適切な事が多い。

取引先CD	取引先名
100001	太平洋食品株式会社
100002	株式会社岩手産業
100003	株式会社茨城電機
100004	愛知事務機株式会社
100005	埼玉商事株式会社
100006	株式会社千葉デン
100007	北海道事務機販売
100008	仙台物産株式

顧客詳細

取引先CD: 100003 取引先名: 株式会社茨城電機

取引先カナ: イチ 〇 〇 〇 〇 〇 〇

郵便番号: 310-0001 都道府県: 茨城県

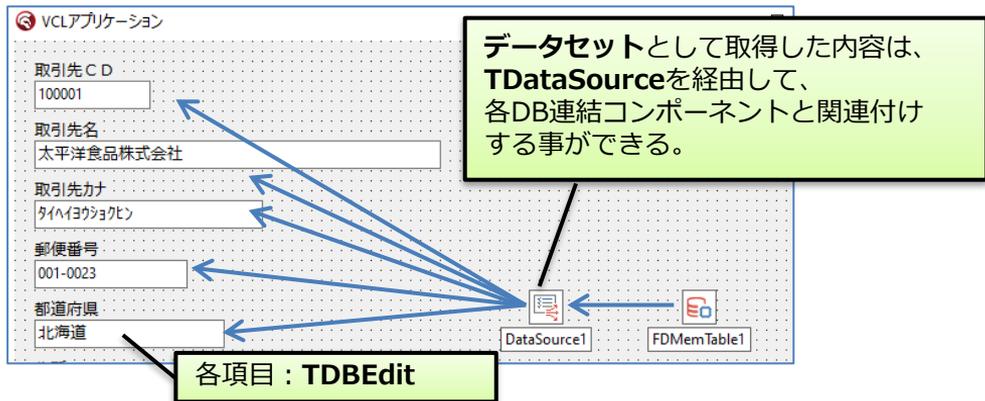
住所: 水戸市上河内町1

TEL:

# ■ データ項目の表示

## • VCL

- **データベース連結コンポーネント(TDBxxx)**を使用すれば、ノンコーディングでデータ項目の表示が可能



【主なデータベース連結コンポーネント (VCL)】

データベース連結コンポーネント	標準コンポーネント
TDBGrid	TStringGrid
TDBNavigator	-
TDBText	TLabel
TDBEdit	TEdit
TDBMemo	TMemo
TDBImage	TImage
TDBListBox	TListBox
TDBComboBox	TComboBox
TDBCheckBox	TCheckBox
TDBRadioGroup	TRadioGroup
TDBLookupListBox	TListBox
TDBLookupComboBox	TComboBox
TDBRichEdit	TRichEdit
TDBCtrlGrid	(TPanel)

## • FireMonkey

- データベース連結コンポーネントがFireMonkeyには存在しない為、標準コンポーネントを使用しなければいけない

FireMonkeyアプリで、データ項目を簡単に表示させるにはどうすればよいか？



# ■ データ項目の表示

- LiveBindingを使用
  - LiveBindingは、コンポーネント同士を関連付ける仕組み
  - 「**LiveBindingデザイナー**」を使用すれば、簡単にデータセット上の項目コンポーネントと標準コンポーネントを結び付ける事もできる。

## LiveBinding設定例

### 【フォームデザイナー】

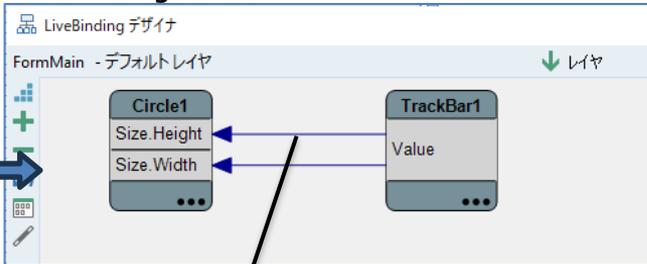
TrackBar1: TTrackBar

Min : 0  
Max: 300

Circle1: TCircle

Align : Center  
Size.Height: 50  
Size.Width : 50

### 【LiveBindingデザイナー】

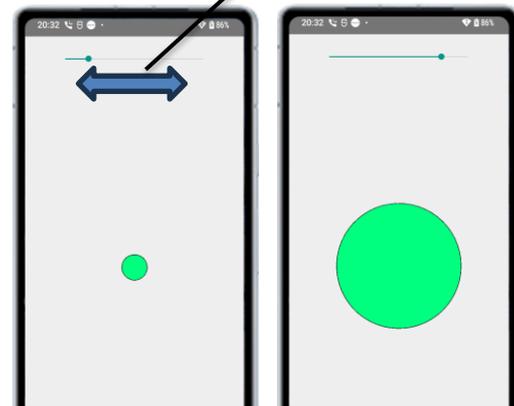


TrackBar1のValueプロパティから  
Circle1のSize.HeightとSize.Widthに  
ドラッグすると項目が関連付けされる

フォーム上で右クリックし  
「ビジュアルにバインド」を選択

トラックバーの位置 (Value) を  
変更すると連動して円の大きさ  
(Size.Height/Width)が自動的に  
変わる

### 【実行イメージ】



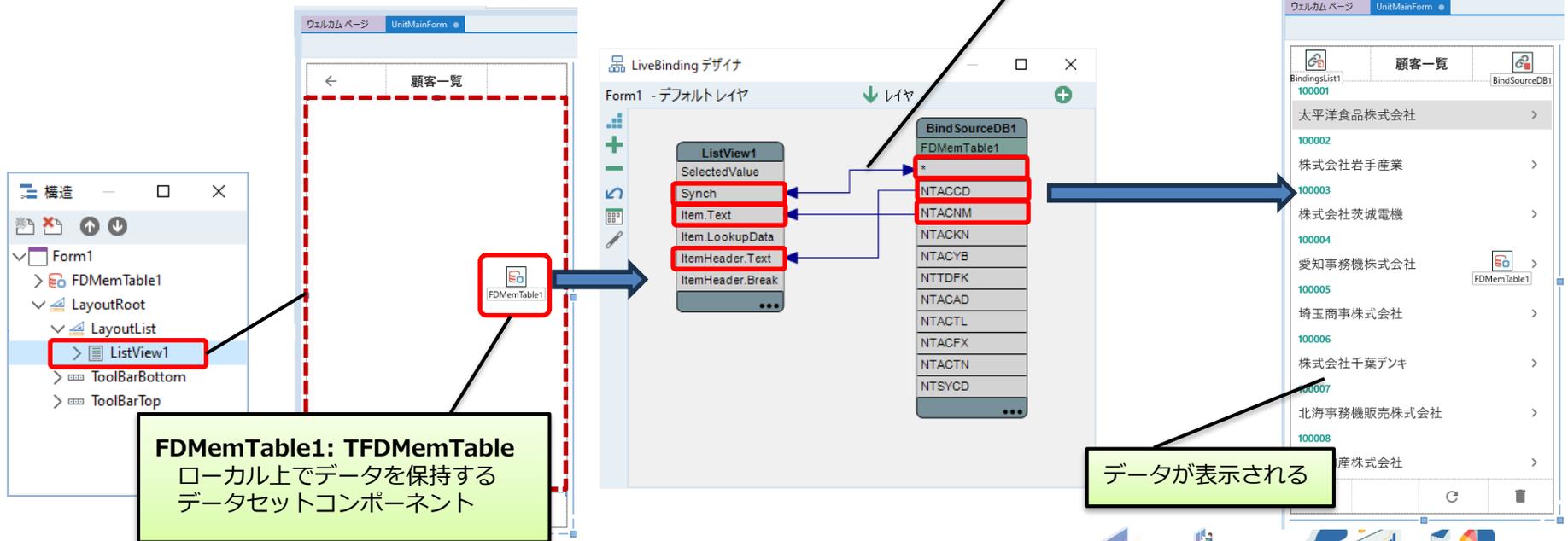
ノンコーディングで項目間の  
関連付けが可能

# ■ データ項目の表示

- 一覧画面
  - リスト表示には、**TListView**を使用
  - 見出し(**ItemHeader**)と項目(**Item**)を定義

次の項目を関連付ける

<b>ListView</b>		<b>FDMemTable1</b>
<b>Synch</b>	⇔ *	
(リスト選択とデータセットを同期)		
<b>Item.Text</b>	←	<b>NTACNM</b> (顧客名)
<b>ItemHeader.Text</b>	←	<b>NTACCD</b> (顧客名)

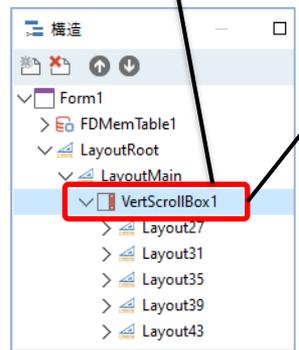


# ■ データ項目の表示

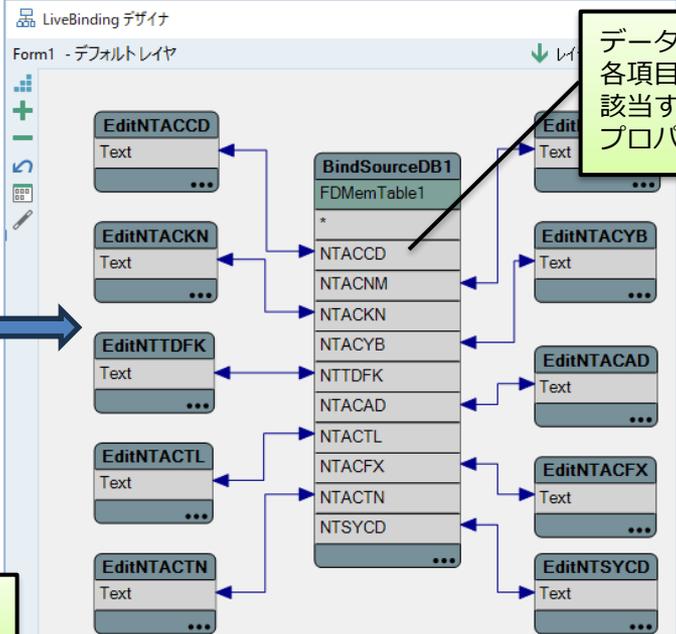
## ● 詳細画面

- フォーム上の各コンポーネントも**TLayout**を使用して相対的な位置で配置
- データ部分には、**TVertScrollBar**を配置。画面に収まらない場合縦スクロールが可能

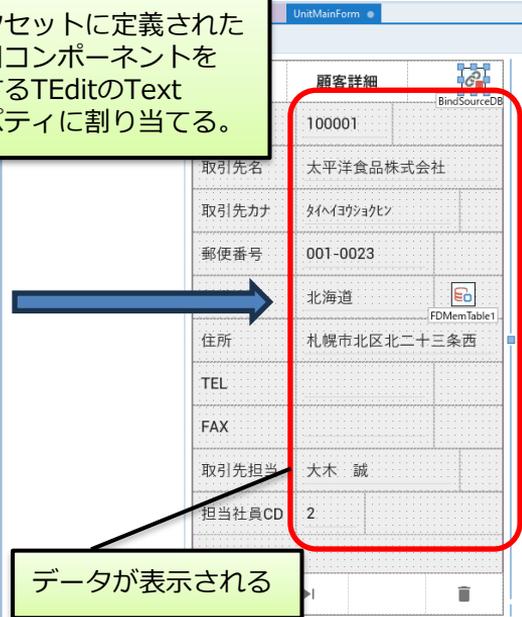
デバイスを横にした場合等に画面に項目が収まらなくてもスクロール可能となる。



TEditを配置  
ReadOnly: True



データセットに定義された各項目コンポーネントを該当するTEditのTextプロパティに割り当てる。



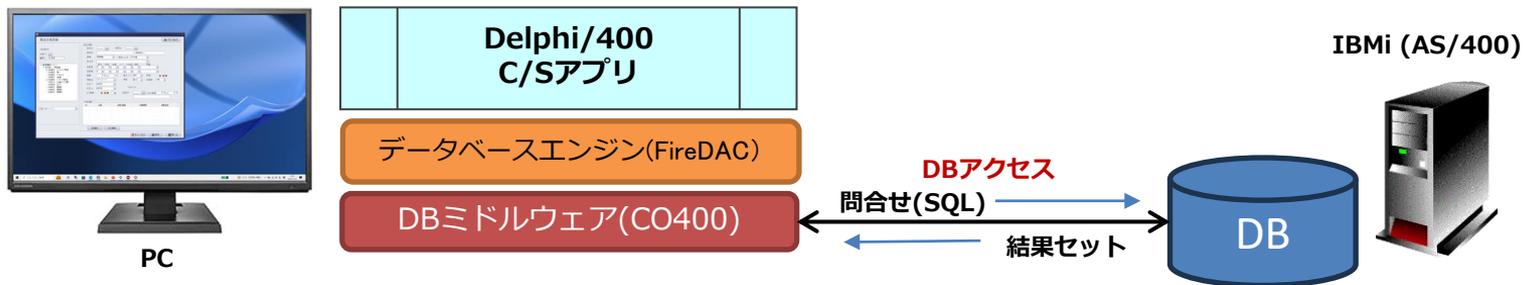
データが表示される

# IBM i DB情報を活用する モバイルアプリ開発手法

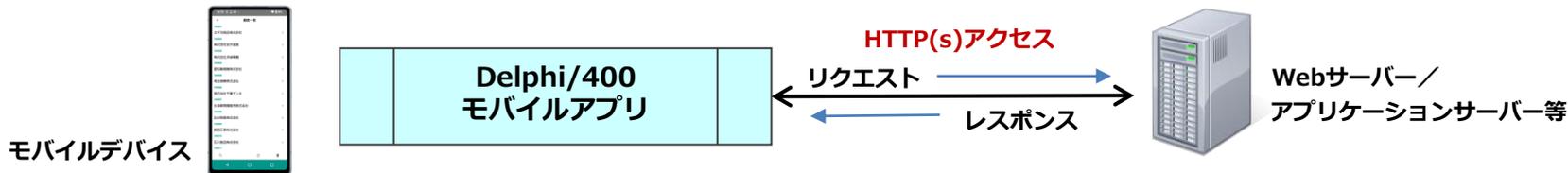


# ■ PCアプリとモバイルアプリのアクセス手法

- PCアプリ(Windows)の場合、データベースエンジン(FireDAC等)とDBミドルウェアを使用して、直接データベースサーバーと通信する事が多い(所謂C/S形式)



- モバイルアプリ (iOS/Android) は、OSの制約上DBミドルウェアを直接組み込めない
  - 外部との通信は、HTTP(s)等を使用したネットワーク通信に限られる



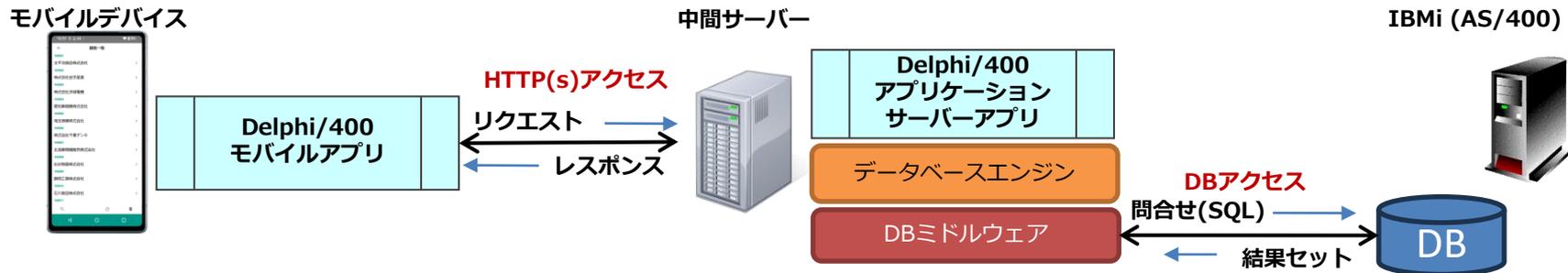
モバイルアプリでDB情報にアクセスするには、どうすればよいか？



# ■ 3層アプリ形式によるDBアクセス

## • 3層アプリ形式で開発を行う

- クライアントからのリクエストを元にアプリケーションサーバーがデータを抽出するSQLを発行
- IBM iから取得したデータセットを元にレスポンスを作成し、クライアントに結果を返す



- Delphi/400では、従来3層アプリを**DataSnap**フレームワークで開発する事が多かった
  - Delphiクライアントアプリとの通信であれば、dbExpressやFireDACを使用して容易に作成できる
  - 基本的なサーバー機能以外は原則全て独自に開発しなければいけない
  - 過去のテクニカルセミナーで作成手順はご紹介済み

[https://www.migaro.co.jp/contents/support/technical\\_seminar\\_search/14th/Session2.pdf](https://www.migaro.co.jp/contents/support/technical_seminar_search/14th/Session2.pdf)

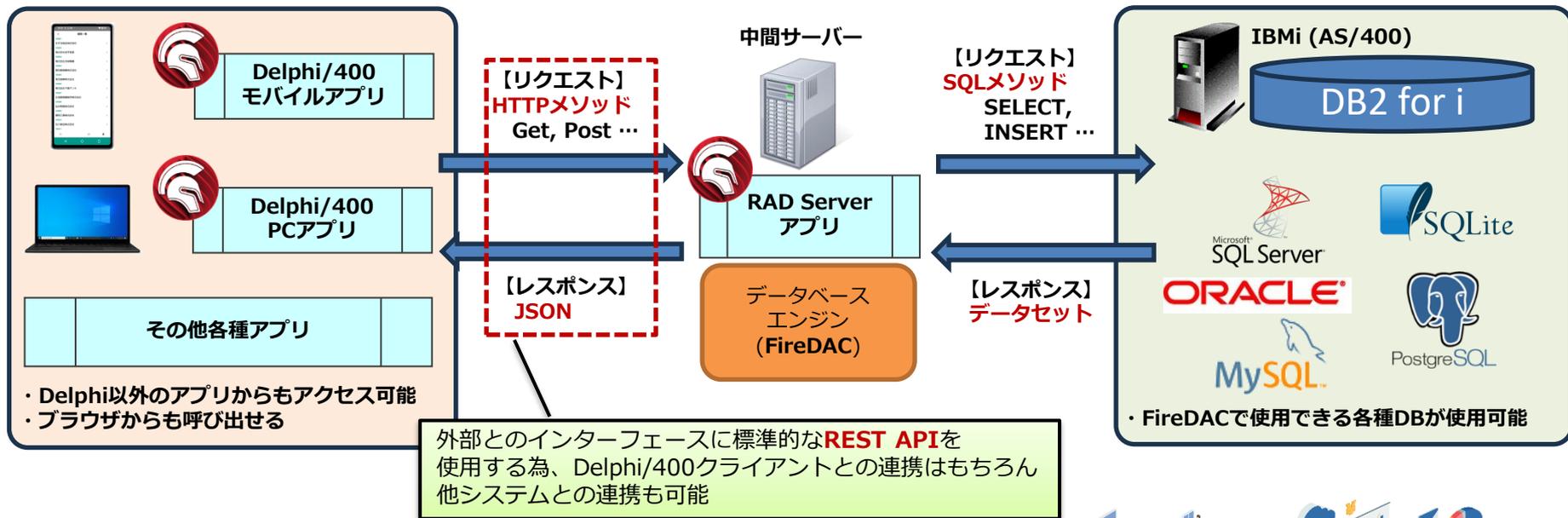
- 現在は新しい**RAD Server**フレームワークを推奨



# ■ RAD Serverとは

## • RAD Server

- Web技術を活用した外部公開用APIサーバーアプリを構築する為のフレームワーク (Web API)
- 業界標準となっているREST+JSON方式 (**REST API**)を採用 (DataSnapでは独自プロトコル)
- アクセス管理機能や分析機能等が標準機能として組み込まれている (DataSnapでは独自実装が必要)



# REST APIの基本

## REST APIにおけるリクエスト

- REST APIサーバーへのリクエストにはHTTPメソッドを使用する
  - ネットワーク上のデータ（リソース）への**アクセスを一意的URI**で指定
- REST APIサーバーは、メソッドの種類にあわせて、データベースに対し適切なSQLを実行する

HTTPメソッド	URI	意味	対応するSQL
GET	<code>/ {resource} /</code> <code>/ {resource} / {item}</code>	データ（リソース）情報を取得 データ（リソース）中の特定の項目（アイテム）の情報を取得	<code>SELECT * FROM {resource}</code> <code>SELECT * FROM {resource} WHERE key = {item}</code>
POST	<code>/ {resource} /</code>	データ（リソース）に新しい項目（アイテム）を追加	<code>INSERT INTO {resource} VALUES [data]</code>
PUT	<code>/ {resource} / {item}</code>	データ（リソース）中の特定の項目（アイテム）の情報を更新	<code>UPDATE {resource} SET [data] WHERE key = {item}</code>
DELETE	<code>/ {resource} / {item}</code>	データ（リソース）中の特定の項目（アイテム）の情報を削除	<code>DELETE FROM {resource} WHERE key = {item}</code>

【処理イメージ】

クライアント



<GETメソッドでリクエスト>

(URL) `http://TestServer/mprodp/C-10001`

REST APIサーバーの仕様  
商品情報を取得する場合、  
リソース名に**mpdrop**を指定  
アイテム名には、**[商品CD]**を指定

REST APIサーバー  
(TestServer)



ファイルレイアウト

PDPDCD (キー)	商品CD
PDPDNM	商品名
PDUNPR	販売単価
PDSTDT	発売日

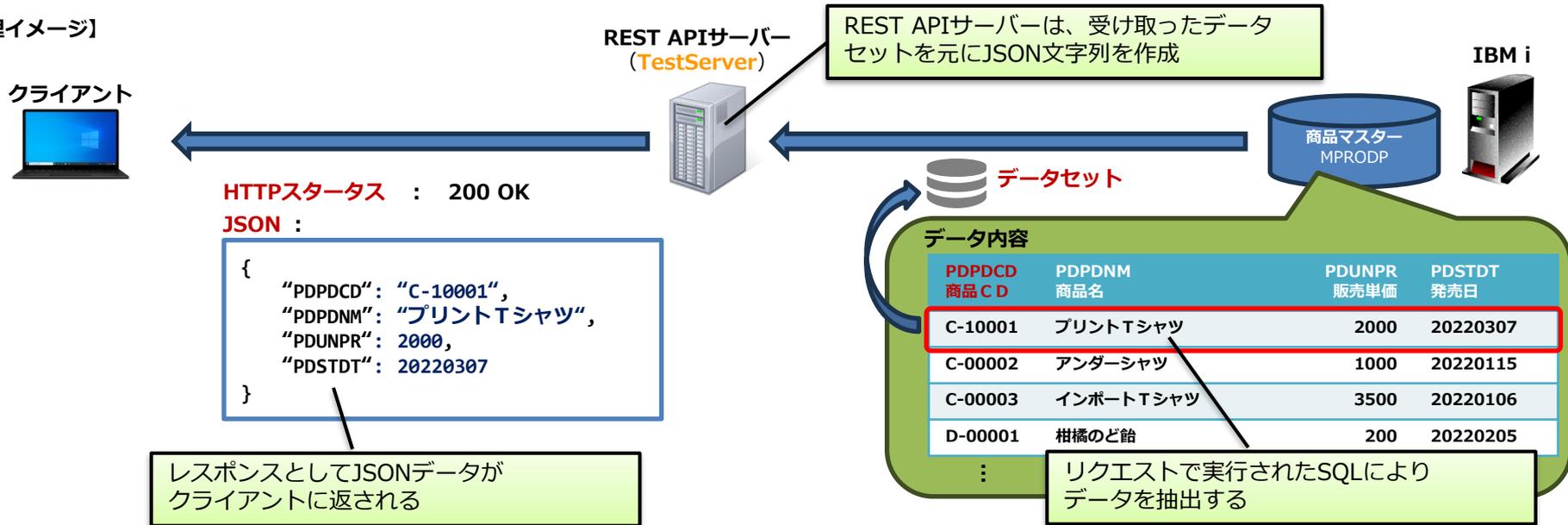
<対応するSQLを実行>

```
SELECT * FROM MPROD  
WHERE PDPDCD = 'C-10001'
```

# REST APIの基本

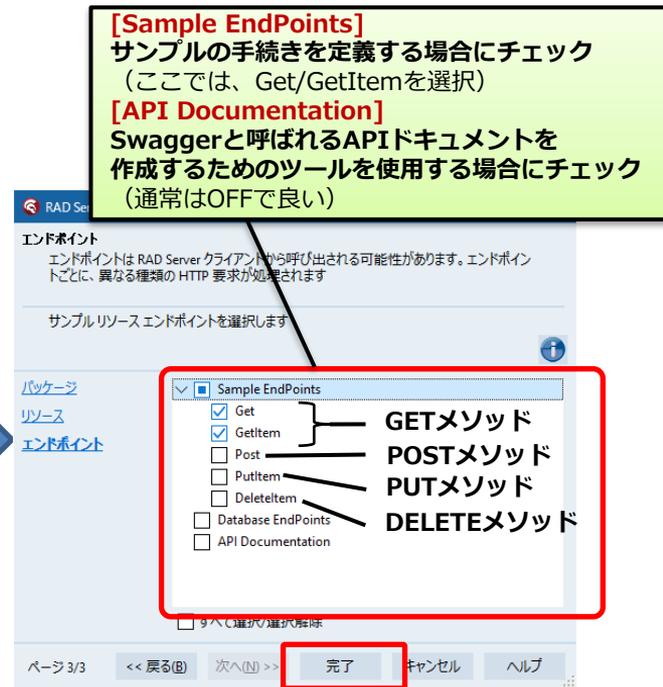
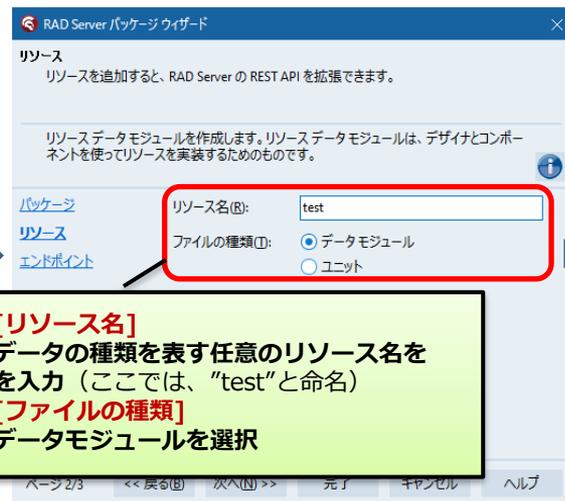
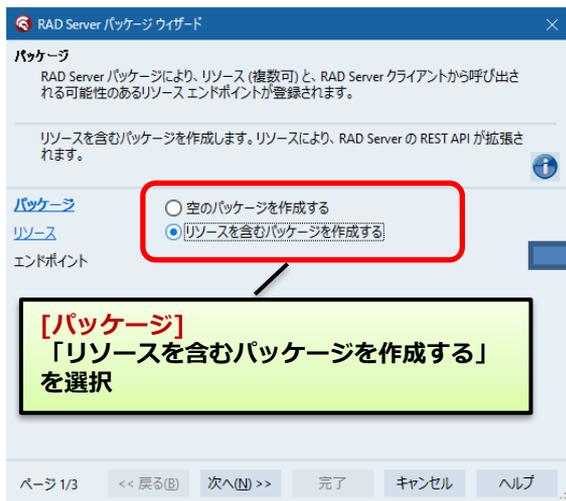
- REST APIにおけるレスポンス
  - データベースから受け取った情報をJSON形式に変換してクライアントに返却する
  - 処理結果をHTTPステータスコードで返す（正常終了の場合、200:OKを返す）

【処理イメージ】



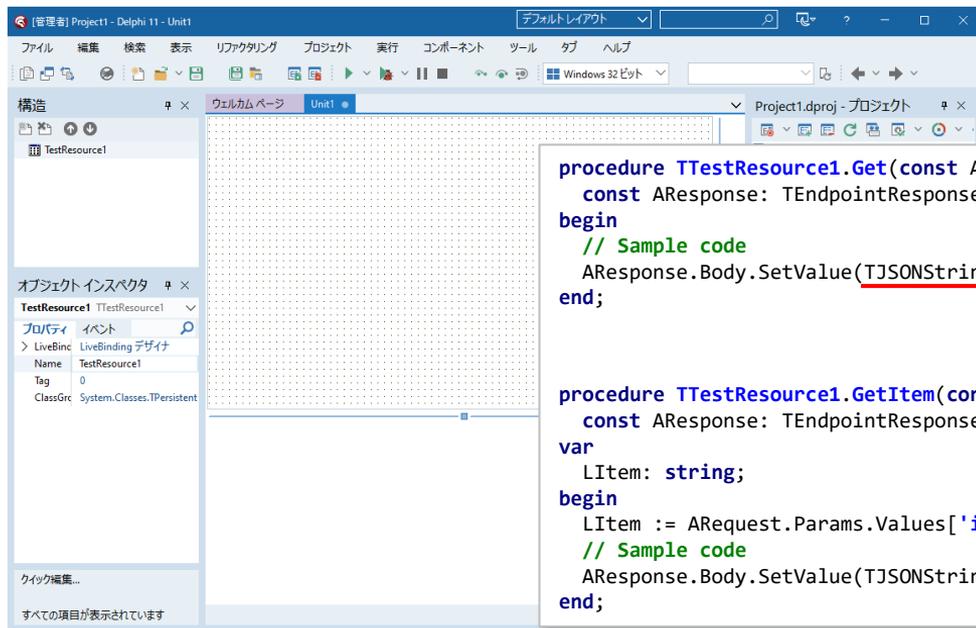
# ■ RAD Serverアプリ開発手順

- RAD Server プロジェクトの新規作成
  - [ファイル]→[新規作成]→[その他]→[RAD Server] →[RAD Server パッケージ]
  - ウィザードに従って初期設定すれば良い



# ■ RAD Serverアプリ開発手順

- RAD Server プロジェクトの新規作成
  - Unit1.pasに、TDataModuleを継承してできたTTestResourceクラスのデータモジュールが生成
  - コード実装部には、ウィザードで選択した**Get/GetItem**手続きのサンプルコードが自動生成



【ソースコード：実装部】

```
procedure TTestResource1.Get(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
    const AResponse: TEndpointResponse);
begin
    // Sample code
    AResponse.Body.SetValue(TJJSONString.Create('test'), True);
end;
```

レスポンスとして、  
『test』というJSON文字列をセットする  
サンプルコード

```
procedure TTestResource1.GetItem(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
    const AResponse: TEndpointResponse);
var
    LItem: string;
begin
    LItem := ARequest.Params.Values['item'];
    // Sample code
    AResponse.Body.SetValue(TJJSONString.Create('test ' + LItem), True);
end;
```

レスポンスとして、  
『test {item}』というJSON文字列を  
セットするサンプルコード

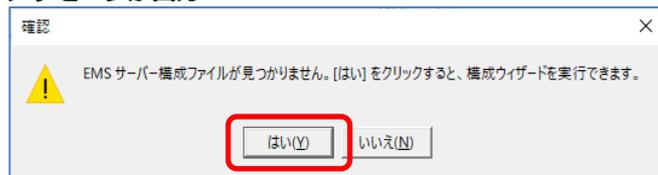


# ■ RAD Serverアプリ開発手順

- RAD Server プロジェクトの実行
  - 開発環境からであれば、[実行]ボタンで開発用サーバーを起動可能
    - RAD Serverの実行には、InterBaseが必要となる為、予めサービスの起動を確認しておく。（InterBaseサーバマネージャ）
    - 開発環境で初めてRAD Server開発用サーバーを起動しようとする時、[EMSサーバ構成ファイルが見つからない]というメッセージが出力される。構成ウィザードを進めれば、サーバ構成が作成可能。（通常はサーバインスタンス名以外は初期値で作成可能）

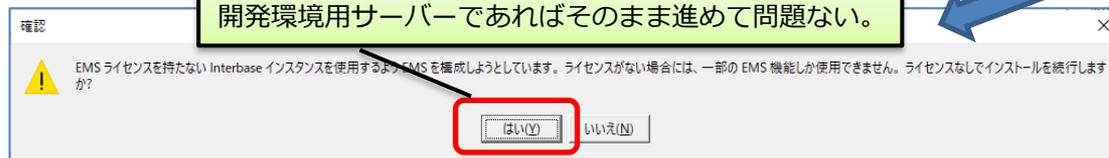


## メッセージが出力



構成ウィザードを開始

途中ライセンスが無いメッセージが出力されるが、開発環境用サーバであればそのまま進めて問題ない。



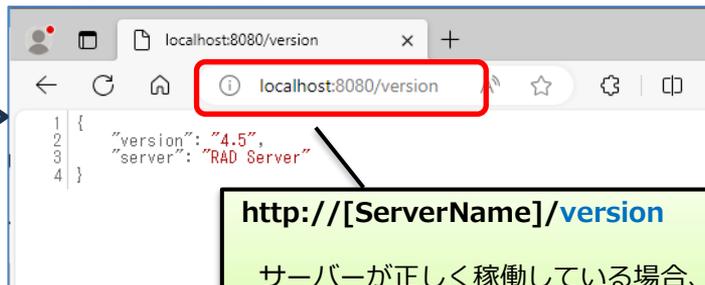
# ■ RAD Serverアプリ開発手順

- RAD Server プロジェクトの実行
  - 開発用サーバーは、**8080**ポートで動作
  - [ブラウザを開く]で動作確認が可能

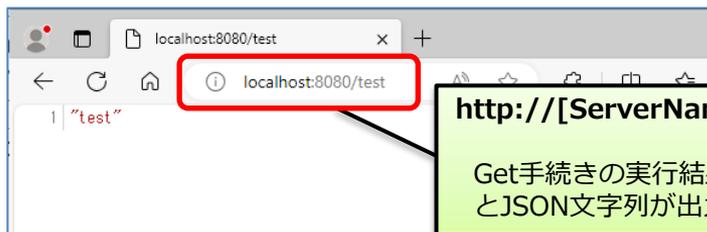
【開発環境用サーバー】



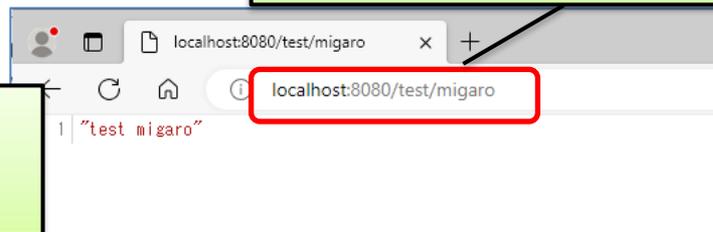
【実行結果 : ブラウザ】



リクエスト (GETメソッド)  
Get手続きが実行

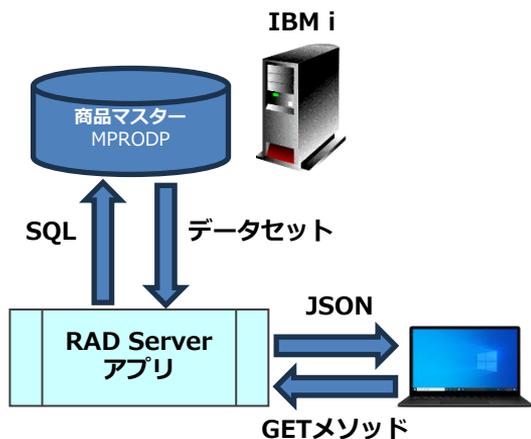


リクエスト (GETメソッド)  
GetItem手続きが実行



# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- 商品マスタ(MPRODP) へアクセスするRAD Serverアプリ
  - HTTPメソッド(GET) を使用して商品マスタの情報を取得する



[http://\[ServerName\]/test/mprod/](http://[ServerName]/test/mprod/)

Get手続きの実行結果として  
商品マスタのデータ一覧がJSON文字列で出力

```
localhost:8080/test/mprod/
localhost:8080/test/mprod/

1 [
2 {
3   "PDELFL": "",
4   "PDPDCD": "C-00001",
5   "PDPDNM": "プリントTシャツ",
6   "PDJNCD": "4924027222510",
7   "PDSPCD": "S00001",
8   "PDCATG": "06",
9   "PDUNPR": 2000,
10  "PDCOPR": 1200,
11  "PDSTDT": 20220307,
12  "PDENFG": "",
13  "PDUPDT": 20230801,
14  "PDUPUS": "OZAKI",
15 }
16 ]
17
18 {
19   "PDELFL": "",
20   "PDPDCD": "C-00002",
21   "PDPDNM": "アンダーシャツ",
22   "PDJNCD": "4953260902737",
23   "PDSPCD": "S00002",
24   "PDCATG": "06",
25   "PDUNPR": 1000,
26   "PDCOPR": 450,
27   "PDSTDT": 20220115,
28   "PDENFG": "1",
29   "PDUPDT": 20230801,
30   "PDUPUS": "OZAKI",
31 }
32
33 {
34   "PDELFL": "",
35   "PDPDCD": "C-00003",
36   "PDPDNM": "インポートTシャツ",
37   "PDJNCD": "4934889471114",
38   "PDSPCD": "S00001",
39   "PDCATG": "06",
40   "PDUNPR": 2500,
```

[http://\[ServerName\]/test/mprod/C-00001](http://[ServerName]/test/mprod/C-00001)

GetItem手続きの実行結果として  
指定した商品CDの商品マスタのデータが  
JSON文字列で出力

[localhost:8080/test/mprod/C-00001](http://localhost:8080/test/mprod/C-00001)

```
localhost:8080/test/mprod/C-00001
localhost:8080/test/mprod/C-00001

1 {
2   "PDELFL": "",
3   "PDPDCD": "C-00001",
4   "PDPDNM": "プリントTシャツ",
5   "PDJNCD": "4924027222510",
6   "PDSPCD": "S00001",
7   "PDCATG": "06",
8   "PDUNPR": 2000,
9   "PDCOPR": 1200,
10  "PDSTDT": 20220307,
11  "PDENFG": "",
12  "PDUPDT": 20230801,
13  "PDUPUS": "OZAKI",
14 }
```

【実行結果 : ブラウザ】

# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- FireDACデータアクセス処理の追加
  - コネクション(TFDConnection)及びデータセット(TFDQuery) コンポーネントを追加
  - **TEMSDatasetResource**を配置

**FDConnection1: TFDConnction**  
LoginPrompt : **False**

**FDQuery1: TFDQuery**

**EMSDatasetResource1: TEMSDatasetResource**

**TFDQuery.SQLプロパティ**  
`SELECT * FROM [ファイル名]  
{IF %SORT} ORDER BY &SORT {FI}`

**TFDQueryのクエリエディタ**  
FireDAC クエリ エディタ - [FDQuery1]  
クエリの SQL、パラメータ、マクロ、オプションをセットアップします

SQL コマンド  
1 `SELECT * FROM MPRODP`  
2 `{IF &SORT} ORDER BY &SORT {FI}`

実行(E)

データが正しく取得できるか  
テスト実行が可能

1行目は取得したいデータの**ファイル名**を記述した**SELECT**句とする。  
2行目は、EMSDatasetResourceの機能で任意の項目でソートが指定できるようにする為のものなので、そのまま記述する。

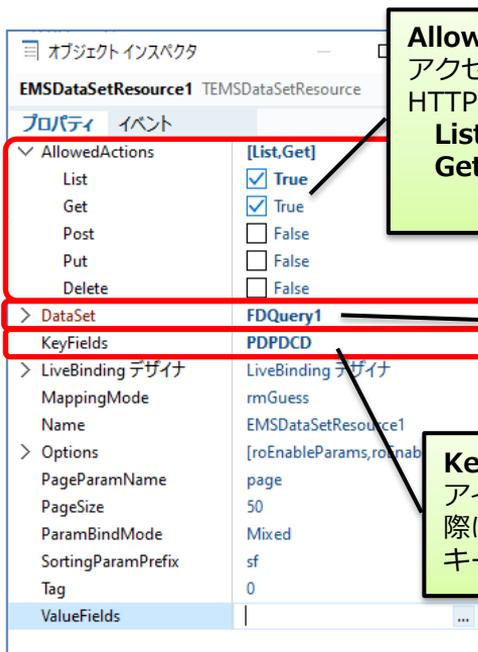
FireDACリンクコンポーネントを配置  
IBM i の場合は、**TFDPhysCO400DriverLink**を使用

【フォームデザイナー】

# ■ IBM i データにアクセスするRAD Serverアプリ(1)

## • TEMSDataSetResource

- 受け取ったHTTPメソッドからSQLを実行して、レスポンスのJSON文字列を返すコンポーネント



**AllowedActions**プロパティ  
アクセスを許可する  
HTTPメソッドを指定  
**List** : 一覧データを返す  
**Get** : 指定されたアイテム  
のデータを返す

**DataSet**プロパティ  
SQLを実行する  
TFDQueryを指定

**KeyFields**プロパティ  
アイテムが指定された  
際に、データを抽出する  
キーフィールドを指定

定義した**EMSDatasetResource1**の宣言部の上に、**[ResourceSuffix('mprodp')]** を行追加する。  
リソース名とターゲットとなる**TEMSDataSetResource**  
コンポーネントとが関連付く

### TTestResource1クラスの宣言部

```
type
[ResourceName('test')]
TTestResource1 = class(TDataModule)
    FDConnection1: TFDConnection;
    FDPhysC0400DriverLink1: TFDPhysC0400DriverLink;
    FDQuery1: TFDQuery;
    [ResourceSuffix('mprodp')]
    EMSDataSetResource1: TEMSDataSetResource;
published
    procedure Get(const AContext: TEndpointContext; const ARequest:
        TEndpointRequest; const AResponse: TEndpointResponse);
    [ResourceSuffix('{item}')]
    procedure GetItem(const AContext: TEndpointContext; const ARequest:
        TEndpointRequest; const AResponse: TEndpointResponse);
end;
```



# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- モバイルアプリからRAD Serverへのアクセス方法
  - RESTコンポーネントを使用して作成
    - TRestClient / TRestRequest / TRestResponse を設定すれば良い。
    - 『RESTデバッガ』を使用すれば容易に作成可能。JSONデータをローカルデータセットに変換するコンポーネントも同時に作成できる。

The screenshot shows the REST Debugger interface with several callouts explaining the configuration and response.

**RESTデバッガ**

REST デバッガ 10.5  
Embarcadero Technologies

要求

要求 パラメータ 認証 接続

メソッド: GET URL: http://192.168.0.89:8080

コンテンツタイプ: application/json

カスタム本体:

応答

**GETメソッドを使用  
受け取るコンテンツタイプは、JSON**

http://[ServerName]/test/mprod/

URL リソース

**[本体]タブ  
JSON文字列を確認できる**

**[要求の送信]  
リクエストのテストができる**

要求の送信

**RESTデバッガ (応答部)**

応答

http://192.168.0.89:8080/test/mprod/  
200: OK - 返されたデータは 15541 バイト。計時情報: 前処理: 0 ミリ秒 - 実行: 0 ミリ秒

ヘッダー 本体 表データ

内容が有効な JSON です JSON ルート要素:

```
{
  "PDPDCD": "C-00001",
  "PDPDNM": "プリントシャツ",
  "PDJNCD": "4924027222510",
  "PDSPCD": "S00001",
  "PDCATG": "06",
  "PDUNPR": 2000,
  "PDCOPR": 1200,
  "PDSTDT": 20220307,
  "PDENFG": "",
  "PDUPDT": 20230801,
  "PDUPLUS": "OZAKI"
},
{
  "PDPDCD": "C-00002",
  "PDPDNM": "アンダーシャツ",
  "PDJNCD": "4953260902737",
  "PDSPCD": "S00002"
}
```

プロキシサーバーが無効です

# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- モバイルアプリからRAD Serverへのアクセス方法
  - RESTデバッガから取得したコンポーネント定義を貼り付け

**RESTデバッガ (応答部)**

メソッド: GET URL: http://192.168.0.89:8080

**[表データ]タブ**  
JSON文字列を一覧形式で確認できる

**[コンポーネントのコピー]**  
[表データ]タブを開いた状態で押下すると、JSON文字列をデータセットに変換するコンポーネントもコピーされる

**[使用型]**  
『文字列』を選択して[適用]をクリック

コンポーネントのコピー

文字列 適用

**FireMonkey モバイルアプリプロジェクト**

ListView1: TListView

編集(M) 元に戻す(U) Ctrl+Z  
コントロール(O) 切り取り(D) Ctrl+X  
ビジュアルにバインド(Q)... コレ(C) Ctrl+C  
クイック編集(S)... 貼り付け(E) Ctrl+V  
編集モードの切り替え(I) 削除(D) Ctrl+D  
デザイン モードの切り替え(U) すべて選択(L) Ctrl+A

コピーしたコンポーネントを貼り付け

PDPDCD	PDPDNM	PDJNCD	PDSPCD	PDCATG	PDUNPR	PDCOPR	PDSDD1	PDENPG
C-00001	プリントTシャツ	4924027222510	S00001	06	2000	1200	20220307	
C-00002	アンダーシャツ	4953260902737	S00002	06	1000	450	20220115	1
C-00003	インポートTシャツ	4934989471114	S00001	06	3500	2000	20220106	
D-00001	柑橘のど飴	4987920773949	S00002	04	200	120	20220205	
D-00002	フルーツのど飴	4935331278420	S00001	04	230	140	20220912	

# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- モバイルアプリからRAD Serverへのアクセス方法
  - LiveBindingの設定
  - 実行イベントの定義

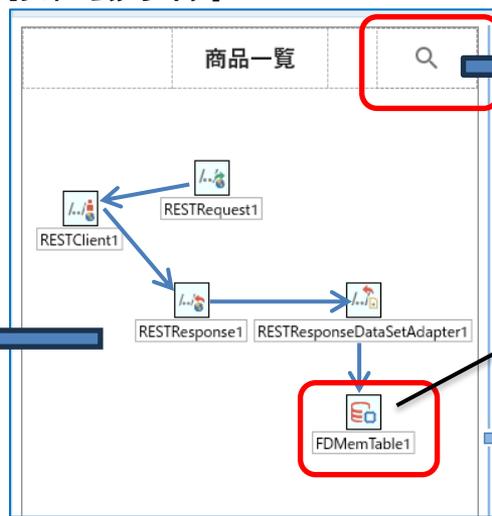
## 【LiveBindingデザイナー】

LiveBinding デザイン

FormMain - デフォルトレイヤ

Property	Value
RESTClient1.BaseURL	...
RESTResponse1.Content	...
RESTResponse1.ContentType	...
RESTResponse1.ContentLength	...
RESTResponse1.ContentEncoding	...
RESTRequest1.Resource	...
RESTRequest1.ResourceSuffix	...
ButtonSearch.Text	...
BindSourceDB1.FDMemTable1	...
BindSourceDB1.PDPDCD	...
BindSourceDB1.PDPDNM	...
BindSourceDB1.PDJNCD	...
BindSourceDB1.PDSPCD	...
BindSourceDB1.PDCATG	...
BindSourceDB1.PDUNPR	...
BindSourceDB1.PDCOPR	...
BindSourceDB1.PDSTDT	...
BindSourceDB1.PDENFG	...
BindSourceDB1.PDUPDT	...
BindSourceDB1.PDUPUS	...
BindSourceDB1.JSONValue	...
BindSourceDB1.JSONText	...
ListView1.SelectedValue	Synch
ListView1.Content	Item.Text
ListView1.ContentLength	Item.LookupData
ListView1.ContentEncoding	ItemHeader.Text

## 【フォームデザイナー】



**FDMemTable1: TFDMemTable**  
取得したJSONデータを受け取る  
ローカルデータセット

## 【ButtonSearchのOnClickイベント】

```
procedure TFormMain.ButtonSearchClick(Sender: TObject);
begin
    RESTRequest1.Execute;
end;
```



# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- モバイルアプリからRAD Serverへのアクセス方法
  - 完成したモバイルアプリの実行

※ Androidの場合、デフォルト設定では、https以外の通信トラフィックが禁止されている為、マニフェストファイルにて事前に許可を与える必要がある。

[検索]ボタンをタップすると商品一覧が表示される

商品一覧

16:44 100%

商品一覧

検索

16:45 100%

商品一覧

C-00001  
プリントTシャツ  
C-00002  
アンダーシャツ  
-00003  
インポートTシャツ  
D-00001  
柑橘のど飴  
D-00002  
フルーツのど飴  
J-10001  
クリアファイル  
J-10002  
フロッピーディスクケース  
J-10003  
ワープロ用感熱紙  
J-10004  
バインダー  
J-10005  
コンピュータ用紙  
J-10006

AndroidManifest.template.xml

Projects > TEC2023 > Mobile app 6 butt

名前

Android64

AndroidManifest.template.xml

MobileApp\_06.deployproj

MobileApp\_06.dpr

MobileApp\_06.dproj

MobileApp\_06.dproj.local

MobileApp\_06.identcache

[AndroidManifest.template.xml]

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3
4 <application>タグの中に以下の設定を追加する
5 android:usesCleartextTraffic="true"
6
7
8 <uses-sdk android:minSdkVersion="%minSdkVersion%" android:
9 <%uses-permission%>
10 <uses-feature android:glEsVersion="0x00020000" android:rec
11 <queries>
12 <%queries-child-elements%>
13 </queries>
14 <application>
15 android:persistent="%persistent%"
16 android:restoreAnyVersion="%restoreAnyVersion%"
17 android:label="%label%"
18 android:debuggable="%debuggable%"
19 android:largeHeap="%largeHeap%"
20 android:icon="%icon%"
21 android:theme="%theme%"
22 android:hardwareAccelerated="%hardwareAccelerated%"
23 android:resizableActivity="false"
24 android:requestLegacyExternalStorage="true"
25 android:usesCleartextTraffic="true"
26
27 provider%>
28 </application meta-data%>
```

プロジェクトフォルダ内にあるマニフェストファイルをテキストエディタ等で編集する

# ■ IBM i データにアクセスするRAD Serverアプリ(1)

- (補足)オプションパラメータについて
  - TEMSDataSetResourceを使用した場合、データ取得時に以下のオプションパラメータが使用可能

## 【パラメーター一覧】

パラメータ	概要	記述例
page	指定したページのデータのみ取得 (1ページ当りの取得レコード数は、PageSizeプロパティで設定可能。初期値は50)	?page=1 (1ページ目(1~50件目)のみ取得)
sf	指定したフィールドで昇順/降順にソート A: 昇順 D: 降順	?sfPDPDNM=A (PDPDNMの昇順に取得)

## 【リクエストURL例】

http://[ServerName]/test/mprodp/?page=1&sfPDPDNM=A

取得ページ番号

ソート指定

## 【実行結果 : ブラウザ】

```
1 [
2   {
3     "PDPDCD": "0-10003",
4     "PDPDNM": "A V サラウンドテレビ 2 6",
5     "PDJNCD": "4918738364818",
6     "PDSPCD": "S00002",
7     "PDCATG": "02",
8     "PDUNPR": 212000,
9     "PDCOPR": 88000,
10    "PDSSTD": 20230203,
11    "PDENFG": "1",
12    "PDUPDT": 20230801,
13    "PDUPLS": "OZAKI"
14  },
15  },
16  {
17    "PDPDCD": "0-10002",
18    "PDPDNM": "A V サラウンドテレビ 2 9",
19    "PDJNCD": "4960393078402",
20    "PDSPCD": "S00002",
21    "PDCATG": "02",
22    "PDUNPR": 212000,
23    "PDCOPR": 88000,
24    "PDSSTD": 20230203,
25    "PDENFG": "1",
26    "PDUPDT": 20230801,
27    "PDUPLS": "OZAKI"
28  },
29  },
30  {
31    "PDPDCD": "0-10001",
32    "PDPDNM": "A V サラウンドテレビ 3 8",
33    "PDJNCD": "4999813837092",
34    "PDSPCD": "S00002",
35    "PDCATG": "02",
36    "PDUNPR": 212000,
37    "PDCOPR": 88000,
38    "PDSSTD": 20230203,
39    "PDENFG": "1",
40    "PDUPDT": 20230801,
41    "PDUPLS": "OZAKI"
42  },
43  },
44  ]
```

取得されたJSONデータは、  
1~50件目までの50件のデータで、  
商品名 (PDPDNM) の昇順にソート  
されている

# ■ IBM i データにアクセスするRAD Serverアプリ(2)

- 独自の検索条件を指定して合致するデータを取得するRAD Serverアプリ

【リクエストURL例】

`http://[ServerName]/test/mprodp_opt?catg=02&pdnm=冷蔵庫`

URL

リソース

商品カテゴリ

品名 (部分一致)

【実行結果 : ブラウザ】

- TEMSDatasetResourceでは、  
上記のように独自のパラメータによる  
絞込条件 (SQLにおけるWHERE句) を  
定義する事ができない  
↓
- 受け取ったパラメータを用いて、  
パラメータ付クエリーを実行し、取得した  
データをJSONに変換してレスポンスをセット  
する処理を実装する必要がある

JSON文字列を簡単に  
作成する方法はないか？

```
1 [
2 {
3   "PDPDCD": "K-10003",
4   "PDPDNM": "4 ドア 冷蔵庫レッド",
5   "PDJNCD": "4989856699301",
6   "PDSPCD": "S00002",
7   "PDCATG": "01",
8   "PDUNPR": 220000,
9   "PDCOPR": 85000,
10  "PDSTDT": 20221211,
11  "PDEFNG": "",
12  "PDUPDT": 20230801,
13  "PDPUS": "OZAKI"
14 },
15 {
16   "PDPDCD": "K-10004",
17   "PDPDNM": "4 ドア 冷蔵庫ブラック",
18   "PDJNCD": "4945194860883",
19   "PDSPCD": "S00003",
20   "PDCATG": "01",
21   "PDUNPR": 240000,
22   "PDCOPR": 100000,
23   "PDSTDT": 20220524,
24   "PDEFNG": "",
25   "PDUPDT": 20230801,
26   "PDPUS": "OZAKI"
27 },
28 ]
29
30
```

以下の条件に合致するデータのみ取得  
PDCATG(カテゴリーCD) = "01" &  
PDPDNM(商品名)に"冷蔵庫"を含む

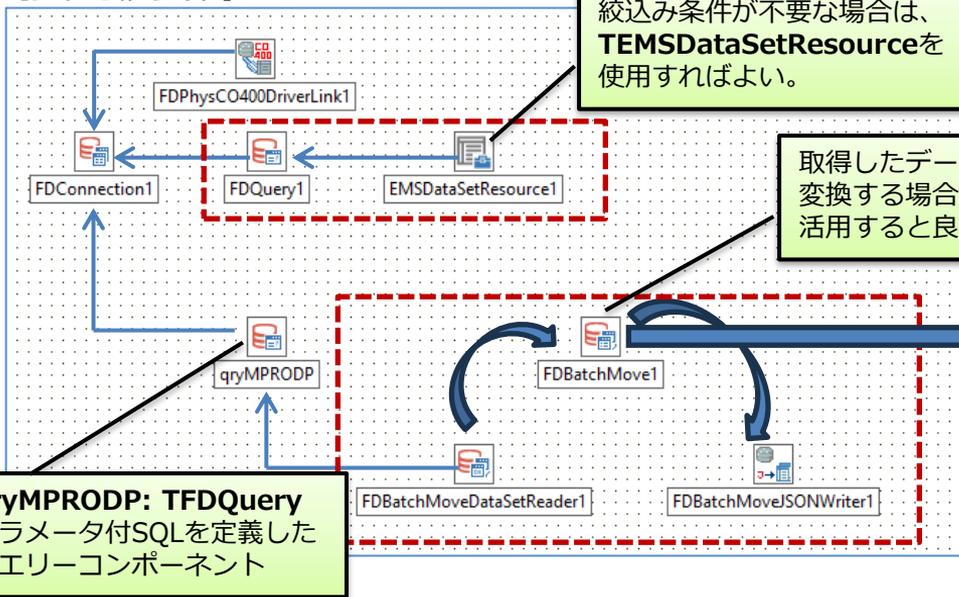
# ■ IBM i データにアクセスするRAD Serverアプリ(2)

## ● 独自の検索条件を指定してデータを取得するRAD Serverアプリ

### ● TFDBatchMoveを活用する

- BatchMoveは、異なるデータ間のコピーを行うコンポーネント
- データセット以外にも、テキストやJSONへのコピーも可能

【フォームデザイナー】



指定されたデータセットを  
コピー元として読み込む

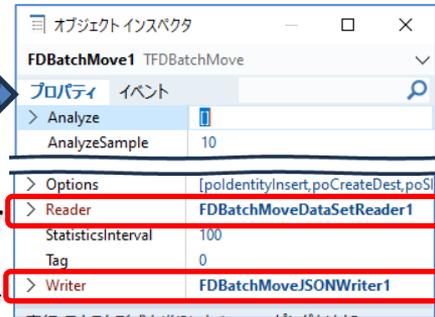
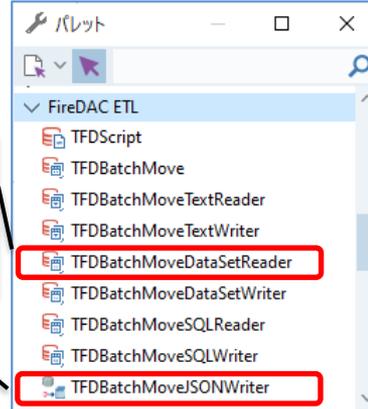
指定された書き出し先を  
JSON文字列をセットする

取得したデータセットをJSON形式に  
変換する場合、**TFDBatchMove**を  
活用すると良い

**Reader**プロパティ  
コピー元となるリーダー  
コンポーネントを指定

**Writer**プロパティ  
コピー先となるライター  
コンポーネントを指定

【BachMove関連コンポーネント】



# ■ IBM i データにアクセスするRAD Serverアプリ(2)

- 独自の検索条件を指定してデータを取得するRAD Serverアプリ
  - パラメータ付SQL実装例
    - 2つのパラメータ“CATG”（カテゴリーCD），“PDNM”（商品名）を定義

## 【qryMPRODPのクエリエディタ】

FireDAC クエリ エディタ - [qryMPRODP]

クエリの SQL、パラメータ、マクロ、オプションをセットアップします

SQL コマンド パラメータ マクロ オプション

```
1 SELECT * FROM MPRODP
2 WHERE
3 ((:CATG = '' ) OR (PDCATG = :CATG))
4 AND ((:PDNM = '' ) OR (PDPDNM LIKE '%' || :PDNM || '%'))
5 ORDER BY PDPDCD
```

実行(E) 次のレコードセット(N) 更新 SQL エディタ(L) 自動ロールバック

PDPDCD	PDPDNM	PDJNCD	PDSPCD	PDCATG	PDUNPR	PDCOPR	PDSTDT
C-00001	プリンター	4924027222510	S00001	06	2000	1200	20220307
C-00002	アンダーシャツ	4953260902737	S00002	06	1000	450	20220115
C-00003	インポートシャツ	4934989471114	S00001	06	3500	2000	20220106
D-00001	柑橘のど飴	4987920773949	S00002	04	200	120	20220205
D-00002	フルーツのど飴	4935331278420	S00001	04	230	140	20220912
J-10001						150	20220428

PDPDNM (商品名) PDCATG (カテゴリーCD)

## パラメータの値が

空白でも、空白以外でも一つのSQL文で処理が可能

```
((:CATG = '' ) OR (PDCATG = :CATG))
```

AND

パラメータ“CATG”=空白の場合は、  
全てのデータを抽出する  
パラメータ“CATG”≠空白の場合は、  
[PDCATG]の値がパラメータ“CATG”と  
合致するデータのみ抽出する

```
((:PDNM = '' ) OR (PDPDNM LIKE '%' || :PDNM || '%'))
```

[PDPDNM](商品名)は、パラメータ“PDNM”と  
部分一致するデータを抽出する

# ■ IBM i データにアクセスするRAD Serverアプリ(2)

- 独自の検索条件を指定してデータを取得するRAD Serverアプリ
  - ソースコード作成 (宣言部)

TTestResource1クラスの宣言部

```
type
[ResourceName('test')]
TTestResource1 = class(TDataModule)
  FDConnection1: TFConnection;
  FDPhysC0400DriverLink1: TFDPPhysC0400DriverLink;
  FDQuery1: TFDQuery;
  [ResourceSuffix('mprodp')]
  EMSDataSetResource1: TEMSDataSetResource;
  qryMPRODP: TFDQuery;
  FDBatchMove1: TFDBatchMove;
  FDBatchMoveDataSetReader1: TFDBatchMoveDataSetReader;
  FDBatchMoveJSONWriter1: TFDBatchMoveJSONWriter;
published
//   procedure Get(const AContext: TEndpointContext; const ARequest:
//     TEndpointRequest; const AResponse: TEndpointResponse);
//   [ResourceSuffix('{item}')]
//   procedure GetItem(const AContext: TEndpointContext; const ARequest:
//     TEndpointRequest; const AResponse: TEndpointResponse);
//   [ResourceSuffix('mprodp_opt')]
  procedure GetMPRODP(const AContext: TEndpointContext; const ARequest:
    TEndpointRequest; const AResponse: TEndpointResponse);
end;
```

手順(3)

始めにウィザードで作成した2つの手続きは不要の為、削除。(コメントアウト)  
※実装部も要コメントアウト

手順(2)

定義した**GetMPRODP**の宣言部の上に、**[ResourceSuffix('mprodp\_opt')]** を行追加する。  
リソース名とターゲットとなる手続きが関連付けされる。

手順(1)

ウィザードで自動作成されたGet手続きの宣言をコピーして、新たに**GetMPROP**手続きを作成  
※[Ctrl]+[Shift]+[C]で実装部のテンプレートを作成可能



# ■ IBM i データにアクセスするRAD Serverアプリ(2)

- 独自の検索条件を指定してデータを取得するRAD Serverアプリ
  - ソースコード作成 (実装部)

TTestResource1クラスの実装部

```
procedure TTestResource1.GetMPRODP(const AContext: TEndpointContext;  
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);  
var  
  sCATG: String; //カテゴリー  
  sPDNM: String; //品名(部分一致)  
begin  
  //オプションパラメータを取得し変数にセット(存在しない場合空白)  
  if not ARequest.Params.TryGetValue('catg', sCATG) then sCATG := '';  
  if not ARequest.Params.TryGetValue('pdmn', sPDNM) then sPDNM := '';  
  
  //パラメータクエリーに値をセット  
  qryMPRODP.Active := False;  
  qryMPRODP.ParamByName('CATG').AsString := sCATG;  
  qryMPRODP.ParamByName('PDMN').AsString := sPDNM;  
  
  //クエリー結果をJSONにコピーしてレスポンスとしてセット  
  FDBatchMoveJSONWriter1.JsonWriter := AResponse.Body.JSONWriter;  
  FDBatchMove1.Execute;  
end;
```

**ARequest.Params.TryGetValue**メソッド

指定したパラメータの値を取得し、変数に代入  
パラメータが存在しない場合、Falseが返却

qryMPRODPにパラメータをセット

JSONライターの書き出し先に、  
**Aresponse.Body.JSONWriter**を指定すればよい。

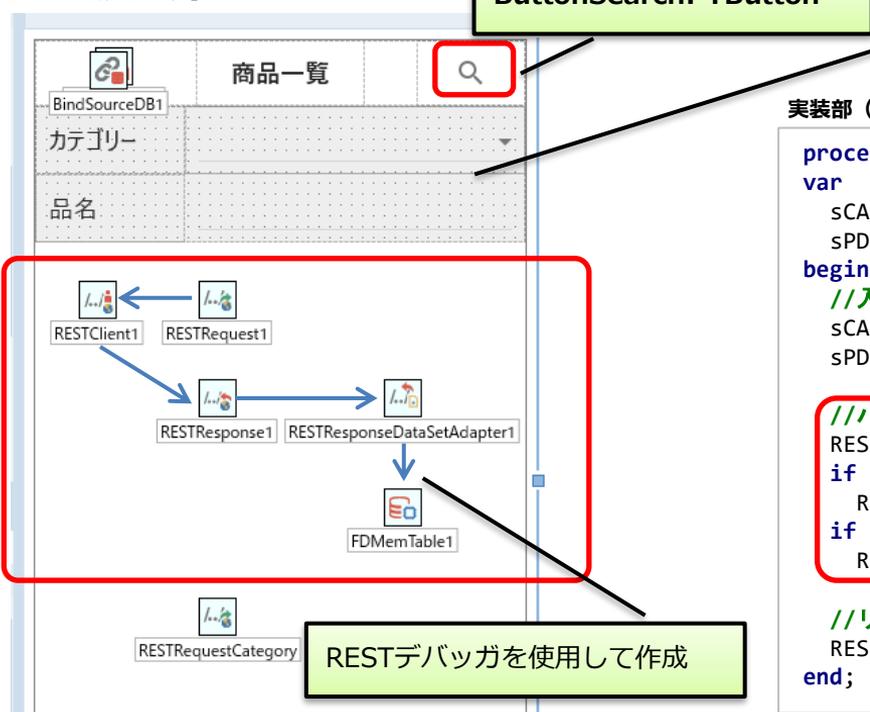
**FDBatchMove1.Execute**により、クエリーが実行  
され取得したデータセットをJSON文字列にコピーする。



# ■ IBM i データにアクセスするRAD Serverアプリ(2)

## • モバイルアプリ作成例

【フォームデザイナー】



カテゴリの選択と、品名の入力を行う条件指定欄  
cmbCATG: TComboBox  
EditPDNM: TEdit

実装部 (一部抜粋)

```
procedure TFormMain.ButtonSearchClick(Sender: TObject);
var
  sCATG: String; //カテゴリCD
  sPDNM: String; //商品名 (部分一致)
begin
  //入力値を取得
  sCATG := GetCATG; //----選択されたカテゴリCDを取得して変数にセット
  sPDNM := Trim(EditPDNM.Text);

  //パラメータの初期化
  RESTRequest1.Params.Clear;
  if sCATG <> '' then
    RESTRequest1.Params.AddItem('catg', sCATG, pkQUERY);
  if sPDNM <> '' then
    RESTRequest1.Params.AddItem('pdnm', sPDNM, pkQUERY);

  //リクエスト実行
  RESTRequest1.Execute;
end;
```

リクエストパラメータの追加処理



# ■ IBM i データにアクセスするRAD Serverアプリ(2)

- モバイルアプリ作成例
  - 完成したモバイルアプリを実行



さいごに



# ■ モバイルアプリ開発についての関連情報

- ミガロ、テクニカルセミナー
  - 過去に実施した主なDelphi/400モバイルアプリ開発に関する技術セッション

実施回	セッションタイトル	概要
第13回	<a href="#">Delphi/400 XE5 –こんなに簡単！IBM iスマートデバイスネイティブ開発–</a>	XE5を使用した基本的なモバイルアプリ開発手法
第15回	<a href="#">マルチデバイスに対応したIBM i業務システム開発のポイント</a>	モバイル画面設計ポイントとDataSnapサーバーとの連携
第15回	<a href="#">実践！iOS / Android ネイティブ機能開発</a>	バーコード読み取りや、手書き署名、オフライン保存方法
第20回	<a href="#">VCL開発者が知っておきたいFireMonkeyアプリ開発のポイント</a>	VCLとFireMonkeyの違い、各種コンポーネントについて
第21回	<a href="#">ステップアップ！モバイルアプリケーション開発</a>	アップテザリング、モバイルアプリにおける帳票実装
第22回	<a href="#">モバイルからクラウドサービスまで！活用が広がる多層アプリ開発</a>	DataSnapやRADServer構築方法
第23回	<a href="#">レスポンス向上！モバイルアプリ実践開発テクニック</a>	並列処理の実装、アニメーション効果について

※上記各資料は、作成当時のバージョンのDelphi/400を元に作成されています。現在のバージョンとは仕様や内容が異なる場合がございます。



# ■ モバイルアプリ開発に関するその他情報

- Delphiにおけるモバイルアプリ開発
  - エンバカデロ YouTube Embarcadero Developers TV

<https://www.youtube.com/@EmbarcaderoTechJapan>

『Delphiモバイル開発ファーストステップ』 1～5

- 開発環境の構築方法についての詳細な解説
- Layoutコンポーネントの活用方法等

- RadServerについて
  - RAD Serverガイド

<https://www.embarcadero.com/jp/resources/white-papers/rad-server-guide>

- より詳細なRADServerアプリの開発手法
- サーバー運用環境の構築方法
- RadServerLiteについて



ご清聴ありがとうございました

