【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】

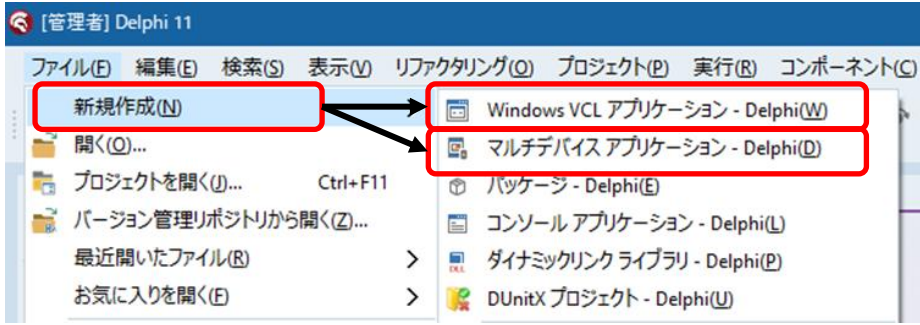
Delphi/400を使用した最新モバイルアプリ開発入門

- 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

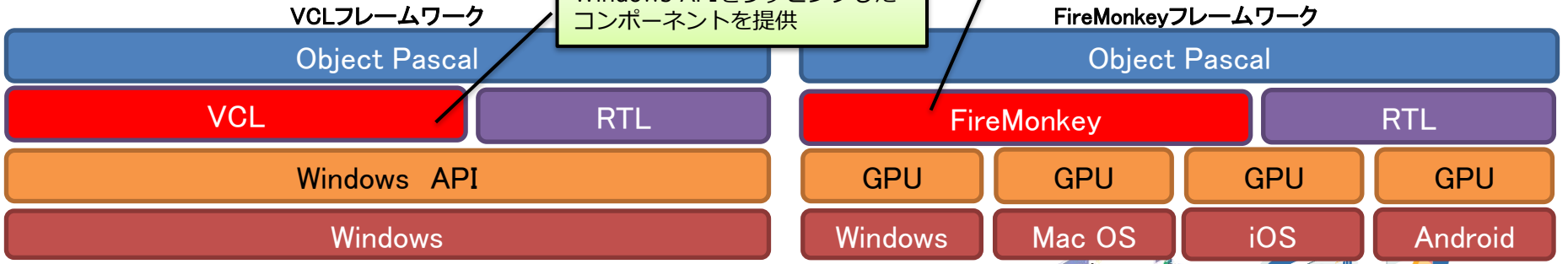
- 2つのGUIフレームワークが使用可能
  - **Windows VCL アプリ（VCL）**
    - Windows APIをラッピングした従来から使用可能なフレームワーク
    - Windowsアプリ開発専用
  - **マルチデバイスアプリ（FireMonkey）**
    - Delphi/400 XE5以降に搭載されたマルチデバイスアプリ開発用フレームワーク
    - Win/Mac/iOS/Androidアプリ開発に対応



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

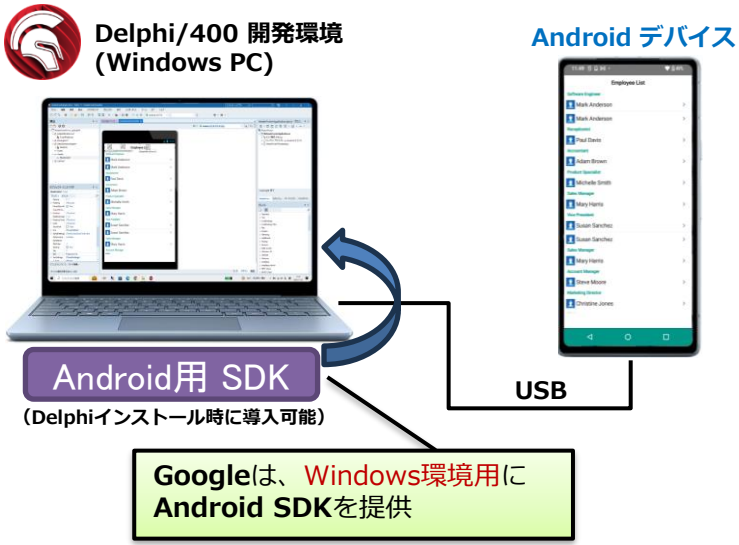
視覚効果がより高いアプリが作成可能

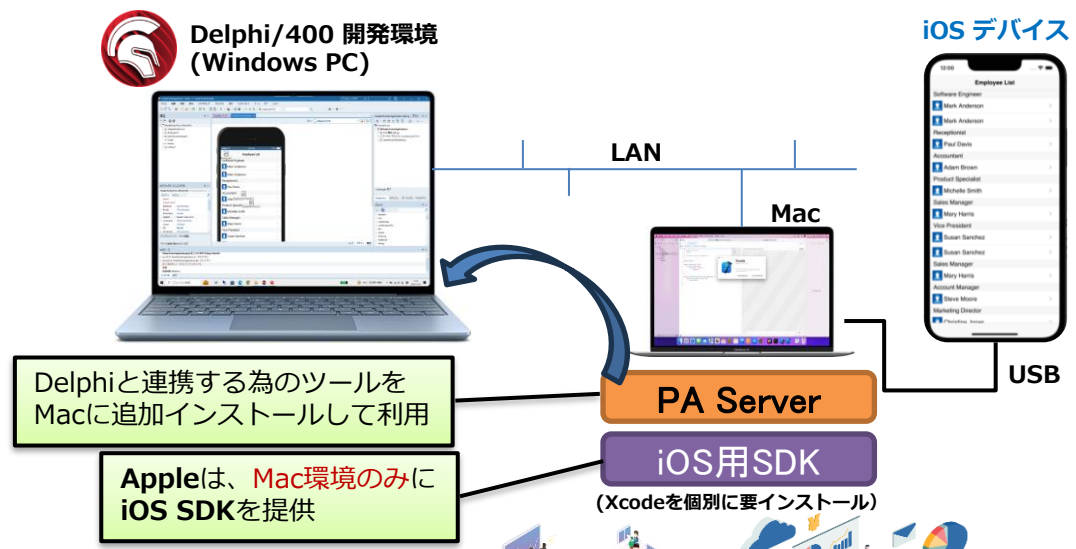
Windows APIをラッピングしたコンポーネントを提供



Delphi/400におけるモバイルアプリ開発

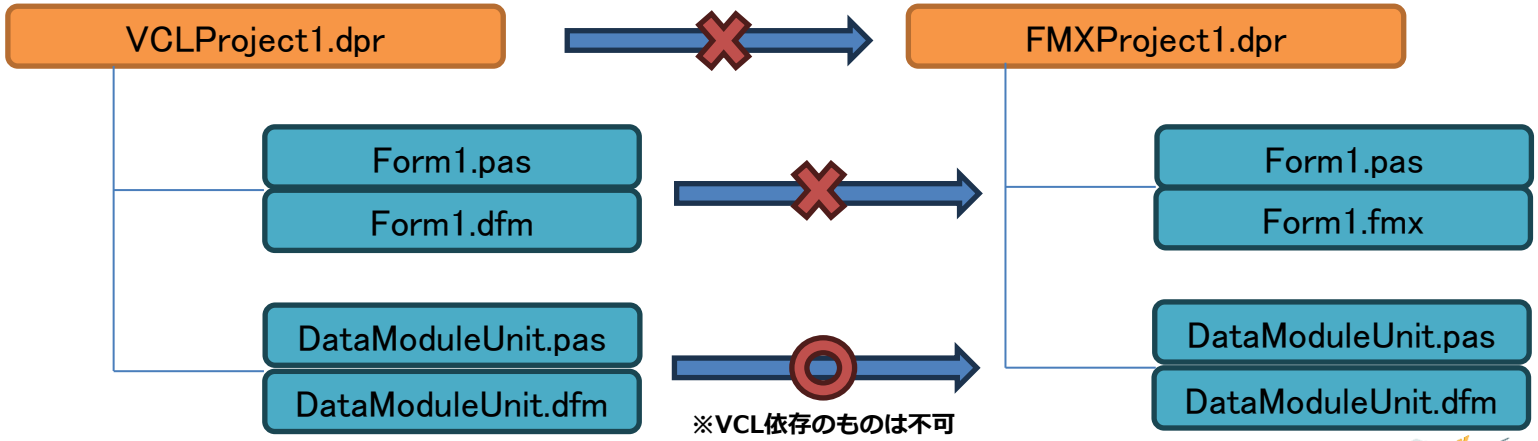
- モバイルアプリ開発には、**FireMonkey**を使用する
  - iOS, Androidのいずれのモバイル用OS向けアプリ開発にも対応可能。
  - ターゲットとなるデバイスに加え、iOSの場合、モバイルアプリをビルド・配置する為にMacやApple Developer Programへの加入が別途必要。





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

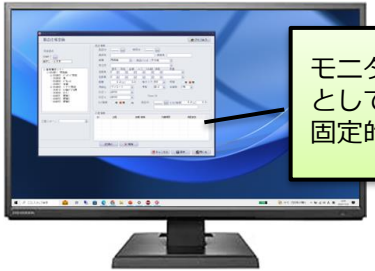
- 言語は、いずれのフレームワークもObject Pascalを使用
- 文字列リスト(TStringList）のようなRTLは共通に使用可能
  - RTL（ランタイム ライブラリ）の利用
    https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/RTL（ランタイム_ライブラリ）の利用
- TEditやTButtonのようにコンポーネントは同名のものが多く、VCL同様に直感的な開発は可能だが、**似て非なるものとしてFireMonkeyの流儀を知る事**が重要
  - VCL と FireMonkey のよくある違い
    https://docwiki.embarcadero.com/RADStudio/Alexandria/ja/VCL_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

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

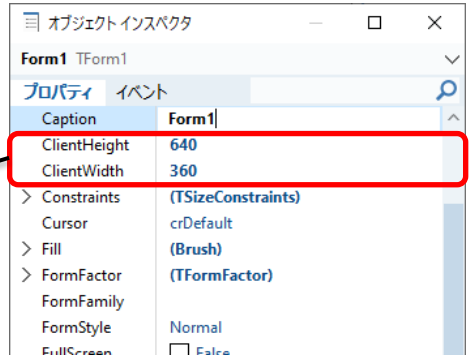
■ モバイルアプリのフォーム

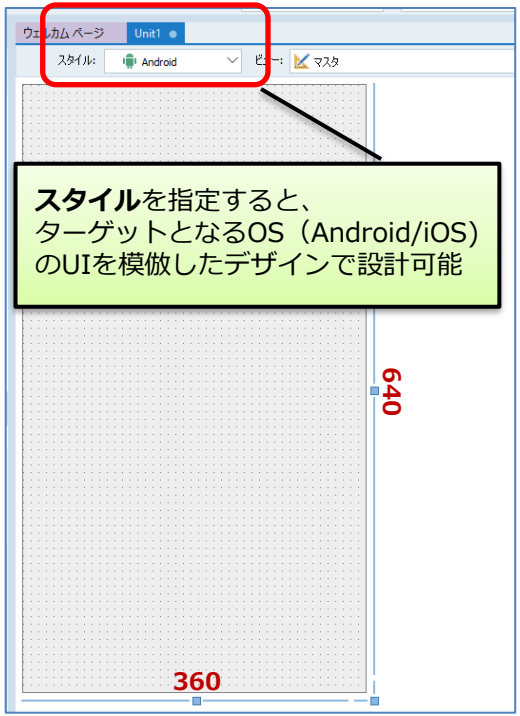
- フォーム(TForm) のサイズ
  - スマートフォンをターゲットとするアプリであれば、縦横比**16:9**となる以下のサイズを基準に設計すればよい。

    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**

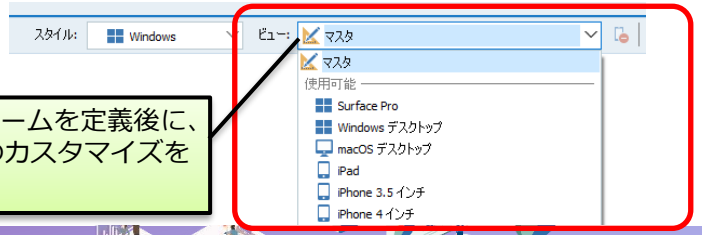
デバイスによって、縦横比は異なるが（近年は、縦に長い傾向）、**16:9**を前提とすれば、凡そのデバイスは網羅可能。





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（ビュー）を活用する事も可能（必須ではない）

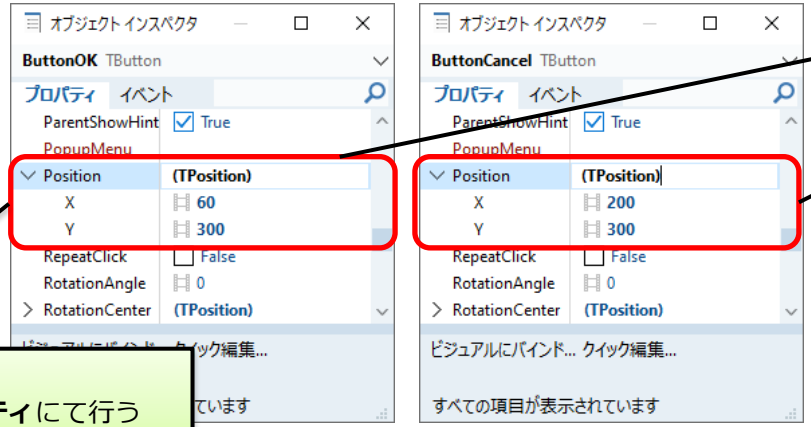


基準となる[マスタ]フォームを定義後に、デバイス毎のフォームのカスタマイズを行い最適化する仕組み

■ 固定的なレイアウト配置

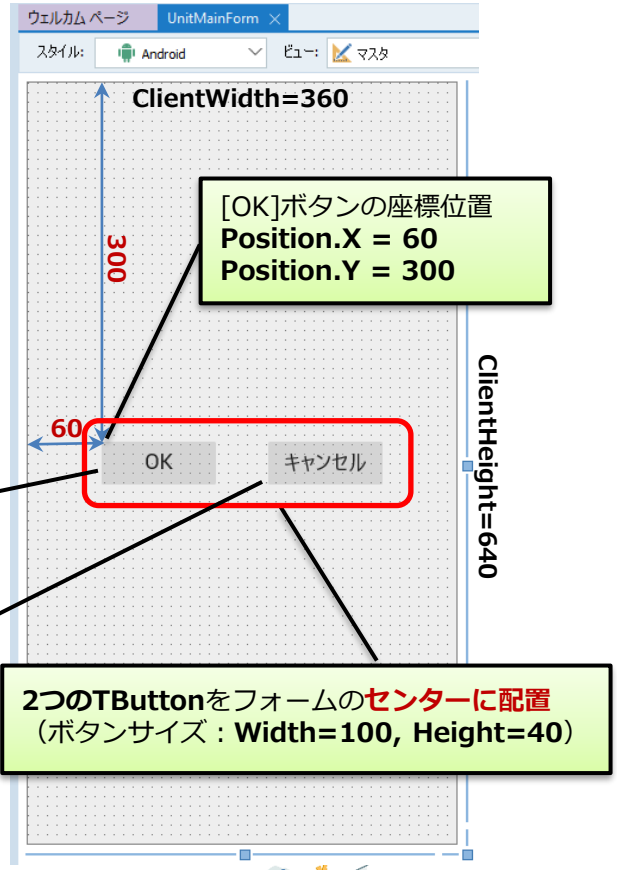
- VCLの場合、各コンポーネントの配置は、Top/Leftプロパティによる絶対座標で指定する事が多い。

- 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】

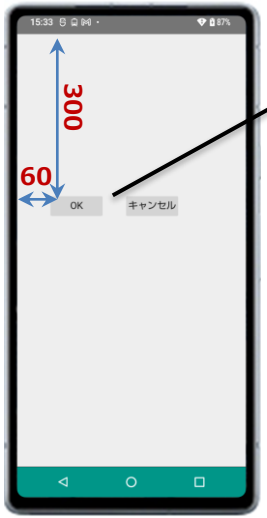


[OK]ボタンの座標位置
Position.X = 60
Position.Y = 300

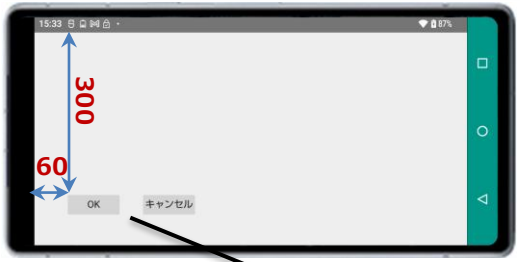
2つのTButtonをフォームの**センターに配置**
（ボタンサイズ：**Width=100, Height=40**）

■ 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



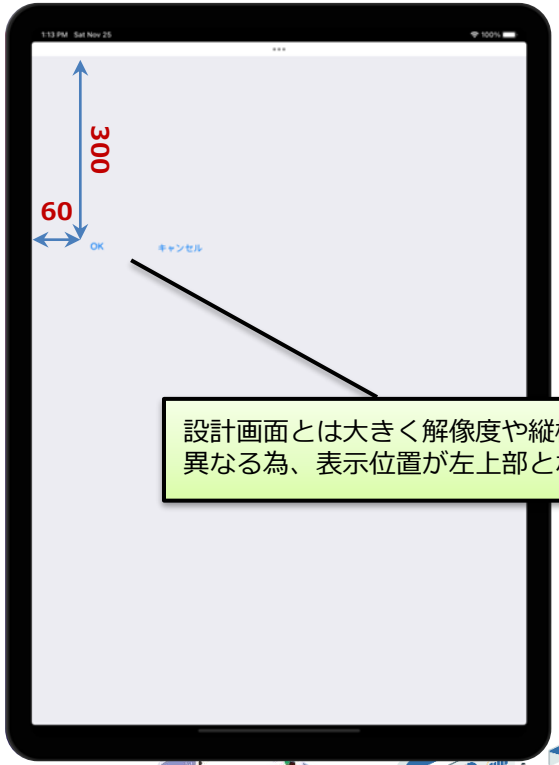
縦横比が設計画面と異なる為、上下/左右とも
センター位置から若干ずれている



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？

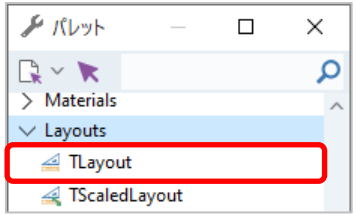
- iPadで実行
  11インチ(1668×2388px)のデバイス



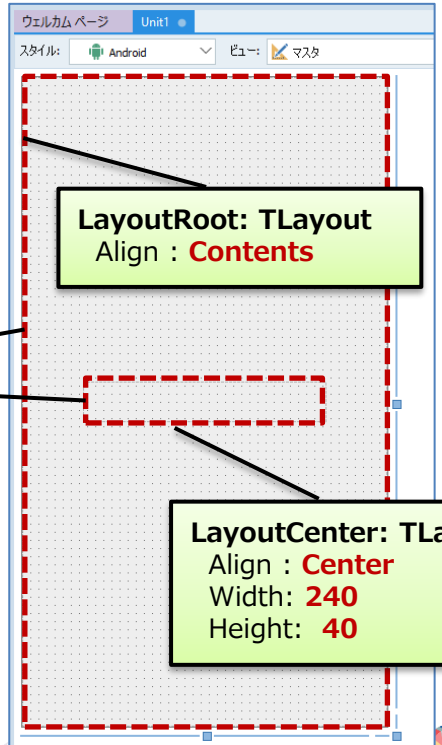
設計画面とは大きく解像度や縦横比が
異なる為、表示位置が左上部となる

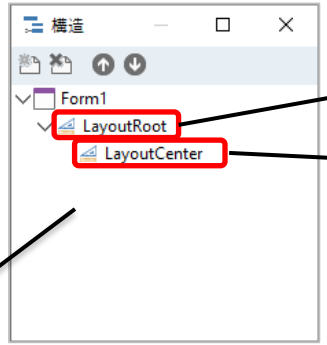
■ モバイルに最適なレイアウト作成方法

- 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】





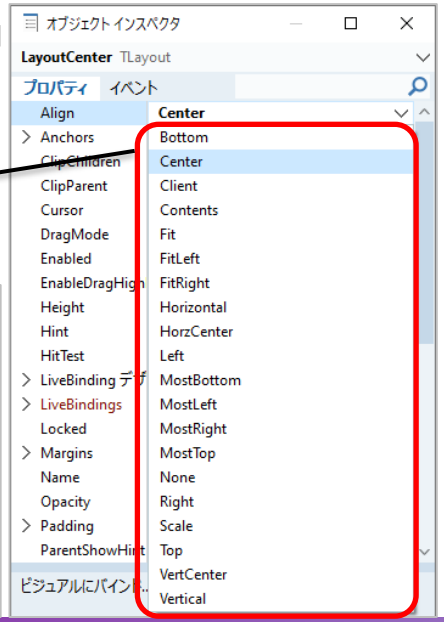
画面全体の土台となるTLayoutコンポーネントを
フォームの上に配置。（**LayoutRoot**）
配置を設定する**Align**は、親（フォーム）コンポーネント
全体にあわせてサイズが変更される**Contents**を指定。

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

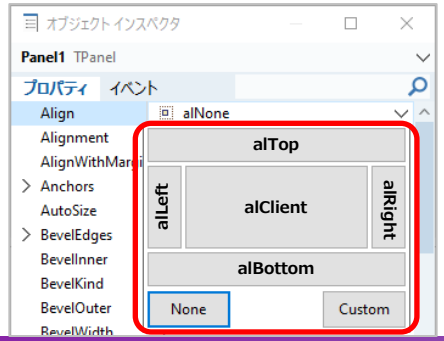
- Alignプロパティ
  - 親コンポーネントに対する
    コンポーネントの相対的な位置を
    設定するプロパティ

【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】

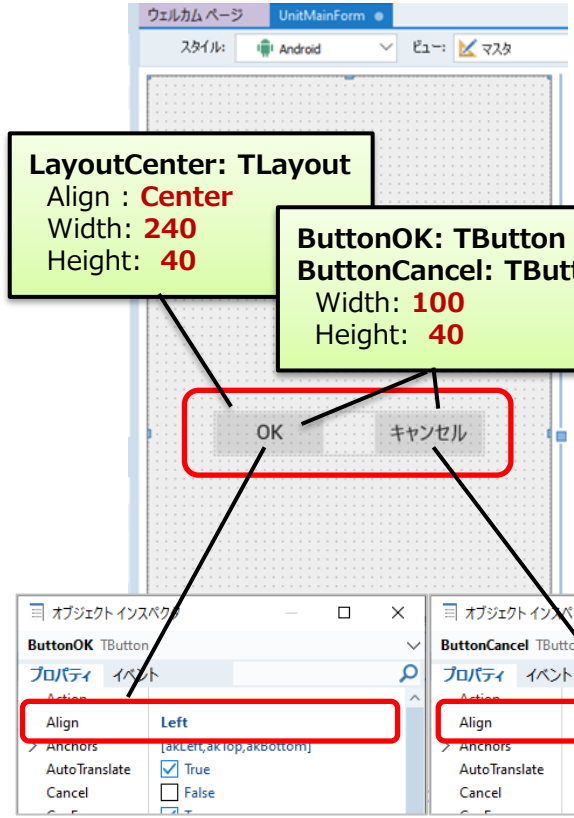


【設定値一覧】

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。	alButtom
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に広がる	alClient
Contents	親領域いっぱいに広がる	—
Fit	縦横比を保ったまま親いっぱいに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に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に広がり親の右端で固定	—
Horizontal	親の幅いっぱいに広がる（高さは変わらない）	—
HorzCenter	横位置は親の中央となり、高さは親いっぱいに広がる。（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、幅は親いっぱいに広がる。（高さは変わらない）	—
Vertical	親の高さいっぱいに広がる（幅は変わらない）	—

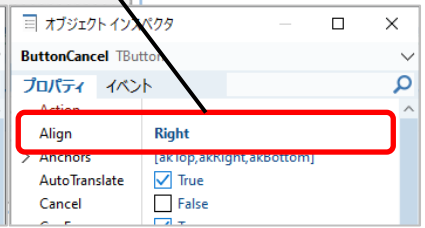
■ モバイルに最適なレイアウト作成方法

【フォームデザイナ】



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

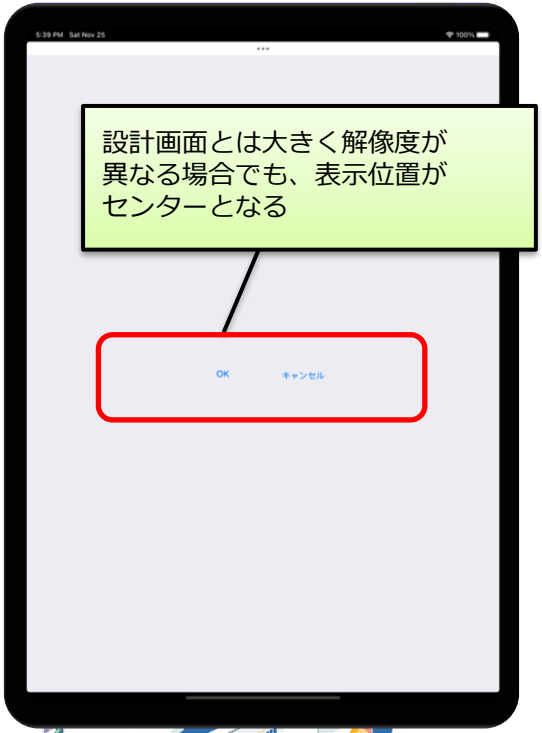
ButtonOK: TButton
ButtonCancel: TButton
Width: **100**
Height: **40**



- 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

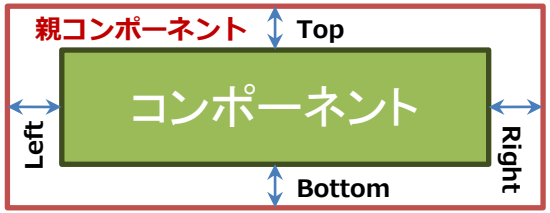


設計画面とは大きく解像度が
異なる場合でも、表示位置が
センターとなる

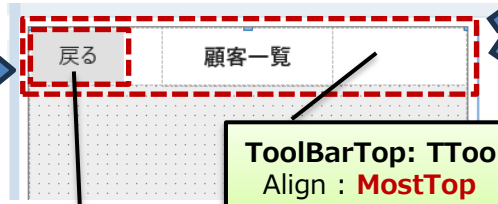
- iPadで実行

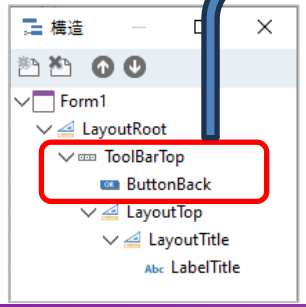
■ モバイルに最適なレイアウト作成方法

- 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



【フォームデザイナ】

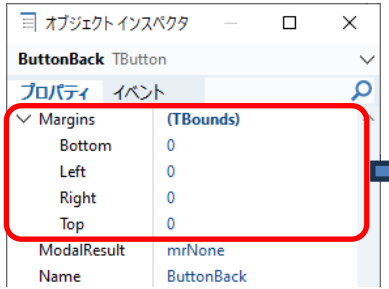


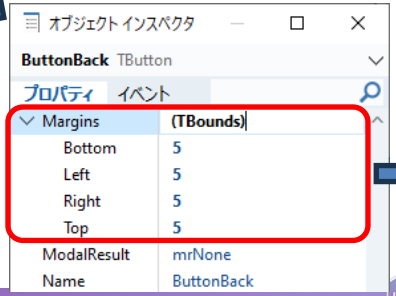


ToolBarTop: TToolBar
Align : **MostTop**

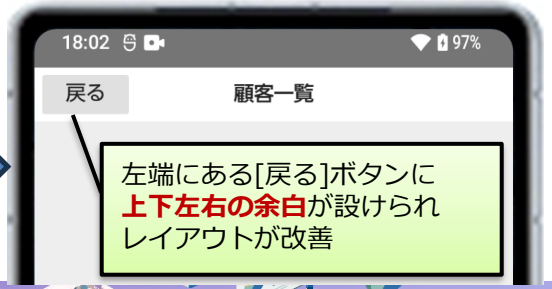
ButtonBack: TButton
Align : **Left**
Width: **80**

Marginsプロパティ





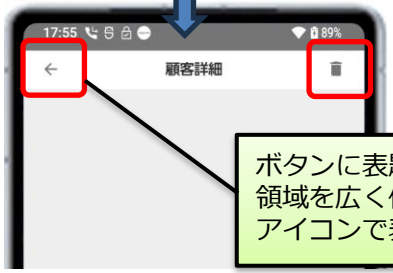
上部ツールバーの中にある
[戻る]ボタンは、**余白無く**
左端(Left)に配置



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

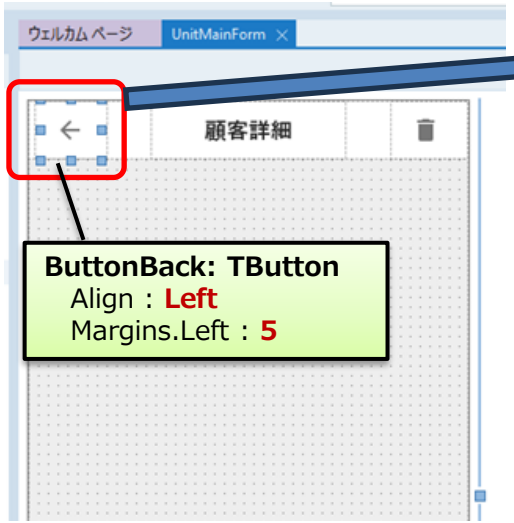
■ モバイルに最適なレイアウト作成方法

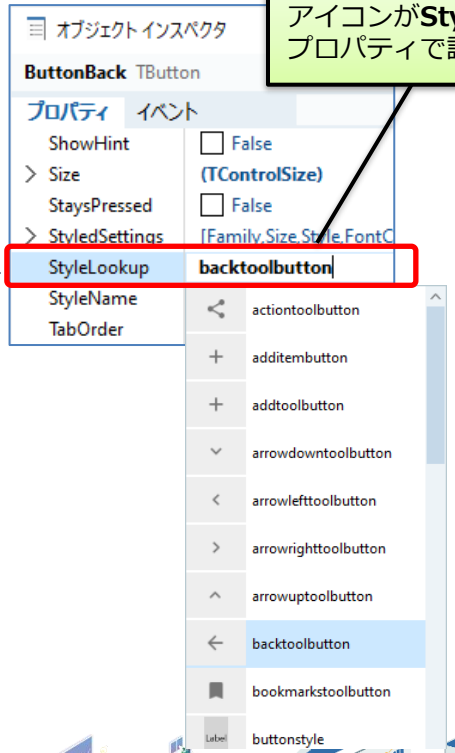
- ボタンアイコンの使用
  - 画面領域の小さいスマートフォンでは、ボタンに表題を設定する代わりアイコンを設定する事が多い



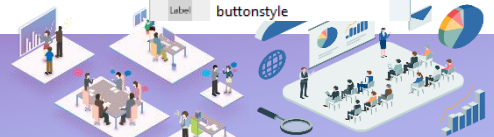
ボタンに表題を設定すると、
領域を広く使ってしまう為
アイコンで表現する事が多い

【フォームデザイナ】





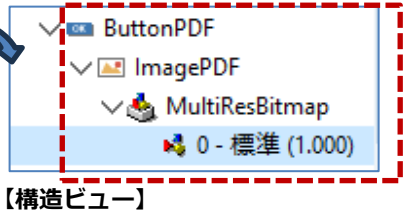
ボタンで多用する機能
アイコンが**StyleLookup**
プロパティで設定可能

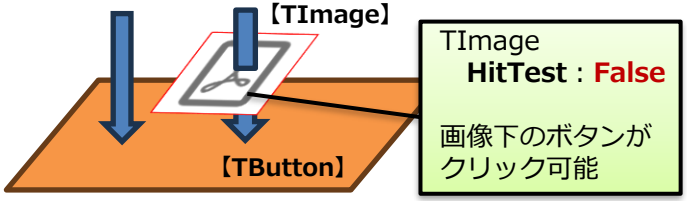


■ モバイルに最適なレイアウト作成方法

- ボタンアイコンの使用

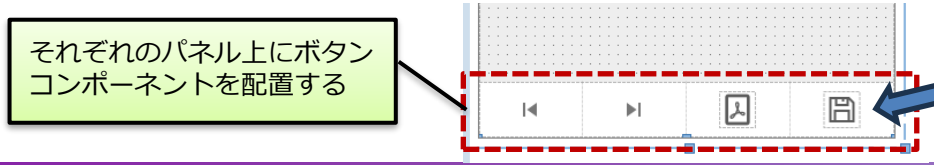
- TImageコンポーネント
  - TButtonに直接画像は設定できないが、ボタンの子として画像を追加して、アイコンが設定可能

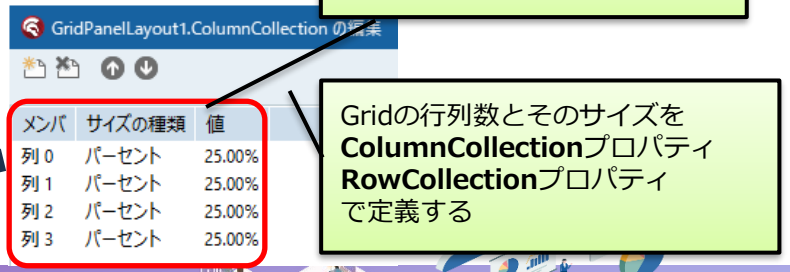




※ Get Itから**[Icon8]**をインストールすれば多彩なアイコン画像が無料で取得が可能

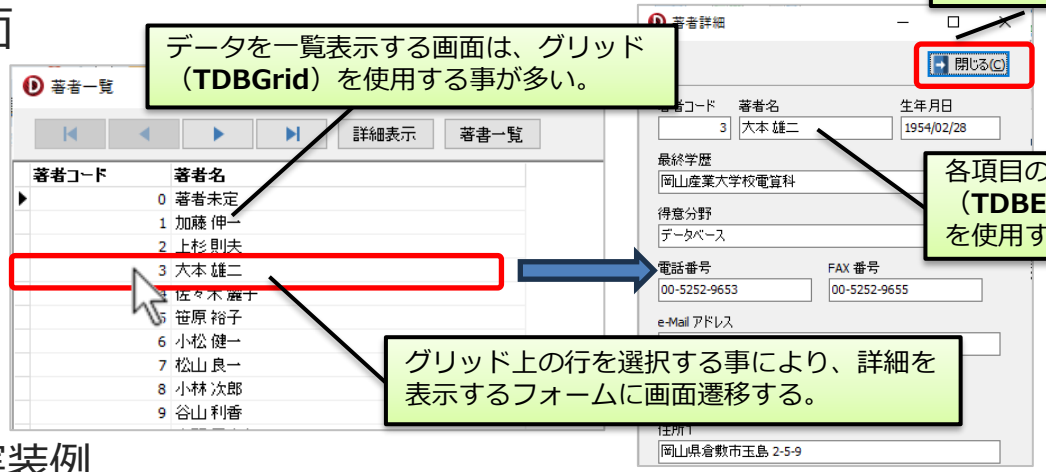
- TGridPanelLayoutコンポーネント
  - Grid上のパネルの上にコンポーネントが配置できるコンテナ





■ 一般的な画面構成

- 一覧画面と詳細画面
  - PCアプリ実装例



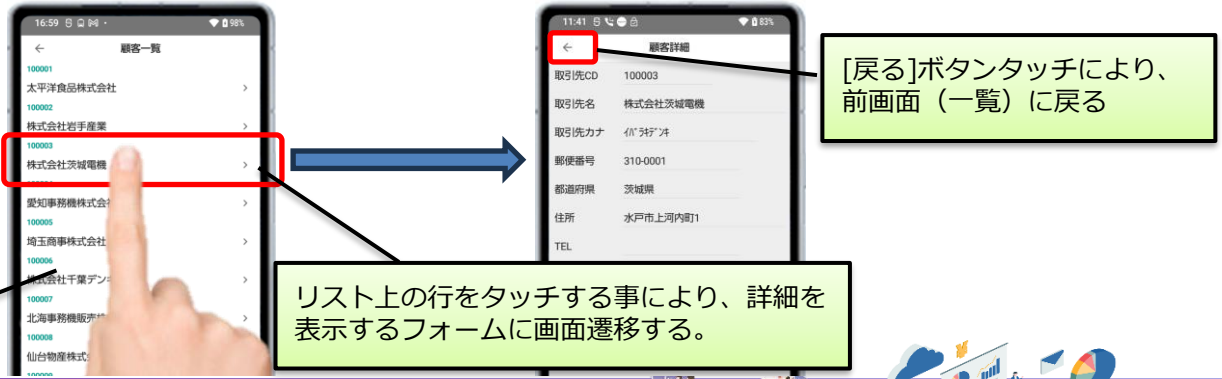
データを一覧表示する画面は、グリッド（**TDBGrid**）を使用する事が多い。

[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。

  - 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。

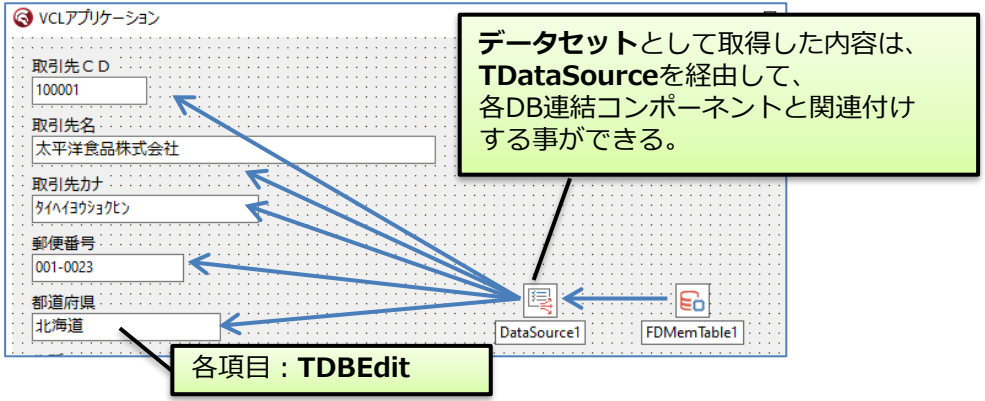
操作性を考慮した場合、リスト形式でデータを一覧表示した方が適切な事が多い。

[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。

■ 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

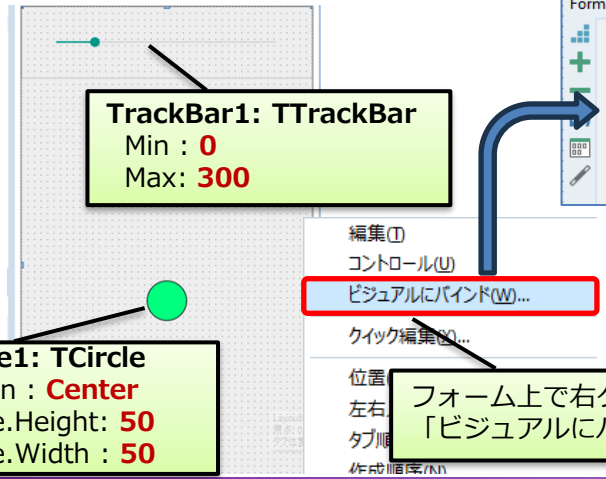
FireMonkeyアプリで、データ項目を簡単に表示させるにはどうすればよいか？

■ 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**」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】

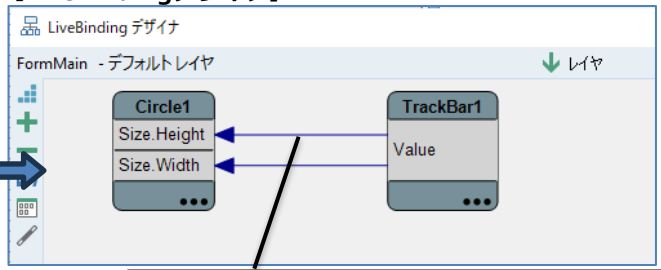


TrackBar1: TTrackBar
Min : **0**
Max: **300**

Circle1: TCircle
Align : **Center**
Size.Height: **50**
Size.Width : **50**

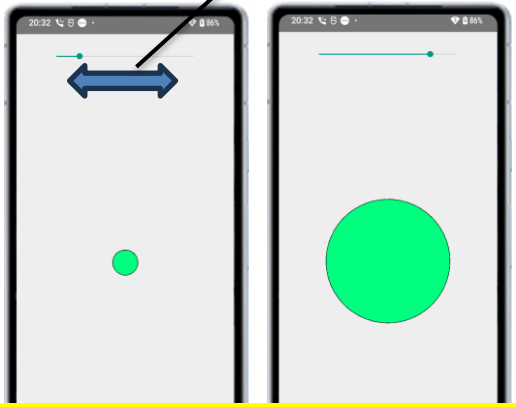
フォーム上で右クリックし
「ビジュアルにバインド」を選択

【LiveBindingデザイナ】



TackBar1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

【実行イメージ】



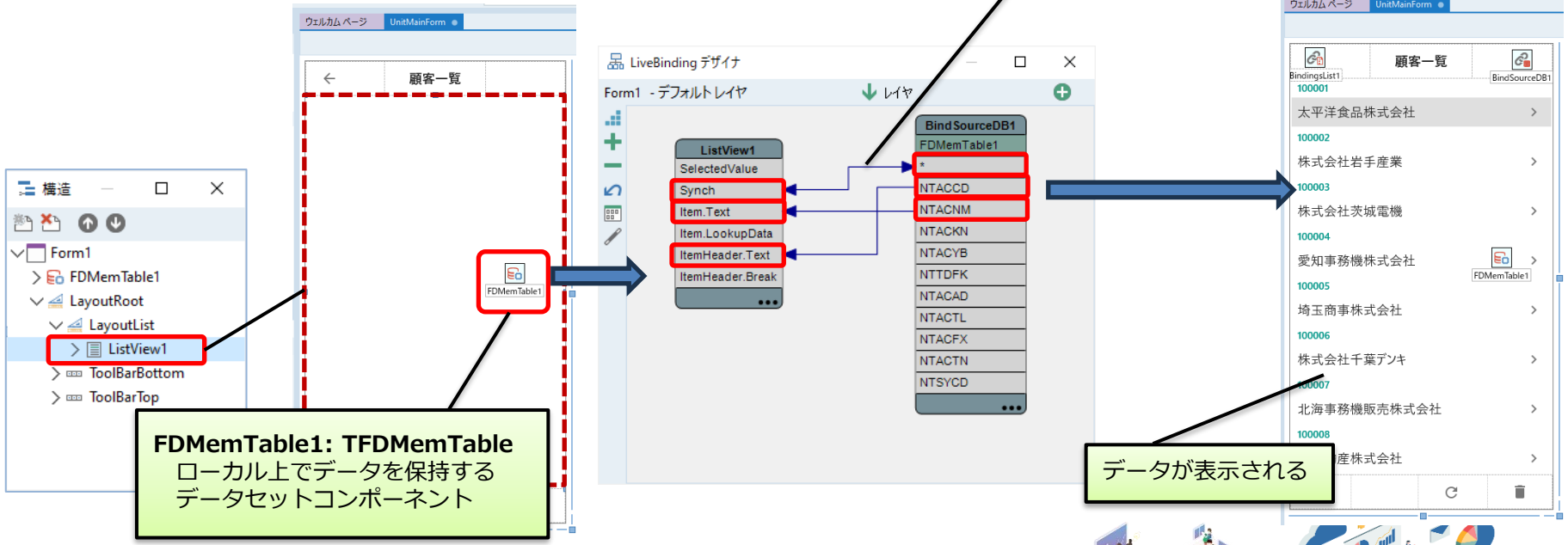
トラックバーの位置（**Value**）を変更すると連動して円の大きさ(**Size.Height/Width**)が自動的に変わる

ノンコーティングで項目間の
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
ListView　　FDMemTable1
Synch　　⇔ *
（リスト選択とデータセットを同期）
Item.Text　　⇐ NTACNM(顧客名)
ItemHeader.Text ⇐ NTACCD (顧客名)

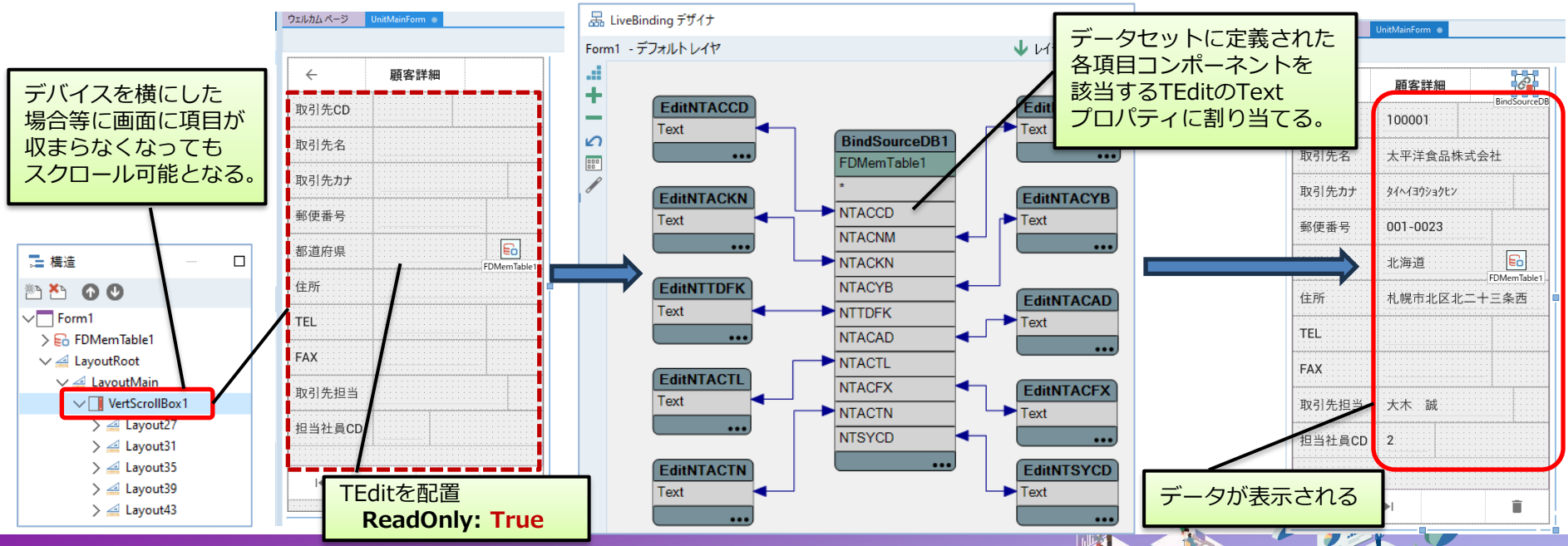


FDMemTable1: TFDMemTable
ローカル上でデータを保持する
データセットコンポーネント

データが表示される

■ データ項目の表示

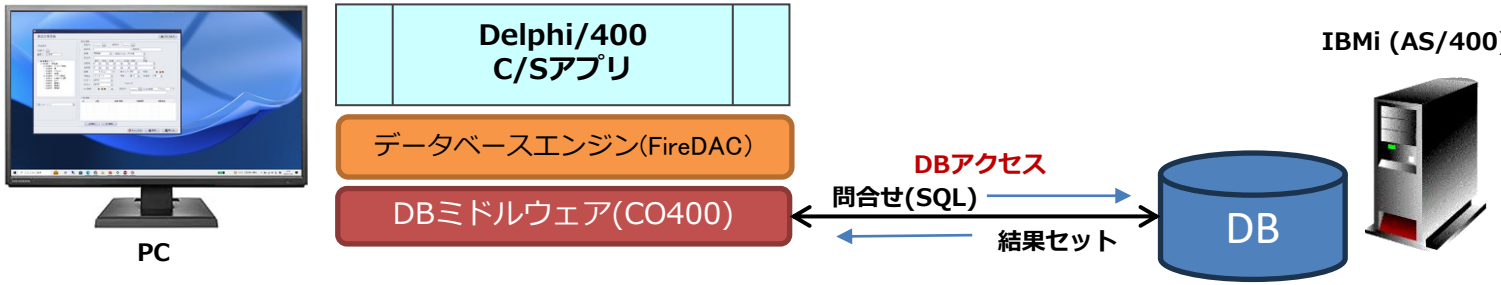
- 詳細画面
  - フォーム上の各コンポーネントも**TLayout**を使用して相対的な位置で配置
  - データ部分には、**TVertScrollBox**を配置。画面に収まらない場合縦スクロールが可能



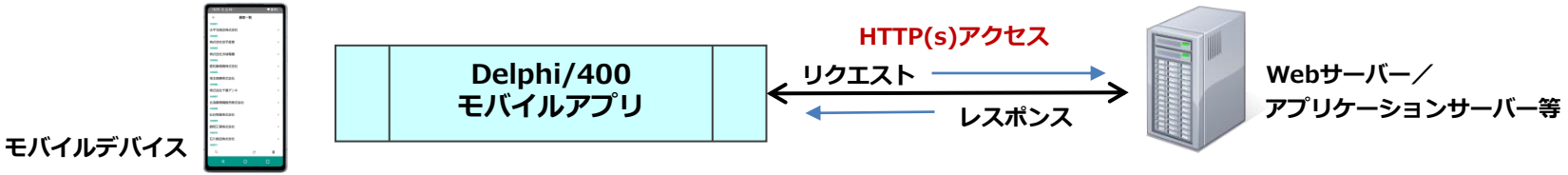
IBM i DB情報を活用する
モバイルアプリ開発手法

■ 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）



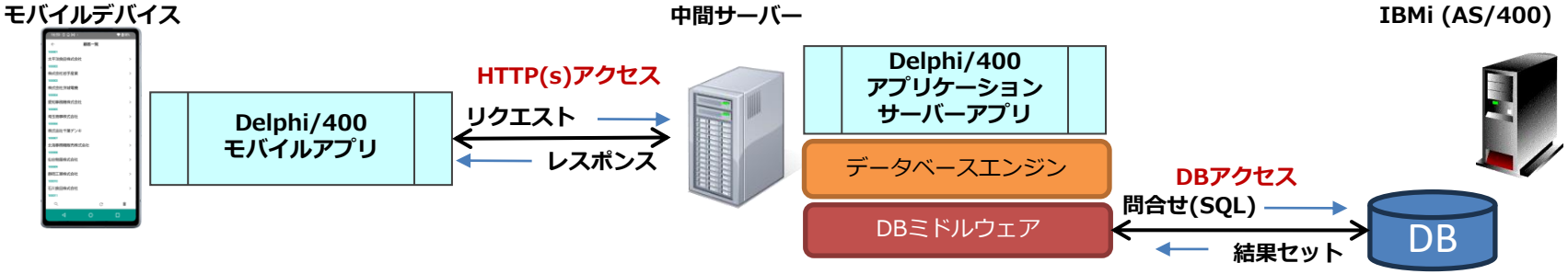
- モバイルアプリ（iOS/Android）は、OSの制約上DBミドルウェアを直接組み込めない
  - 外部との通信は、HTTP(s)等を使用したネットワーク通信に限られる



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

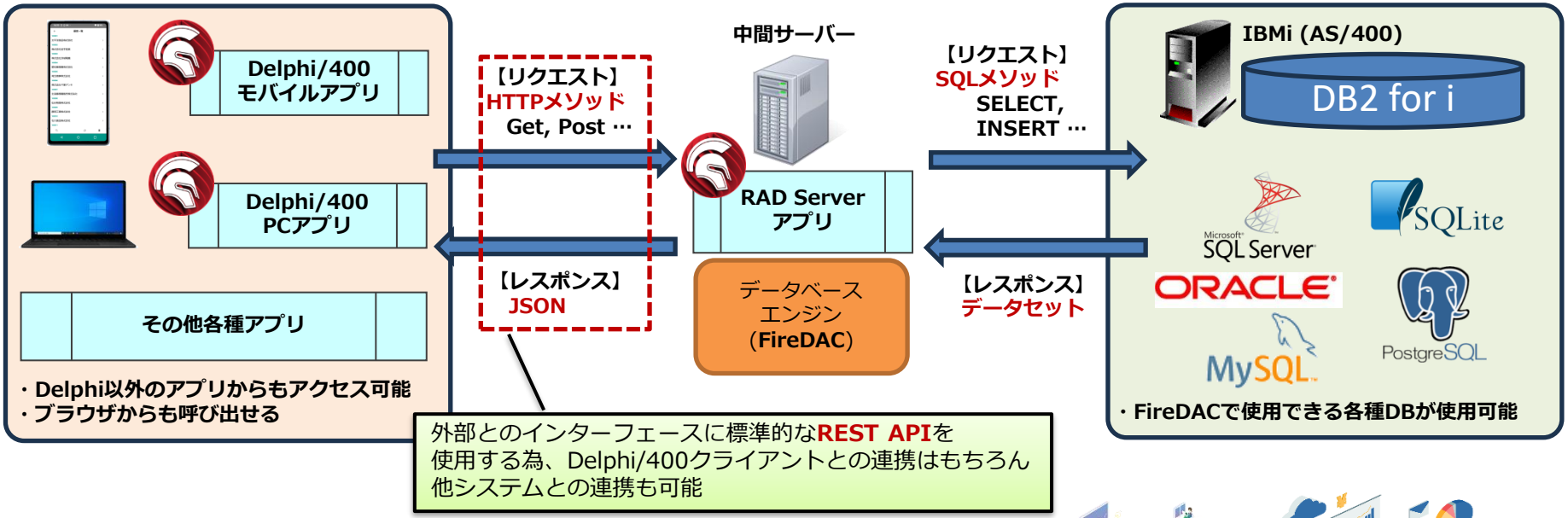


- Delphi/400では、従来3層アプリを**DataSnap**フレームワークで開発する事が多かった
  - Delphiクライアントアプリとの通信であれば、dbExpressやFireDACを使用して容易に作成できる
  - 基本的なサーバー機能以外は原則全て独自に開発しなければいけない
  - 過去のテクニカルセミナーで作成手順はご紹介済み
    https://www.migaro.co.jp/contents/support/technical_seminar_search/14th/Session2.pdf

- 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**アクセスを一意な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	/{resource}/ /{resource}/{item}	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	SELECT * FROM {resource} SELECT * FROM {resource} WHERE key = {item}
POST	/{resource}/	データ（リソース）に新しい項目（アイテム）を追加	INSERT INTO {resource} VALUES [data]
PUT	/{resource}/{item}	データ（リソース）中の特定の項目（アイテム）の情報を更新	UPDATE {resource} SET [data] WHERE key = {item}
DELETE	/{resource}/{item}	データ（リソース）中の特定の項目（アイテム）の情報を削除	DELETE FROM {resource} WHERE key = {item}

【処理イメージ】

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

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
（TestServer）

<対応するSQLを実行>
SELECT * FROM MPRODP
WHERE PDPDCD = 'C-10001'

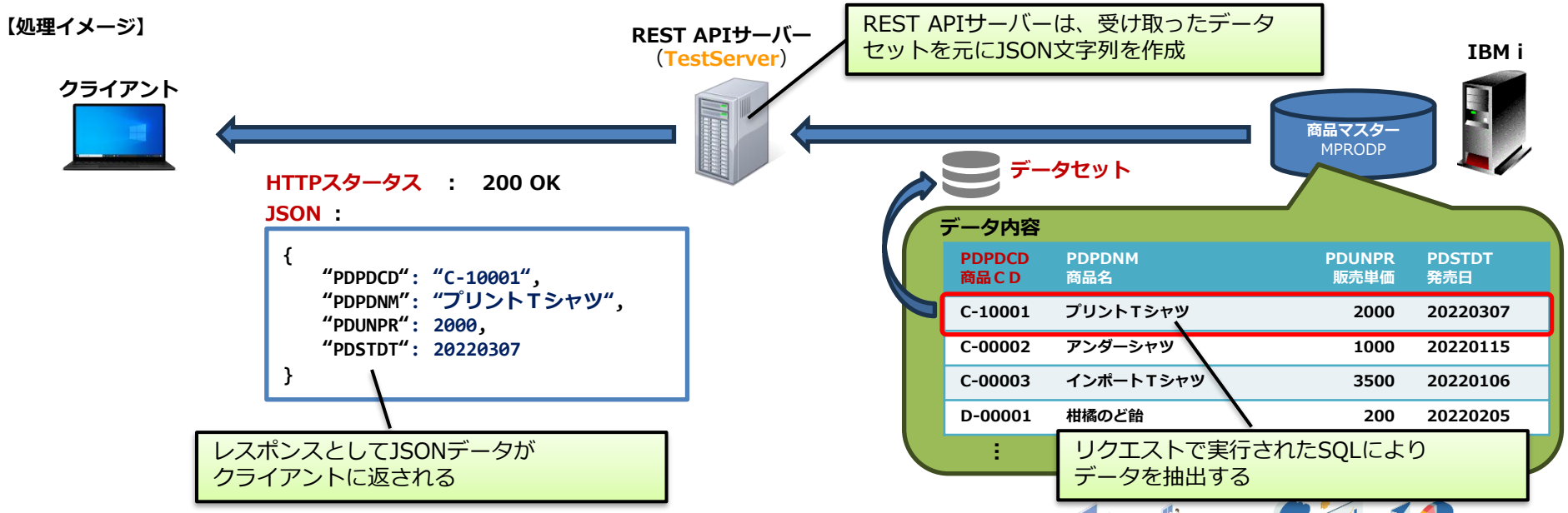
IBM i

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

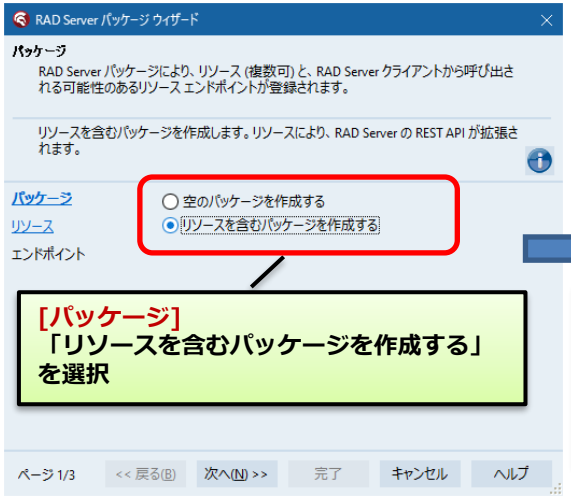
REST APIの基本

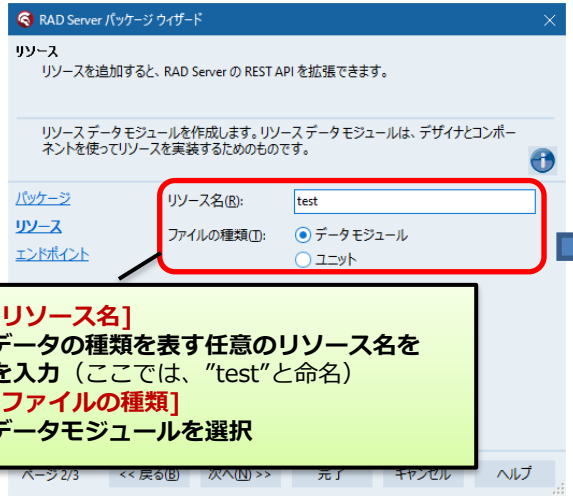
- REST APIにおけるレスポンス
  - データベースから受け取った情報をJSON形式に変換してクライアントに返却する
  - 処理結果をHTTPステータスコードで返す（正常終了の場合、200:OKを返す）

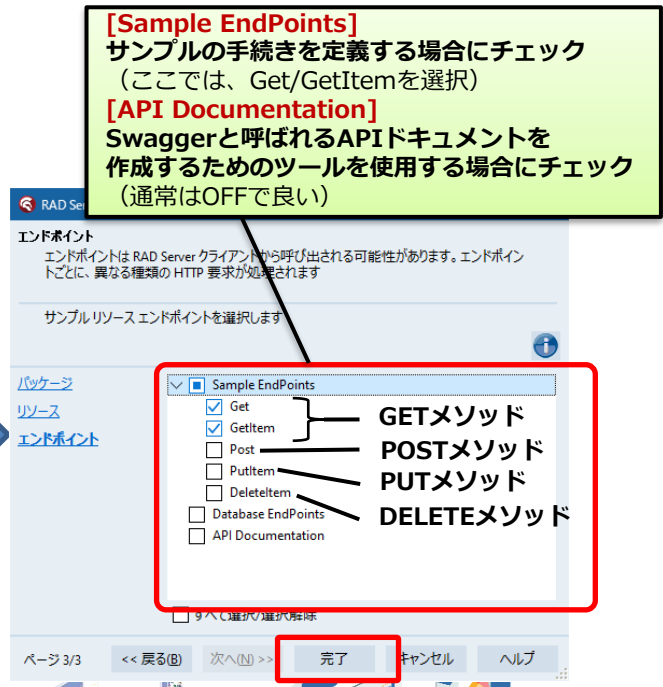


■ RAD Serverアプリ開発手順

- 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

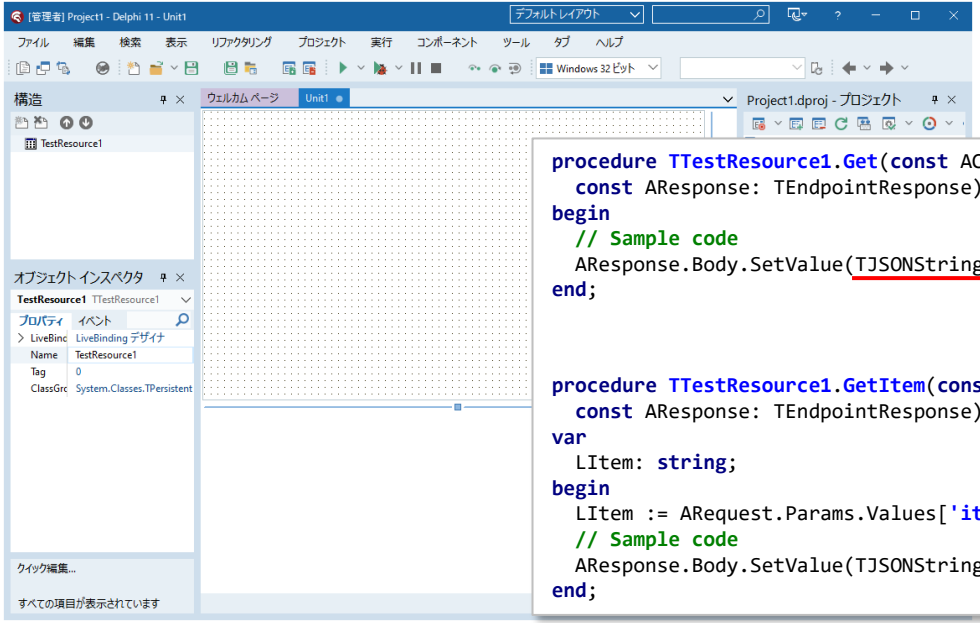






RAD Serverアプリ開発手順

- 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(TJSONString.Create('test'), True);
end;


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(TJSONString.Create('test ' + LItem), True);
end;
```

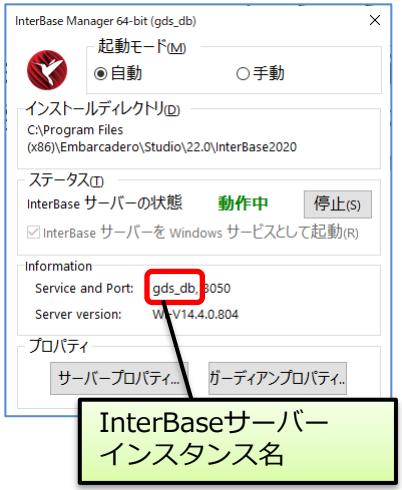
レスポンスとして、
『**test**』というJSON文字列をセットする
サンプルコード

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

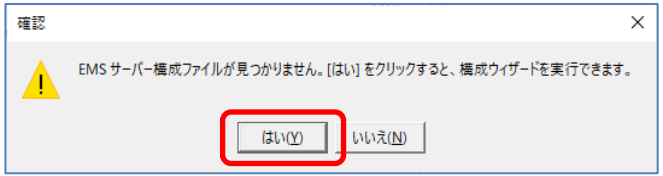
RAD Serverアプリ開発手順

- 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サーバーマネージャ）
    - 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と、[EMSサーバー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、サーバー構成が作成可能。
      （通常はサーバーインスタンス名以外は初期値で作成可能）

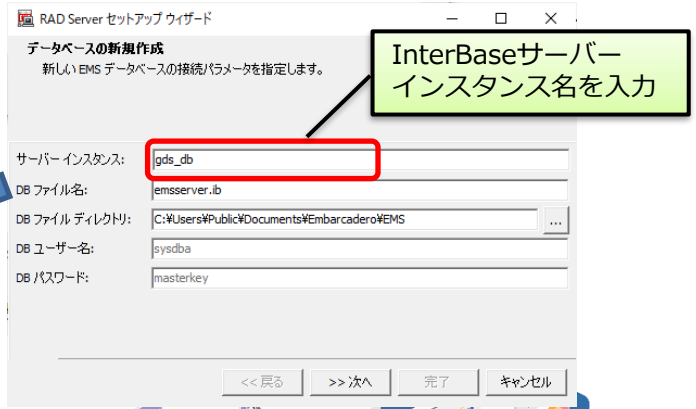
【InterBaseサーバーマネージャー】



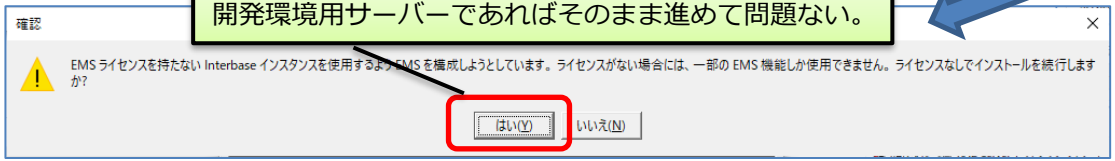
メッセージが出力



構成ウィザードを開始



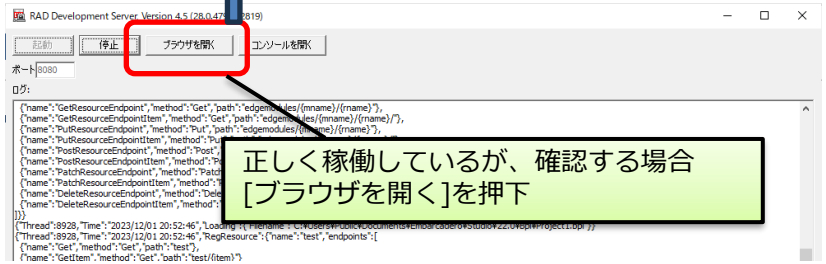
途中ライセンスが無いメッセージが出力されるが、
開発環境用サーバーであればそのまま進めて問題ない。



RAD Serverアプリ開発手順

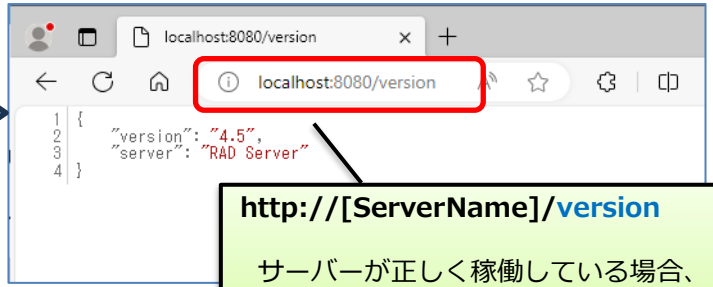
- RAD Server プロジェクトの実行
  - 開発用サーバーは、**8080ポート**で動作
  - [ブラウザを開く]で動作確認が可能

【開発環境用サーバー】



正しく稼働しているが、確認する場合
[ブラウザを開く]を押下

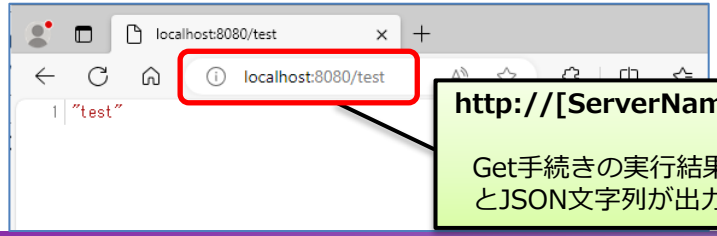
【実行結果：ブラウザ】



http://[ServerName]/version

サーバーが正しく稼働している場合、
RADサーバーのバージョン情報がJSON文字列で出力

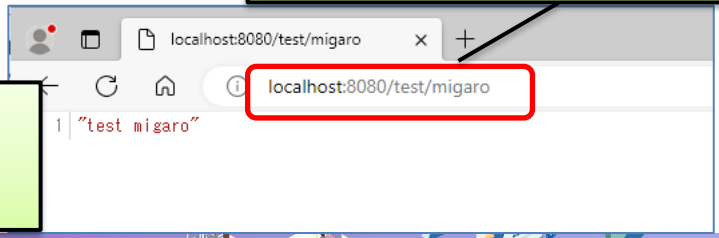
リクエスト（GETメソッド）
Get手続きが実行



http://[ServerName]/test

Get手続きの実行結果として**"test"**
とJSON文字列が出力

リクエスト（GETメソッド）
GetItem手続きが実行

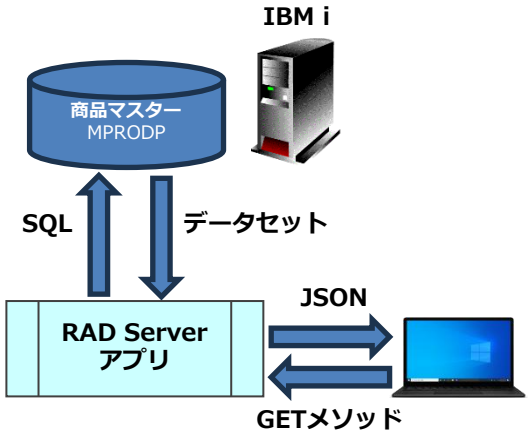


http://[ServerName]/test/migaro

GetItem手続きの実行結果として
"test migaro"とJSON文字列が出力

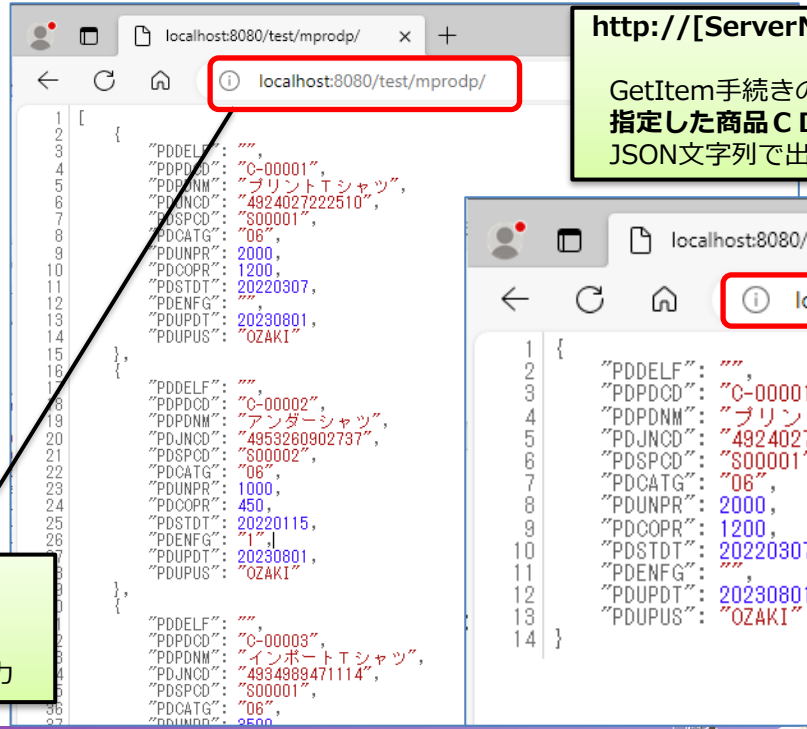
■ IBM i データにアクセスするRAD Serverアプリ(1)

- 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



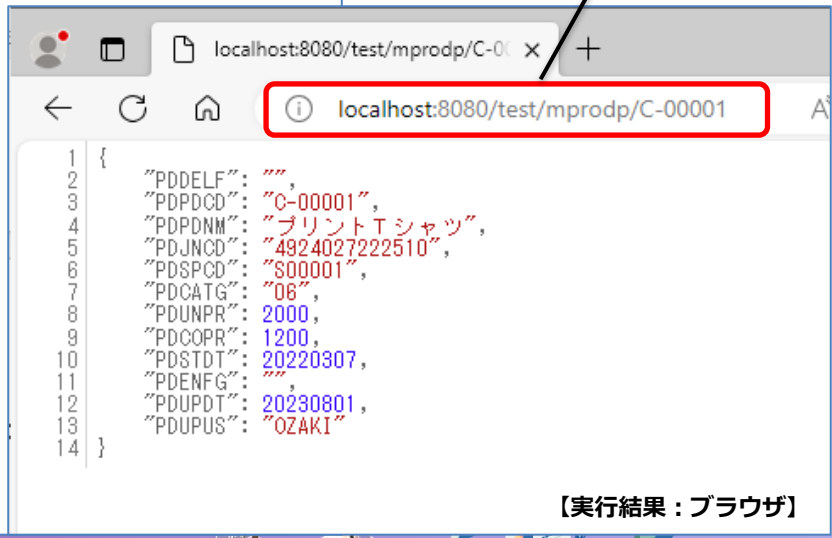
http://[ServerName]/test/mprodp/

Get手続きの実行結果として
商品マスタのデーター覧がJSON文字列で出力



http://[ServerName]/test/mprodp/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



【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

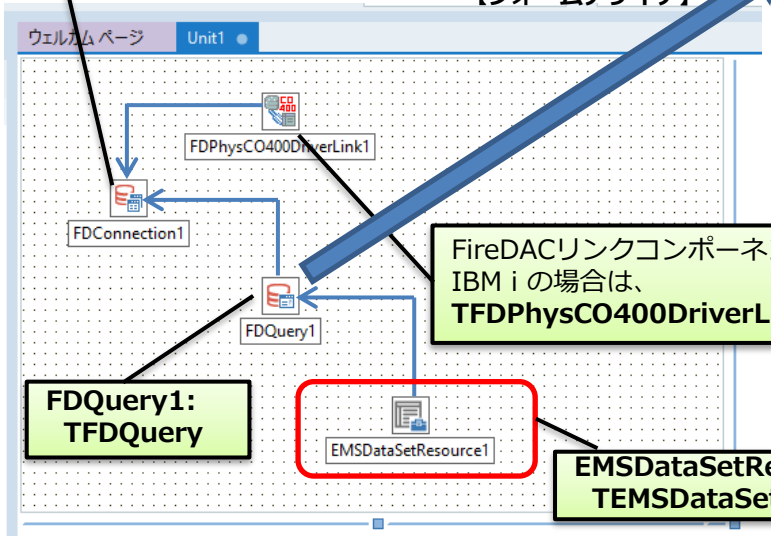
FDConnection1: TFDConnction
LoginPrompt：**False**

TFDQuery.SQLプロパティ

```
SELECT * FROM [ファイル名]
{IF %SORT} ORDER BY &SORT {FI}
```

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、このまま記述する。

【フォームデザイナ】

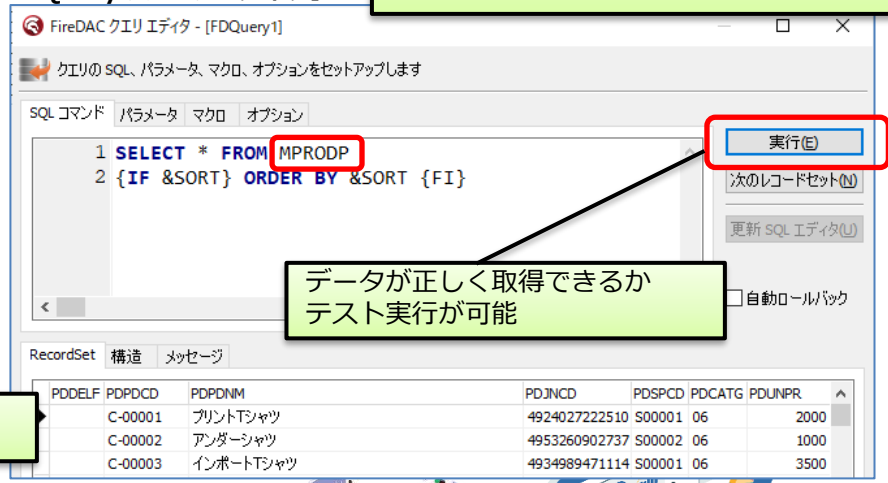


FireDACリンクコンポーネントを配置
IBM i の場合は、
TFDPhysCO400DriverLinkを使用

FDQuery1: TFDQuery

EMSDataSetResource1: TEMSDataSetResource

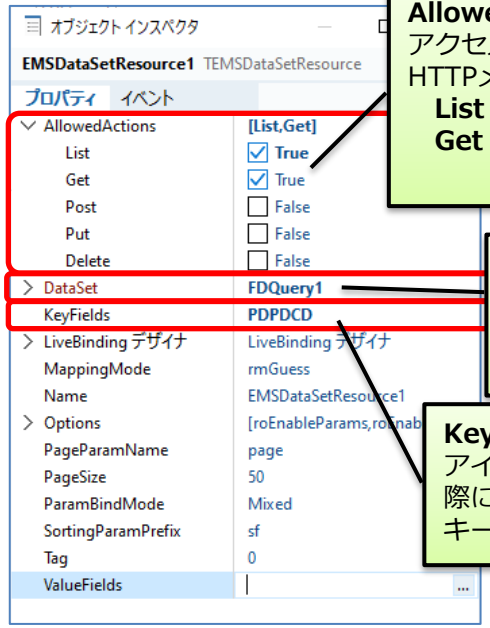
【TFDQueryのクエリエディタ】



データが正しく取得できるか
テスト実行が可能

■ 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

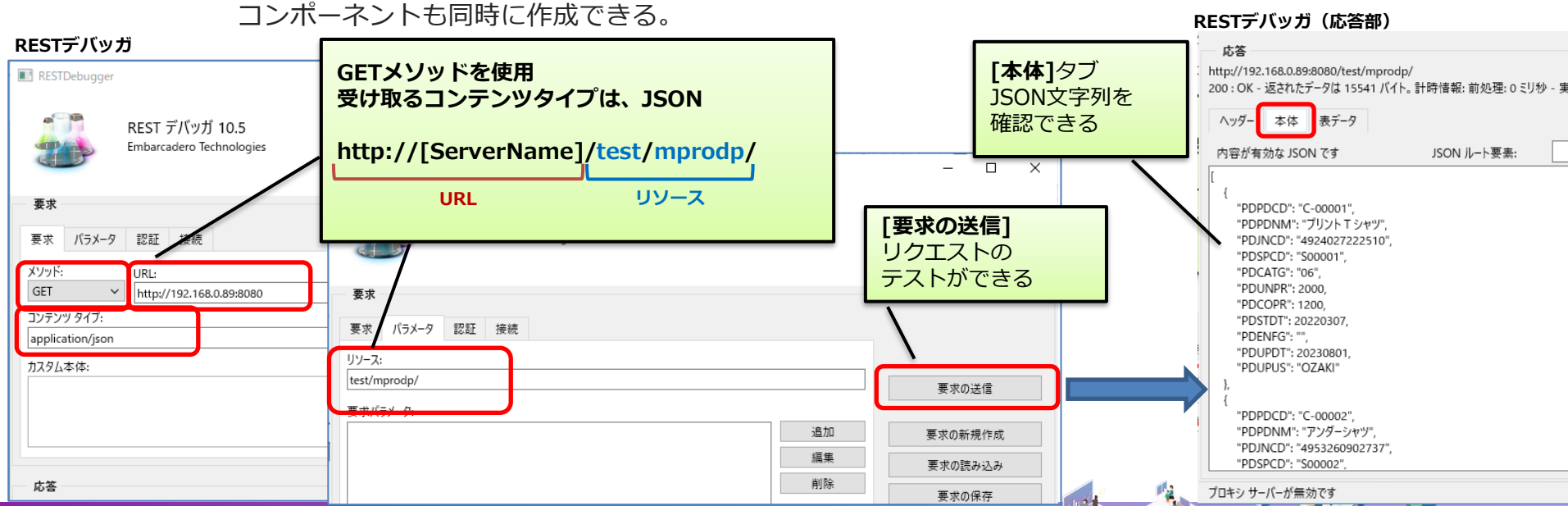
定義した**EMSDataSetResource1**の宣言部の上に、
[ResourceSuffix('mprodp')] を行追加する。
リソース名とターゲットとなるTEMSDataSetResource
コンポーネントとが関連付く

TTestResource1クラスの宣言部

```
type
  [ResourceName('test')]
  TTestResource1 = class(TDataModule)
    FDConnection1: TFDConnection;
    FDPhysCO400DriverLink1: TFDPhysCO400DriverLink;
    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;
```

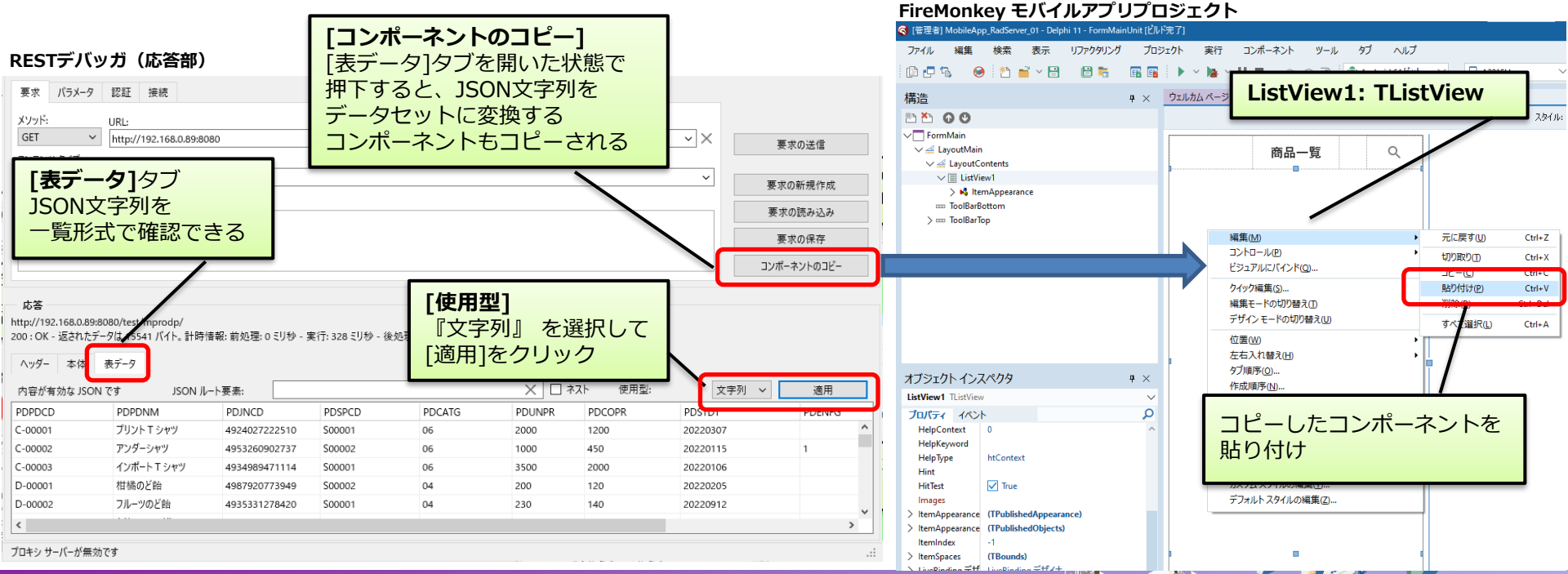
■ IBM i データにアクセスするRAD Serverアプリ(1)

- 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。



■ IBM i データにアクセスするRAD Serverアプリ(1)

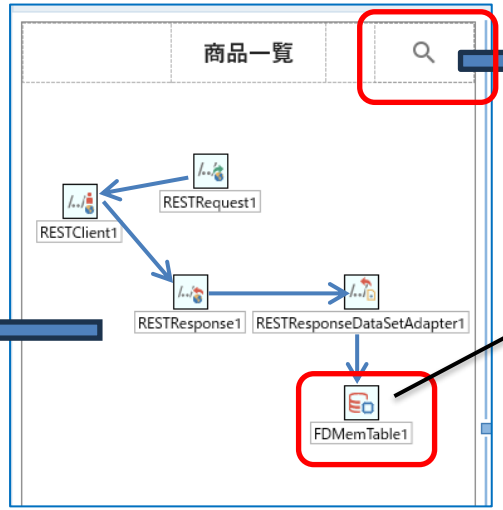
- モバイルアプリからRAD Serverへのアクセス方法
  - RESTデバッガから取得したコンポーネント定義を貼り付け



■ IBM i データにアクセスするRAD Serverアプリ(1)

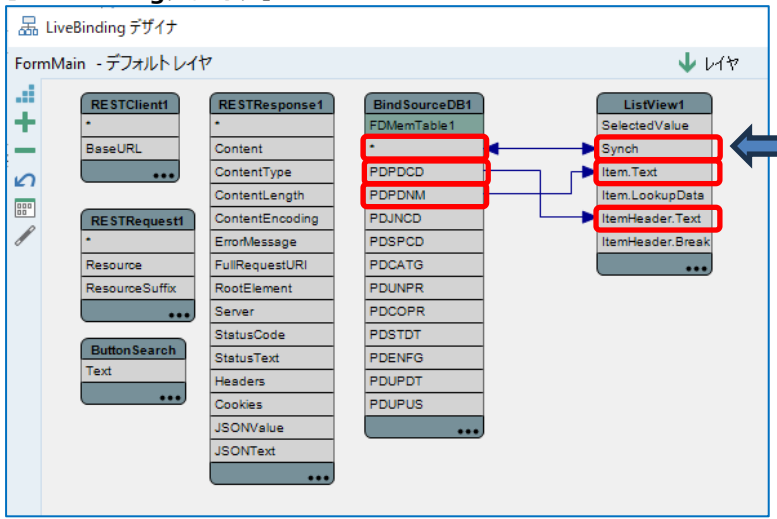
- モバイルアプリからRAD Serverへのアクセス方法
  - LiveBindingの設定
  - 実行イベントの定義

【フォームデザイナ】



FDMemTabl1: TFDMemTable
取得したJSONデータを受け取る
ローカルデータセット

【LiveBindingデザイナ】



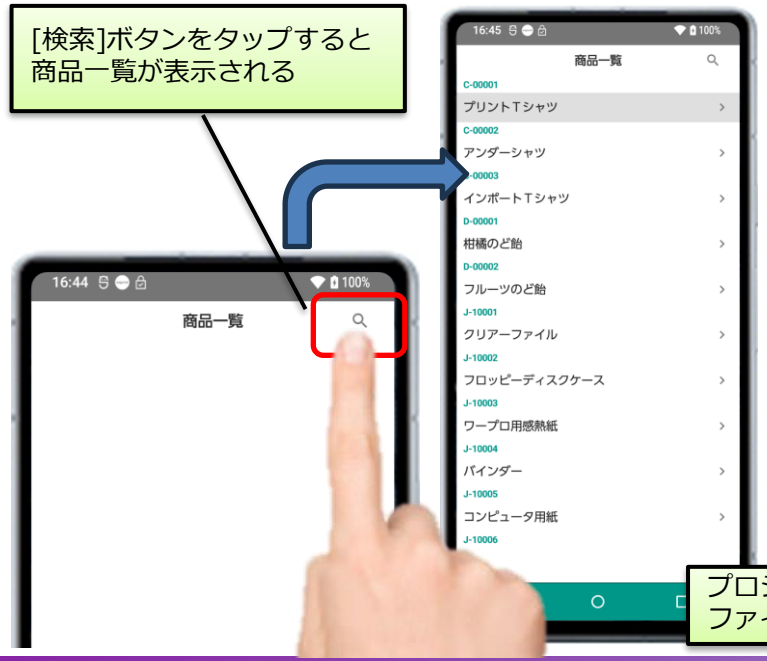
【ButtonSearchのOnclickイベント】

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

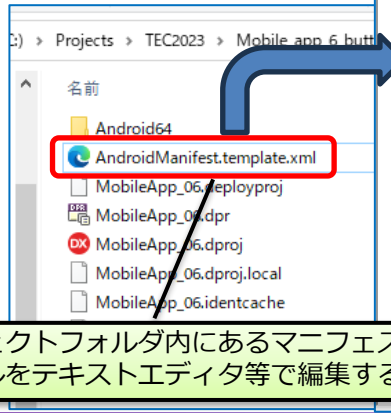
■ IBM i データにアクセスするRAD Serverアプリ(1)

- 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。



【プロジェクトフォルダ】



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

【AndroidManifest.templete.xml】

<application>タグの中に以下の設定を追加する
android:usesCleartextTraffic="true"

```
<?xml version="1.0" encoding="utf-8"?>
    <uses-sdk android:minSdkVersion="%minSdkVersion%" android:
%uses-permission%>
    <uses-feature android:glEsVersion="0x00020000" android:req
    <queries>
%queries-child-elements%>
    </queries>
    <application
        android:persistent="%persistent%"
        android:restoreAnyVersion="%restoreAnyVersion%"
        android:label="%label%"
        android:debuggable="%debuggable%"
        android:largeHeap="%largeHeap%"
        android:icon="%icon%"
        android:theme="%theme%"
        android:hardwareAccelerated="%hardwareAccelerated%"
        android:resizeableActivity="false"
        android:requestLegacyExternalStorage="true"
        android:usesCleartextTraffic="true">
ovider%>
```

■ 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】

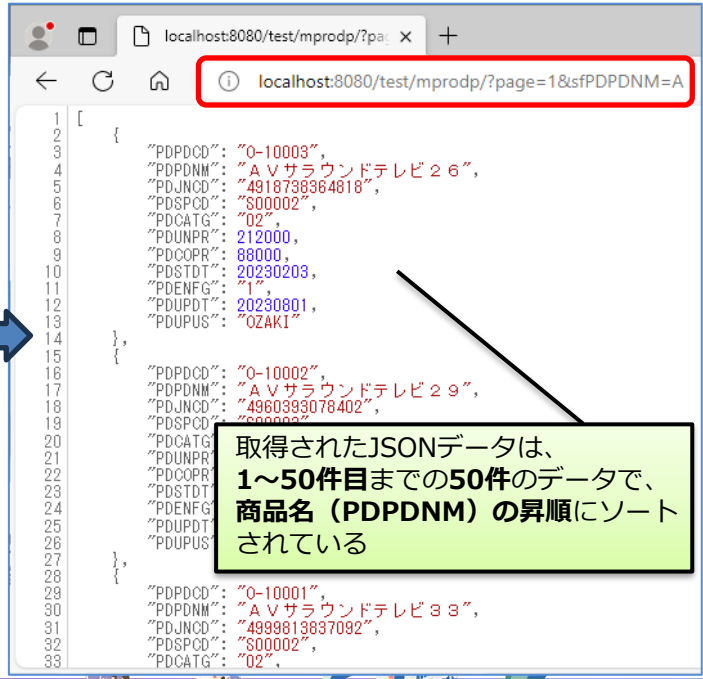
パラメータ	概要	記述例
page	**指定したページのデータのみ取得** （1ページ当りの取得レコード数は、**PageSize**プロパティで設定可能。初期値は50）	**?page=1** （1ページ目(1～50件目）のみ取得）
sf	**指定したフィールドで昇順/降順にソート** **A**：昇順 **D**：降順	**?sfPDPDNM=A** （PDPDNMの昇順に取得）

【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】



■ 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

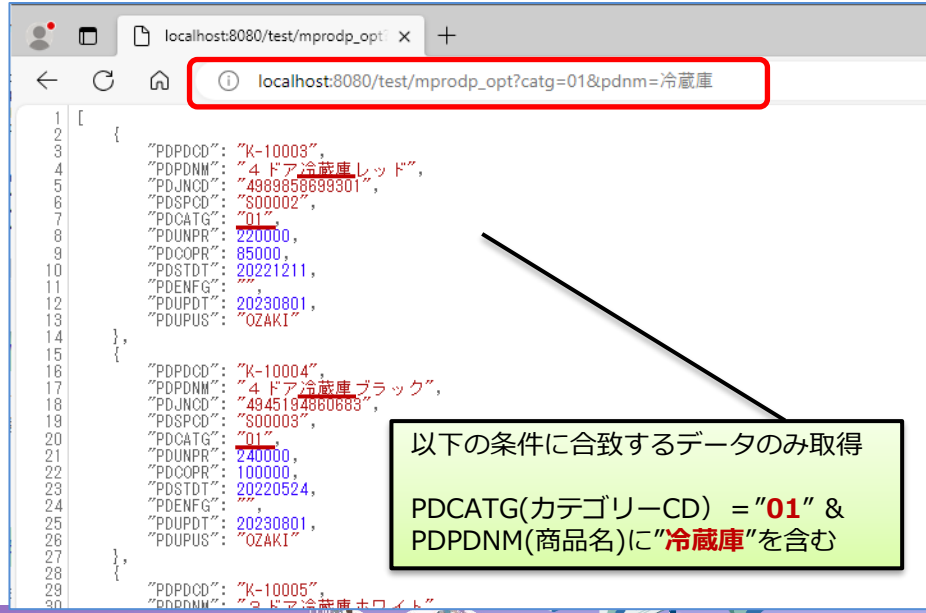
URL　リソース　商品カテゴリ　品名（部分一致）

- 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？

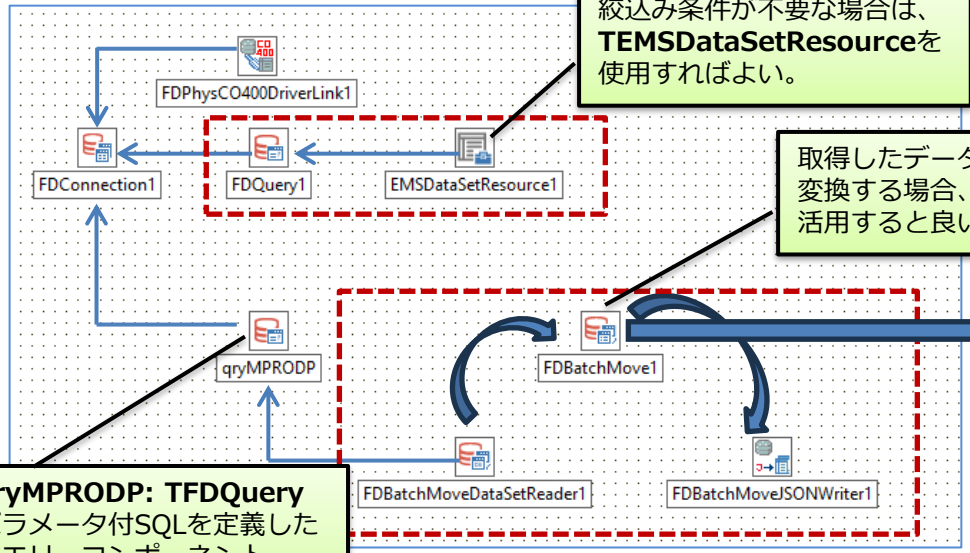
【実行結果：ブラウザ】



■ IBM i データにアクセスするRAD Serverアプリ(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】

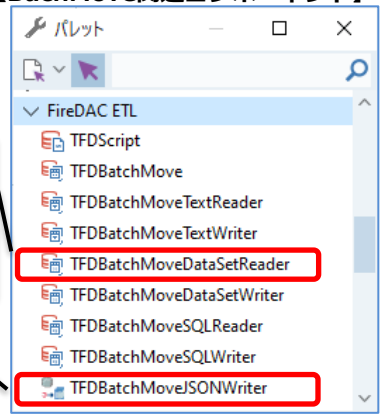


絞込み条件が不要な場合は、**TEMSDataSetResource**を使用すればよい。

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

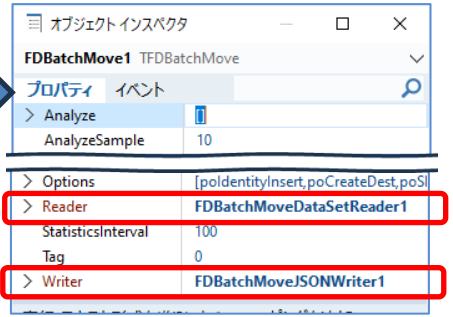
qryMPRODP: TFDQuery
パラメータ付SQLを定義したクエリーコンポーネント

【BachMove関連コンポーネント】



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



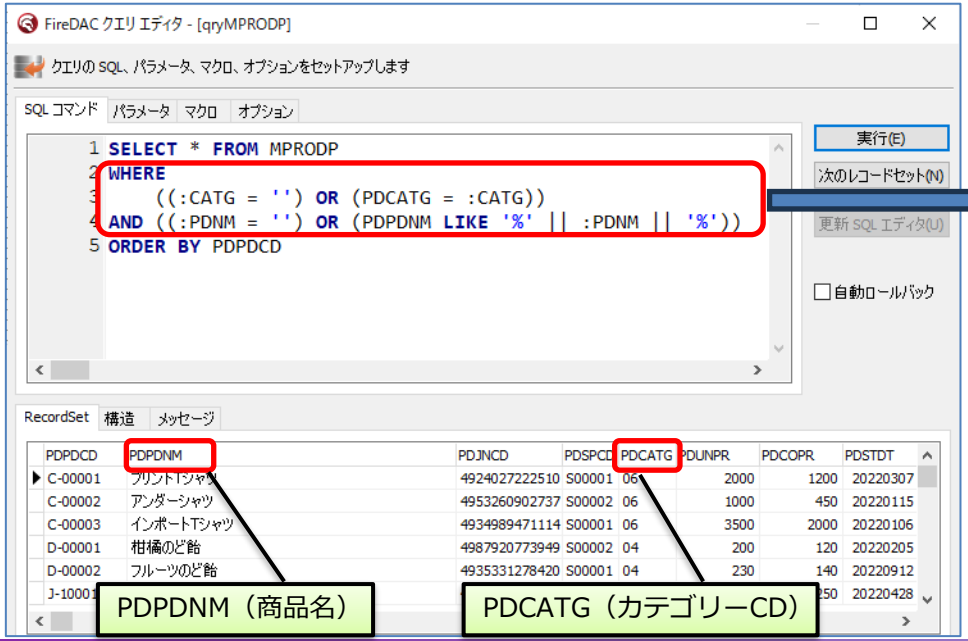
Readerプロパティ
コピー元となるリーダーコンポーネントを指定

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

■ 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】



パラメータの値が
ブランクでも、ブランク以外でも一つのSQL文で処理が可能

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

AND

パラメータ**”CATG”=ブランク**の場合は、
全てのデータを抽出する
パラメータ**”CATG”≠ブランク**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: TFDConnection;
    FDPhysCO400DriverLink1: TFDPhysCO400DriverLink;
    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
※[Crtl]+[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（存在しない場合ブランク）
  if not ARequest.Params.TryGetValue('catg', sCATG) then sCATG := '';
  if not ARequest.Params.TryGetValue('pdnm', 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('PDNM').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

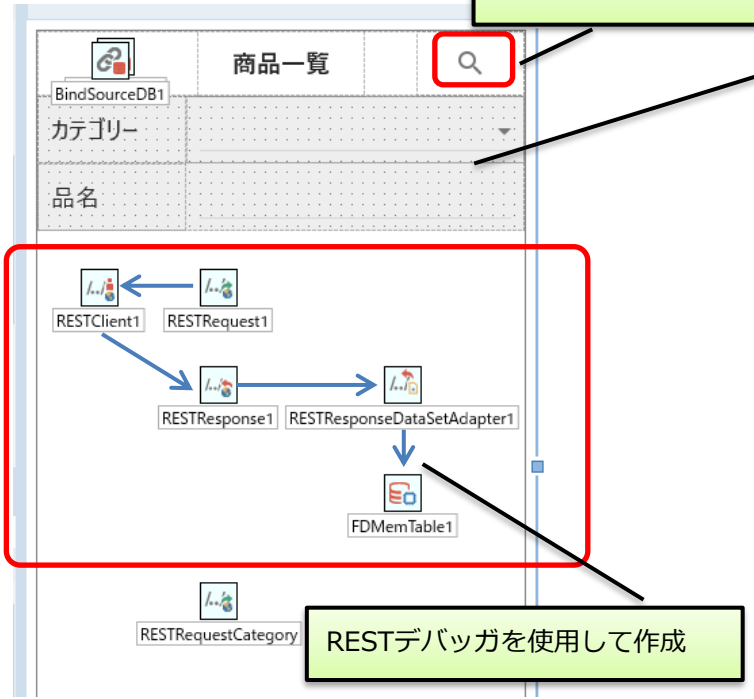
JSONライターの書き出し先に、
Aresponse.Body.JSONWriterを指定すればよい。

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】



ButtonSearch: TButton

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

RESTデバッガを使用して作成

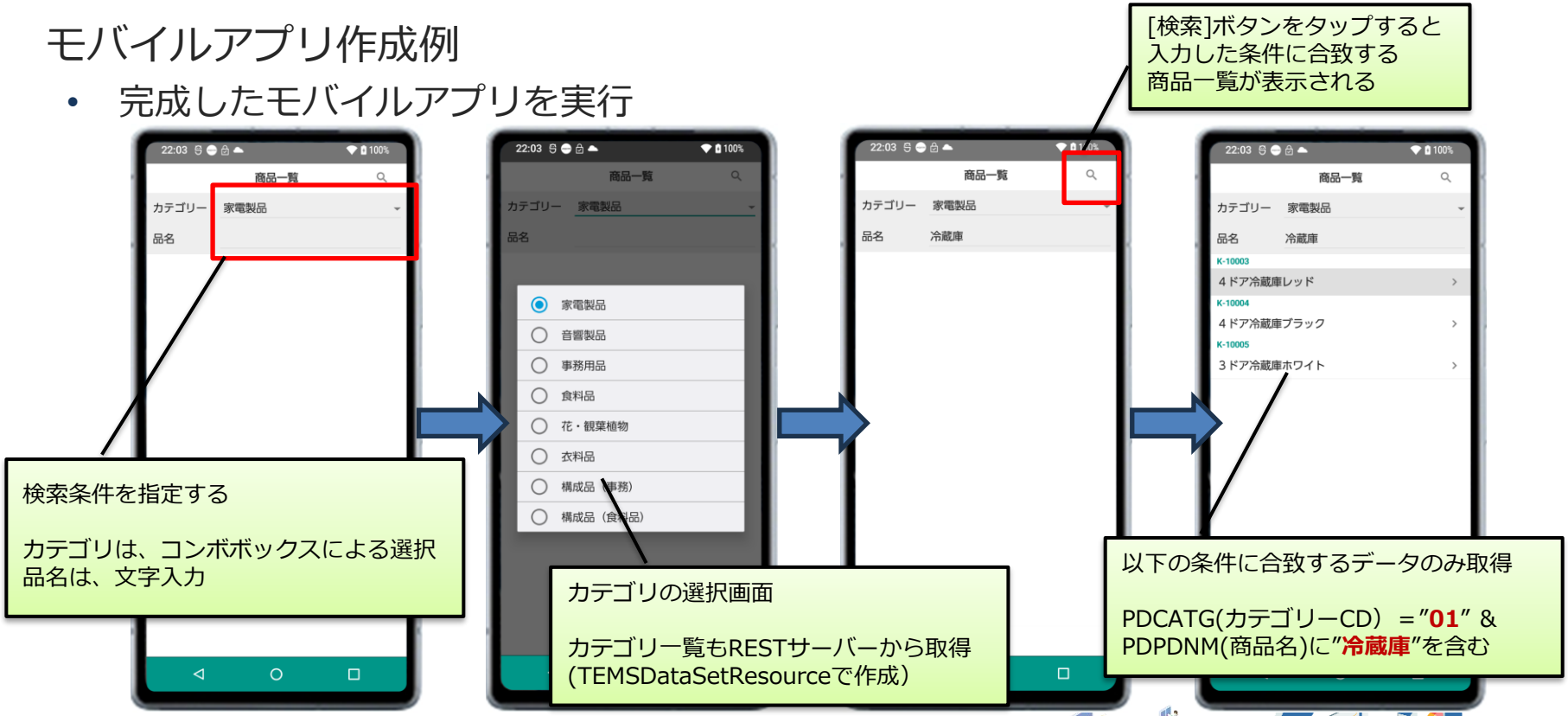
実装部（一部抜粋）

```
procedure TFormMain.ButtonSearchClick(Sender: TObject);
var
  sCATG: String;  //カテゴリー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ー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	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－	XE5を使用した基本的なモバイルアプリ開発手法
第15回	マルチデバイスに対応したIBMi業務システム開発のポイント	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	実践！iOS / Android ネイティブ機能開発	バーコード読取りや、手書き署名、オフライン保存方法
第20回	VCL開発者が知っておきたいFireMonkeyアプリ開発のポイント	VCLとFireMonkeyの違い、各種コンポーネントについて
第21回	ステップアップ！モバイルアプリケーション開発	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	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	DataSnapやRADServer構築方法
第23回	レスポンス向上！モバイルアプリ実践開発テクニック	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