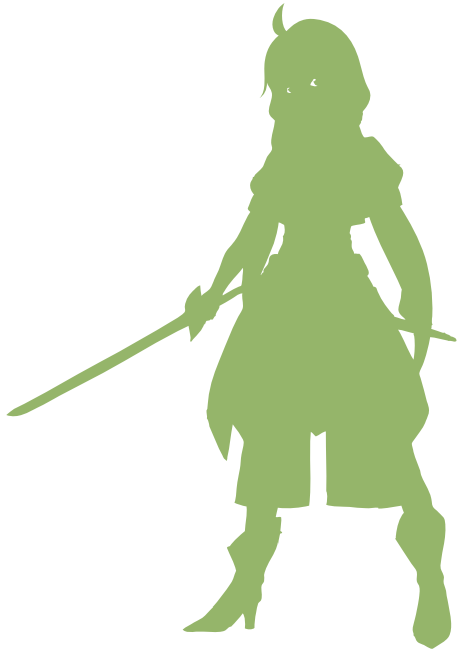
モバイルからクラウドサービスまで 活用が広がる！ Delphiではじめる多層アプリ開発

第35回 エンバカデロ・デベロッパーキャンプ

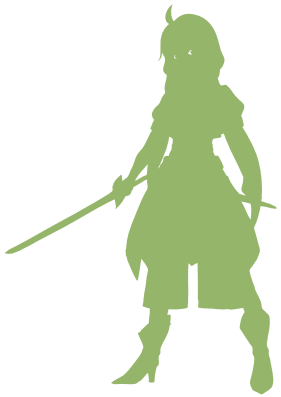
株式会社ミガロ.
RAD事業部 技術支援課 吉原 泰介



embarcadero®
DEVELOPER CAMP

はじめに

- 中間サーバを使った多層アプリケーション開発はされていますか？
  仕組みが面倒だと思われがちですが、実装するメリットも沢山あります。
  もちろんモバイルアプリケーションだけではなく、
  デスクトップアプリケーション（もちろんVCLも！）にも有効な手法です。

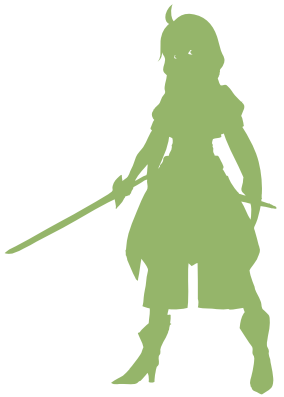


embarcadero®
DEVELOPER CAMP

アジェンダ

- 多層アプリケーションとは？
- 中間サーバのアプリケーションを開発する手法
- DataSnap Server を使った実装手順
- RAD Serverを使った実装手順
- 中間サーバの活用範囲を拡張
  - Enterprise Connectorsを利用したクラウドサービス等の連携
  - Sencha等のWebアプリケーションへの連携

多層アプリケーションとは？



embarcadero®
DEVELOPER CAMP

多層アプリケーションとは？

- 多層アプリケーションとは、複数層で構成されたアプリケーションのこと。例えばClient/Serverアプリケーションは、クライアント層とデータ層で処理される2層構成、Webアプリケーションやモバイルアプリケーションでデータベースにアクセスする場合は、中間層のサーバを経由する3層構成となっている。

多層アプリケーションとは？

- よく使われるアプリケーションの構成

【2層アプリケーションの構成】

Client/Serverアプリケーション

【3層アプリケーションの構成】

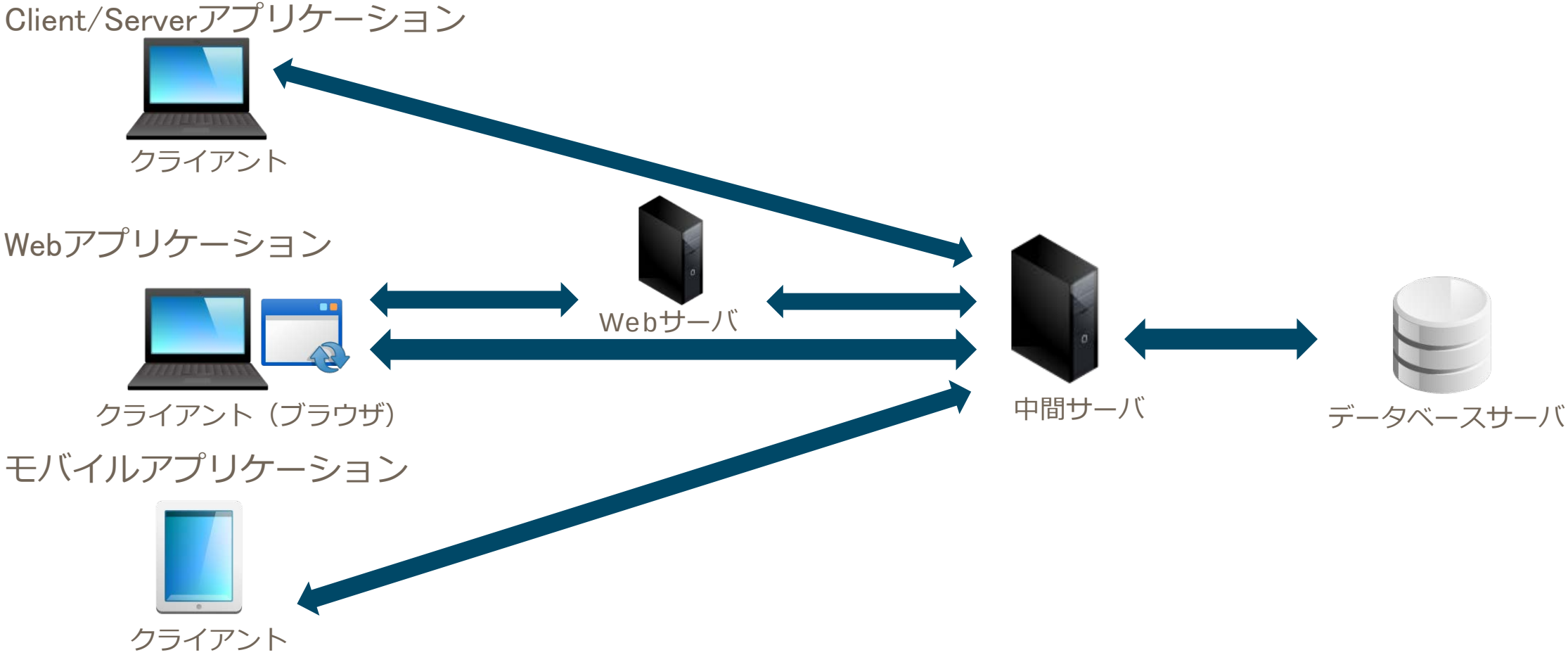
Webアプリケーション

モバイルアプリケーション

多層アプリケーションとは？

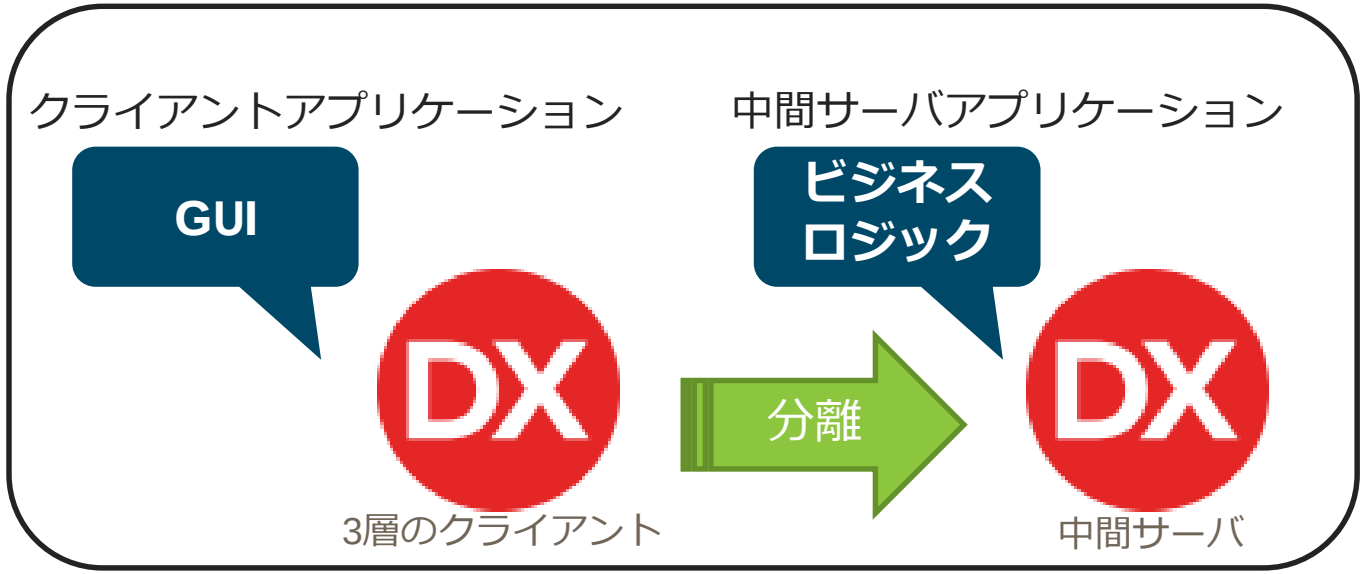
- 中間サーバを活用したアプリケーション構成

【中間サーバを活用したアプリケーションの構成例】



中間サーバを経由する構成のメリット

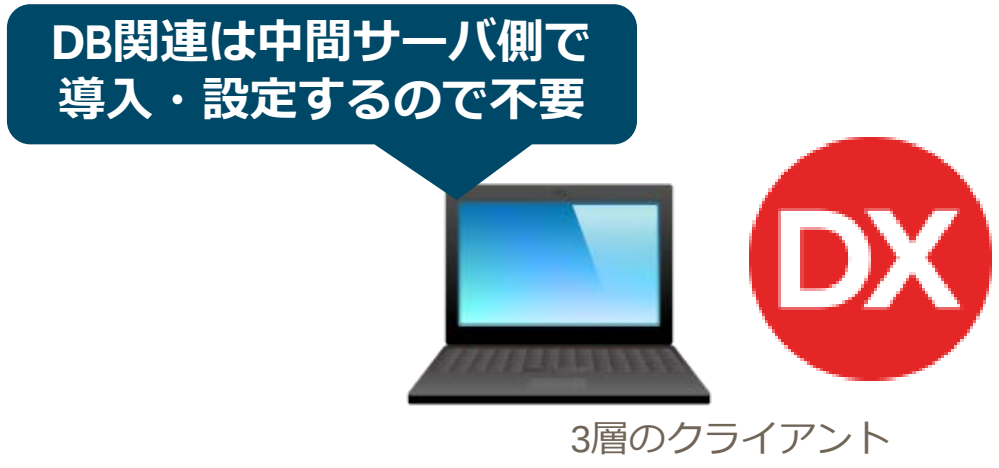
- メリット①
  ビジネスロジックの分離
  画面などのGUI部分とデータ側の業務処理を含むビジネスロジック部分を
  分離して開発・メンテナンスすることができる。
  これによりアプリケーションの実装内容は目的別のシンプルな内容になる。



中間サーバを経由する構成のメリット

- メリット②
  クライアントアプリケーションのダウンサイズ・配布の軽減
  ユーザー環境へ配布するクライアントアプリケーションは
  ビジネスロジックを含まないため、実行ファイルのサイズを
  小さく抑えることができ、変更や再配布の頻度も減らすことができる。
  （ビジネスロジック側の変更でクライアントアプリケーションは変わらない）
  また使用するDB関連の導入や設定等もクライアント側では不要になる。

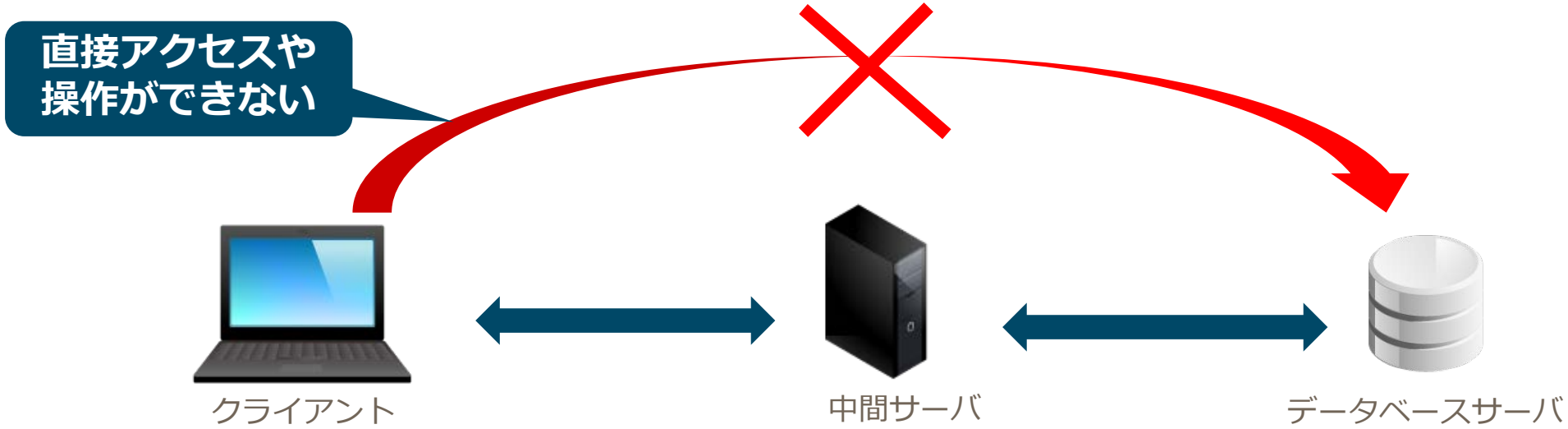
2層のクライアント



3層のクライアント

中間サーバを経由する構成のメリット

- メリット③
  セキュリティ
  クライアント環境からDBに直接アクセスできない構成で環境を構築できる。
  特に外部からも利用する環境を考える場合には、中間サーバの環境で
  セキュリティ上の配慮・対策がしやすい。
  またデータ通信もHTTPS等による暗号化データでやりとりも可能。

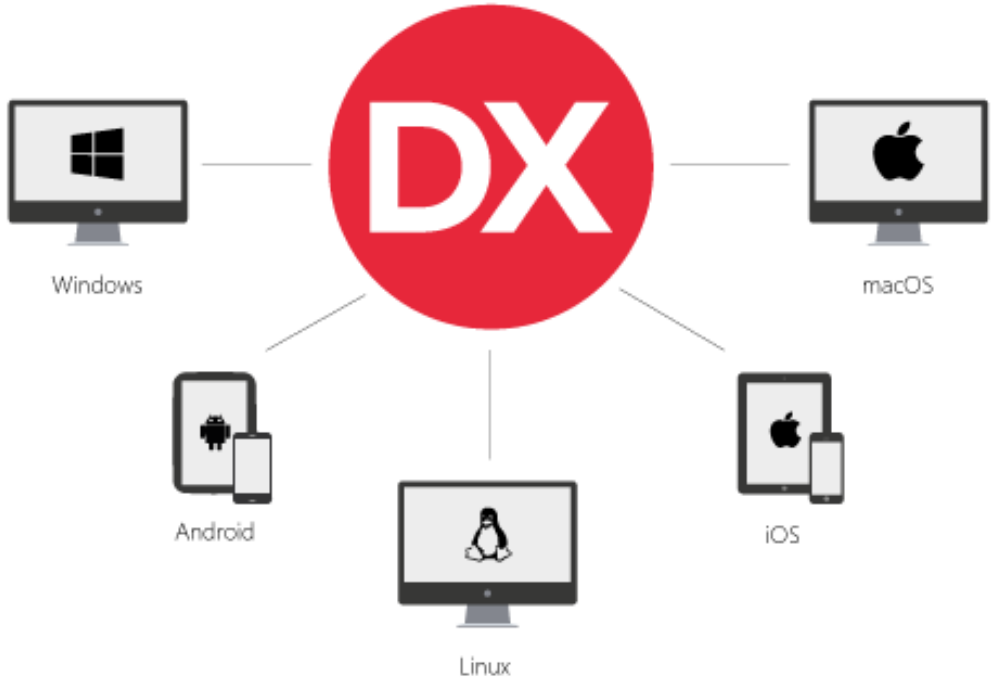


中間サーバを経由する構成のメリット

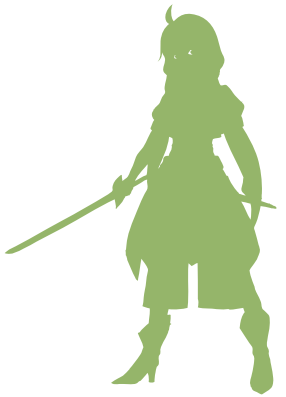
- メリット④
  マルチデバイス対応・展開
  ビジネスロジックを中間サーバ側の
  アプリケーションで持つことで、
  クライアントアプリケーションがデバイス毎に
  異なっても共通で対応できる。
  （同じビジネスロジックをデバイス毎に作成不要）

  Delphiでマルチデバイス開発したアプリケーション
  であればワンソースでの管理もでき、中間サーバ側
  との親和性も高い。

**Delphiで開発したマルチデバイスアプリは
中間サーバアプリと親和性が高い**



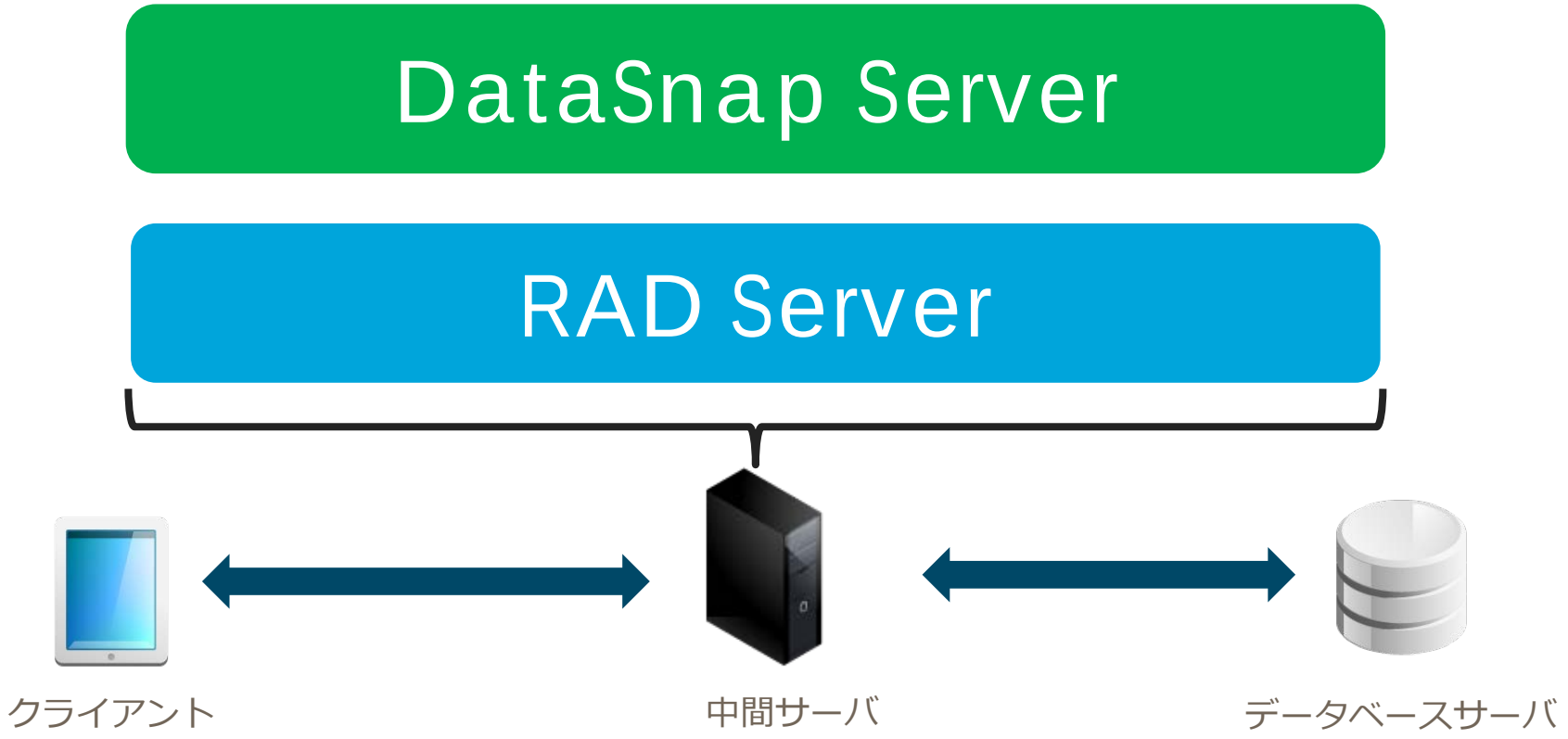
中間サーバのアプリケーションを開発する手法



embarcadero®
DEVELOPER CAMP

中間サーバのアプリケーションを開発する手法

- Delphiを使って中間サーバ側のアプリケーションを開発する手法はいくつかありますが、今回は主な2手法をご紹介。



DataSnap Serverとは？

- 特徴

・多層アプリケーションの開発を可能にするSDK

・サーバ機能はプログラムで開発する必要がある

・開発での実装となるため、プログラムの自由度が高い

・ TCP/IP 、 HTTP（S)、REST、JSON、ＣＯＭなどの標準技術をサポート

RAD Serverとは？

- 特徴

・多層アプリケーションのためのREST APIを構築／公開するサーバー

・サーバで必要となる高度な機能がいくつも提供されている

・ユーザー管理機能、認証機能、分析機能などの標準機能を豊富に搭載

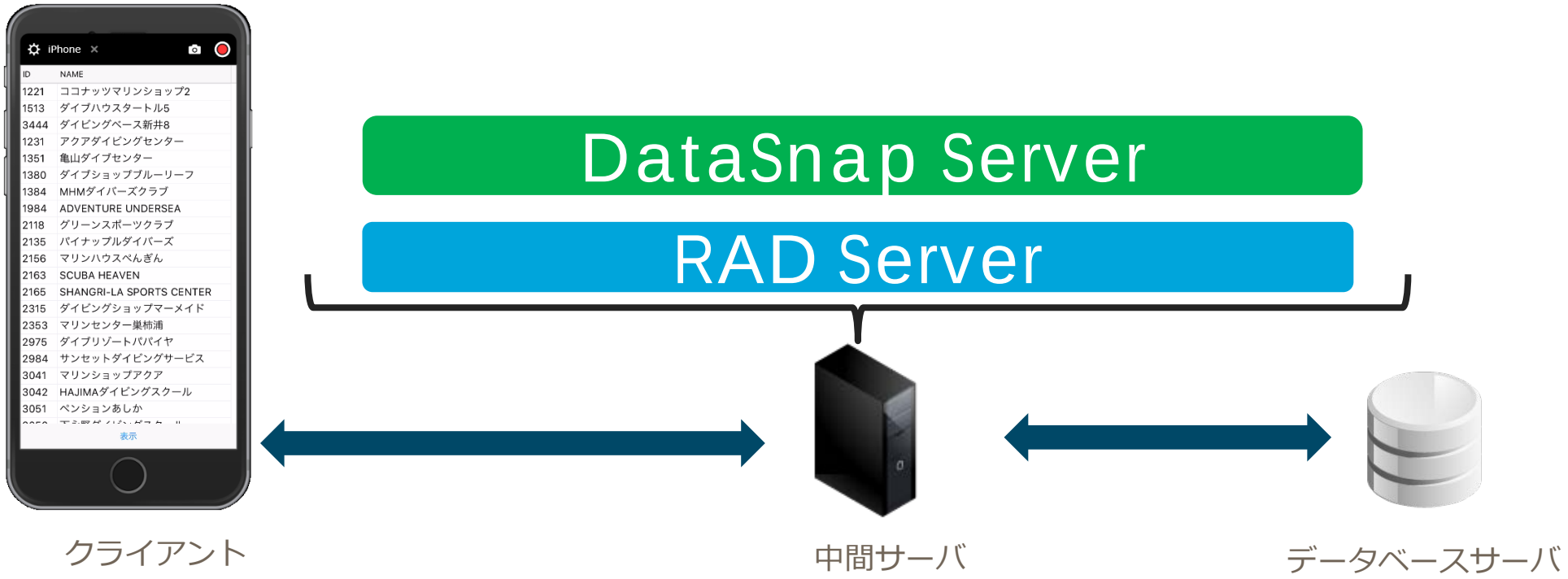
・ HTTP（S)、 REST、JSONなどの標準技術をサポート

DataSnap Server と RAD Serverの主な違い

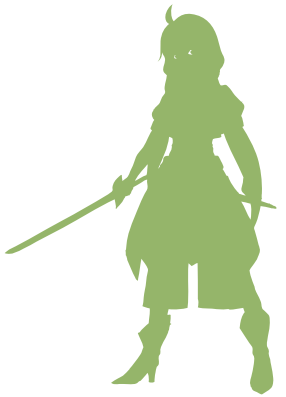
	DataSnap Server	RAD Server
機能開発	全て開発で実装が必要	必要な部分のみ開発
標準通信	TCPIP/HTTP(S)	HTTP(S)
DBエンジン	FireDAC、dbExpress	FireDAC
モバイル対応機能	開発が必要	Push通知、デバイス認証等が標準機能
管理ツール	開発が必要	標準で付属（分析も可能）
ライセンス	開発ライセンスに含まれる （Enterprise以上）	**開発ライセンスに1サイトライセンス付属 （Tokyo Enterprise以降）**

DataSnap Server と RAD Serverの実装手順

- DataSnap ServerとRAD Serverでは中間サーバで担う役割は似ているが、実装内容や手順は異なる。
  単純なデータアクセスを題材に実装手順の違いをご説明。

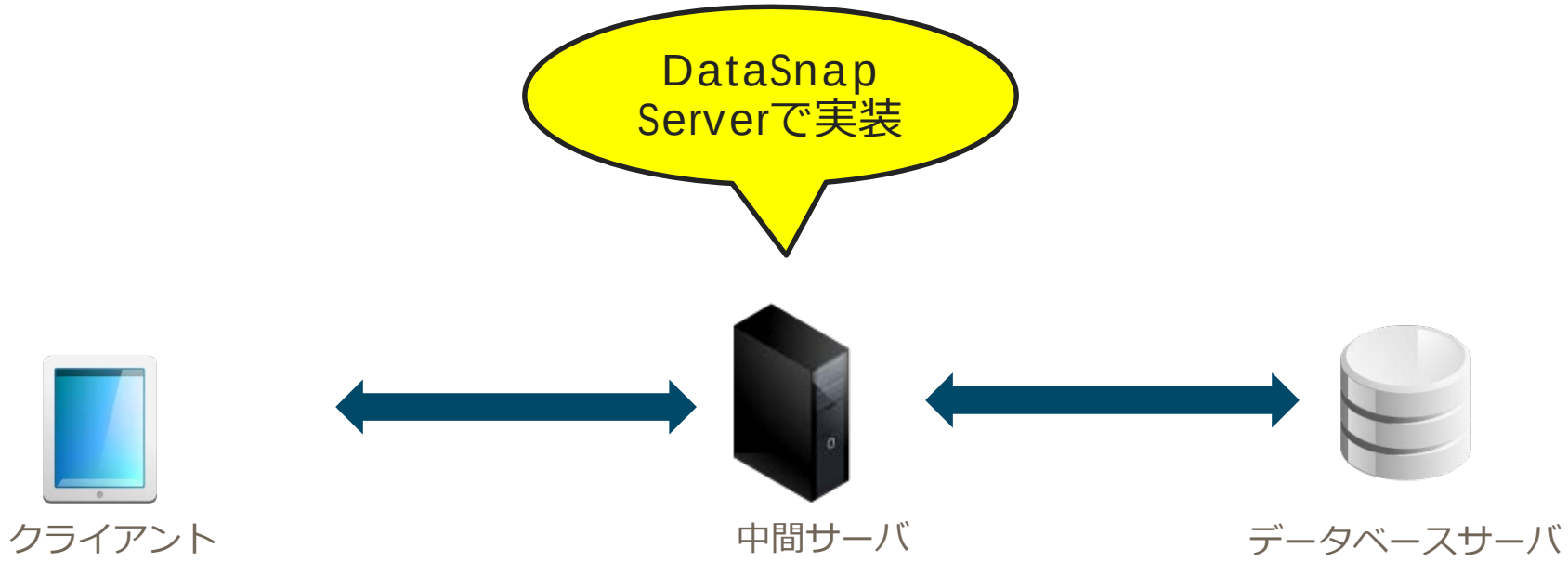


DataSnap Server を使った実装手順



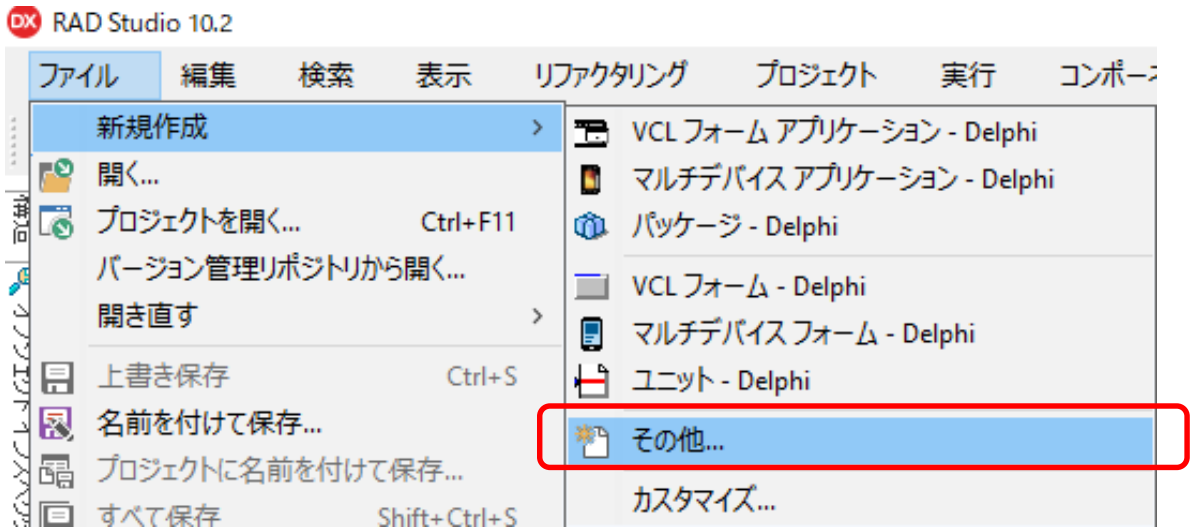
embarcadero®
DEVELOPER CAMP

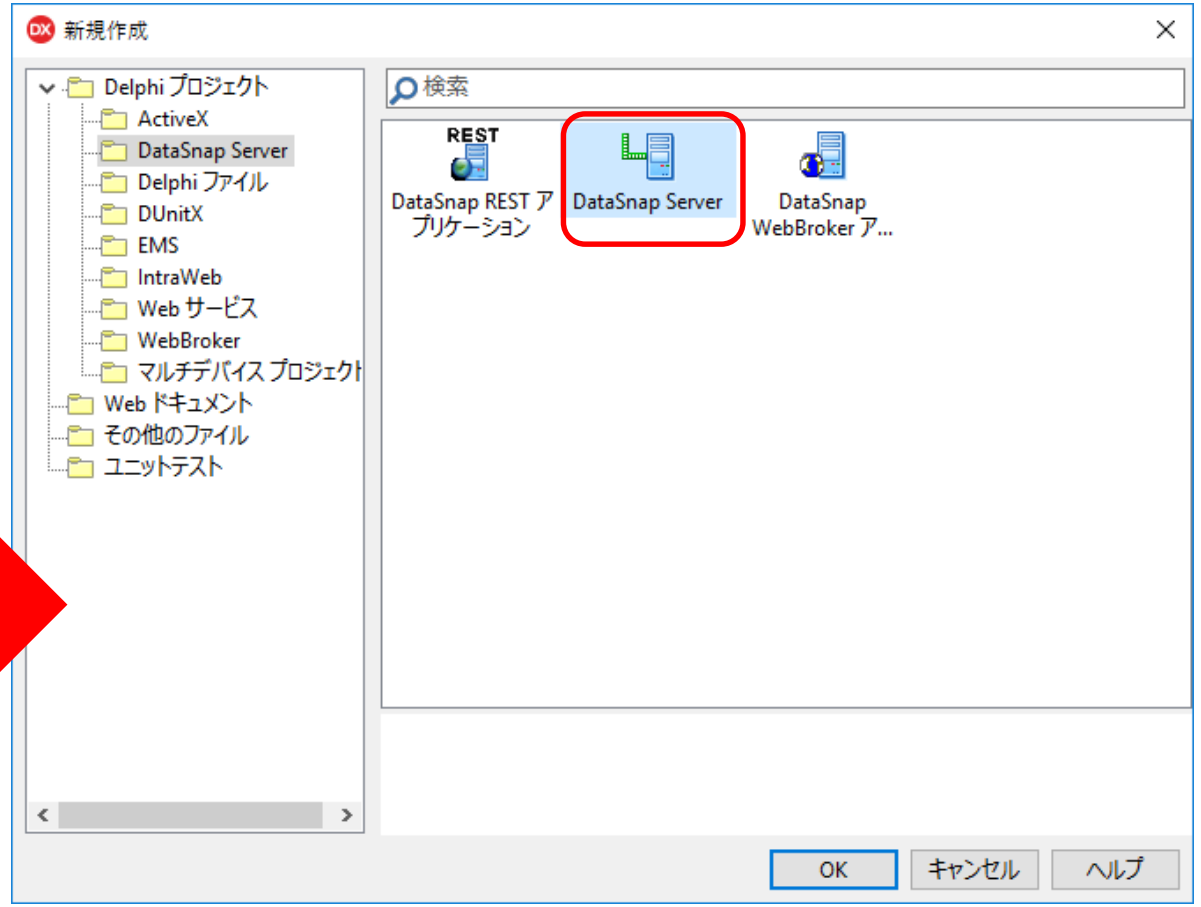
DataSnap Serverを使った実装手順（ウィザード）



DataSnap Serverを使った実装手順（ウィザード）

- 実装手順1
  プロジェクトの作成

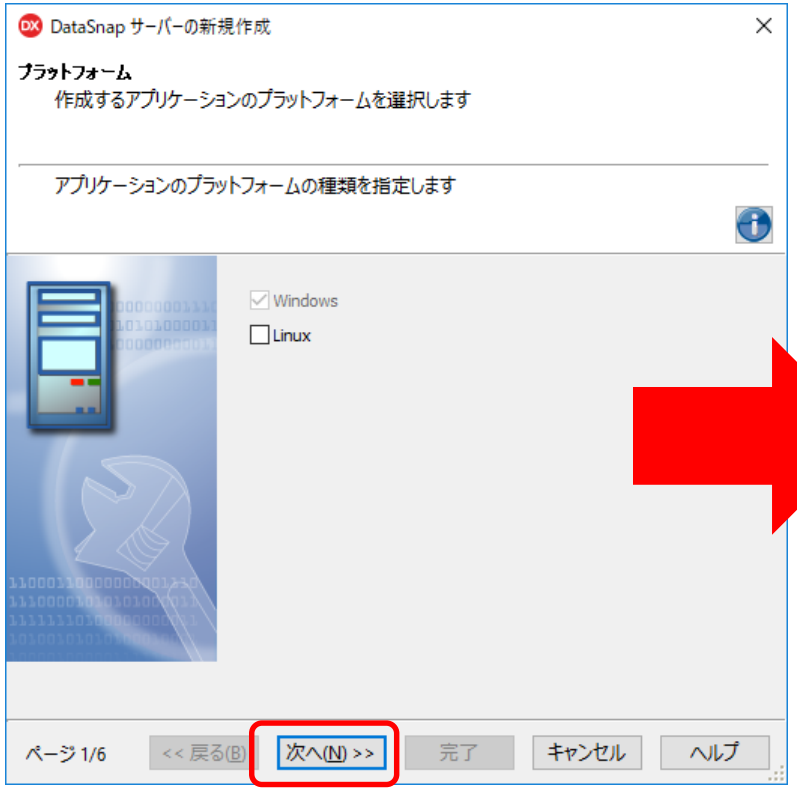


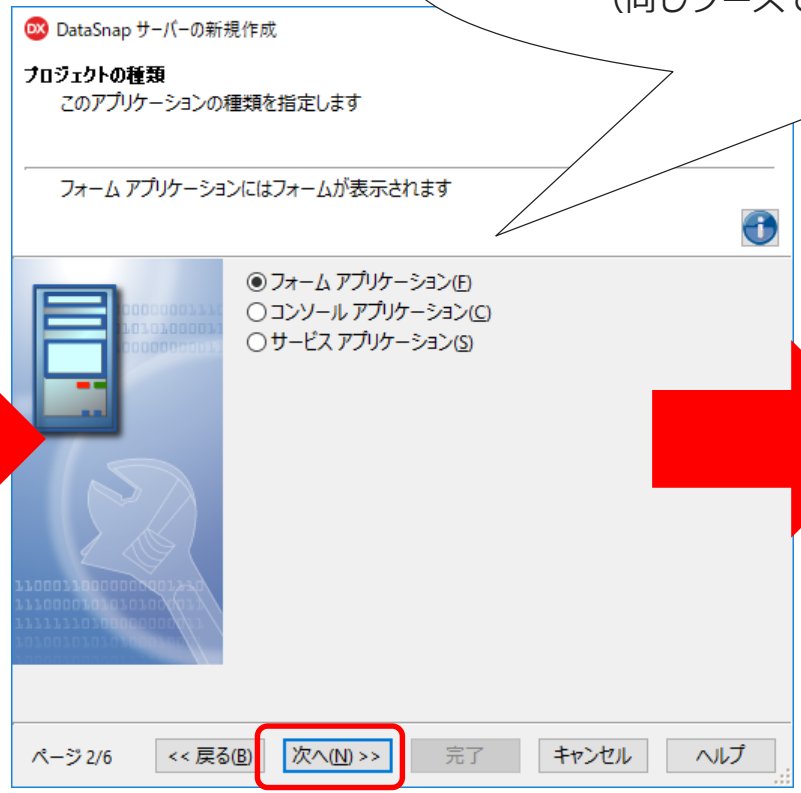


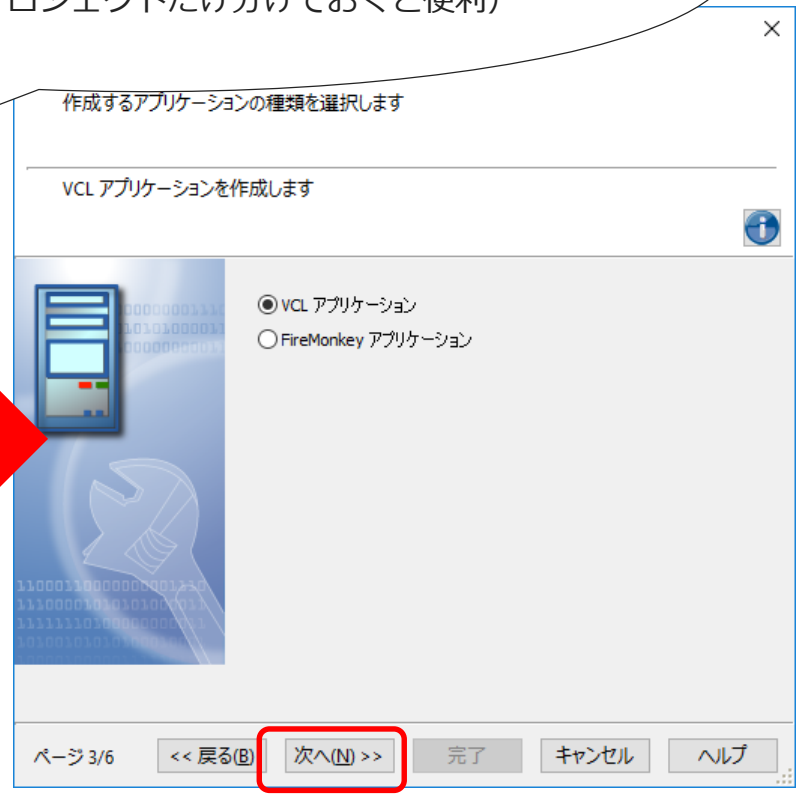
DataSnap Serverを使った実装手順（ウィザード）

- 実装手順2
  ウィザードの指定

テスト時はフォームアプリケーションが手軽だが、運用時はサービスアプリケーションが現実的
（同じソースでプロジェクトだけ分けておくと便利）

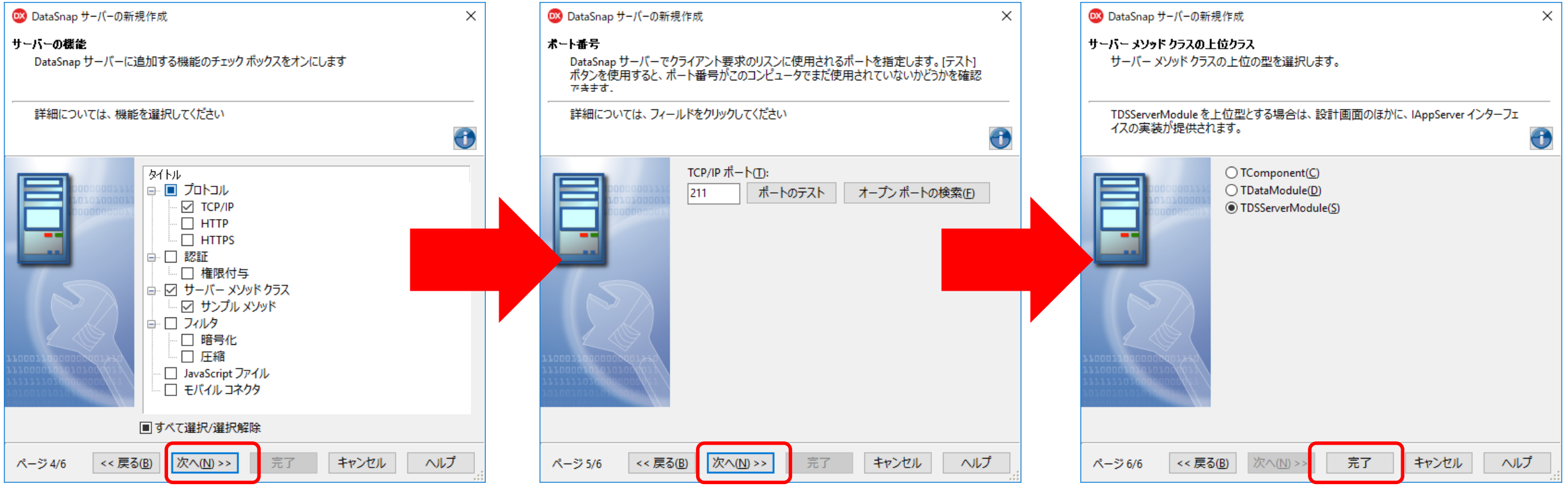






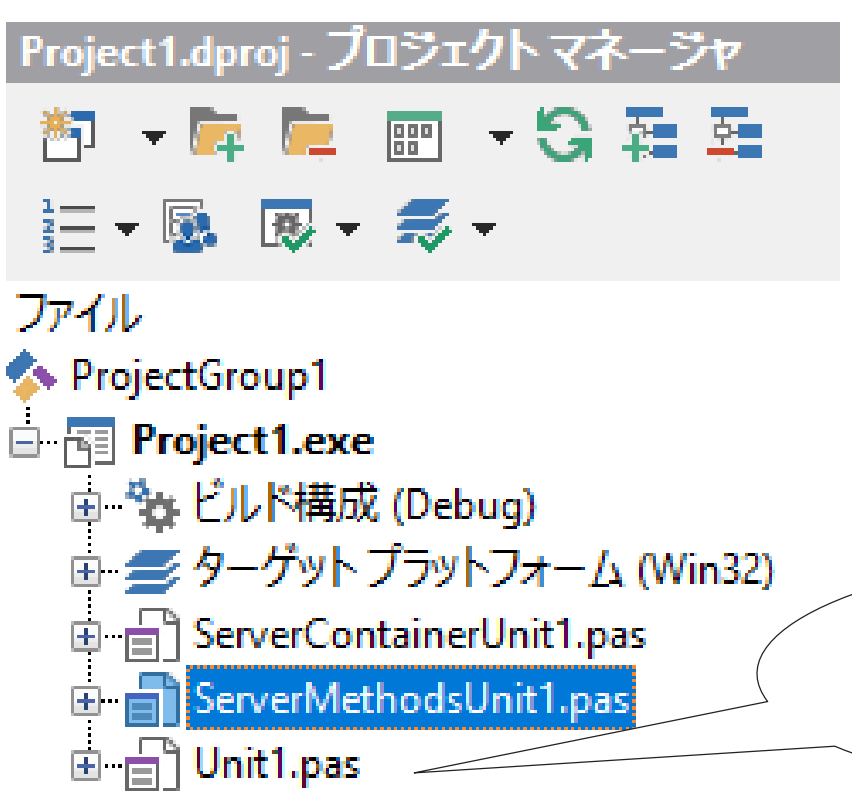
DataSnap Serverを使った実装手順（ウィザード）

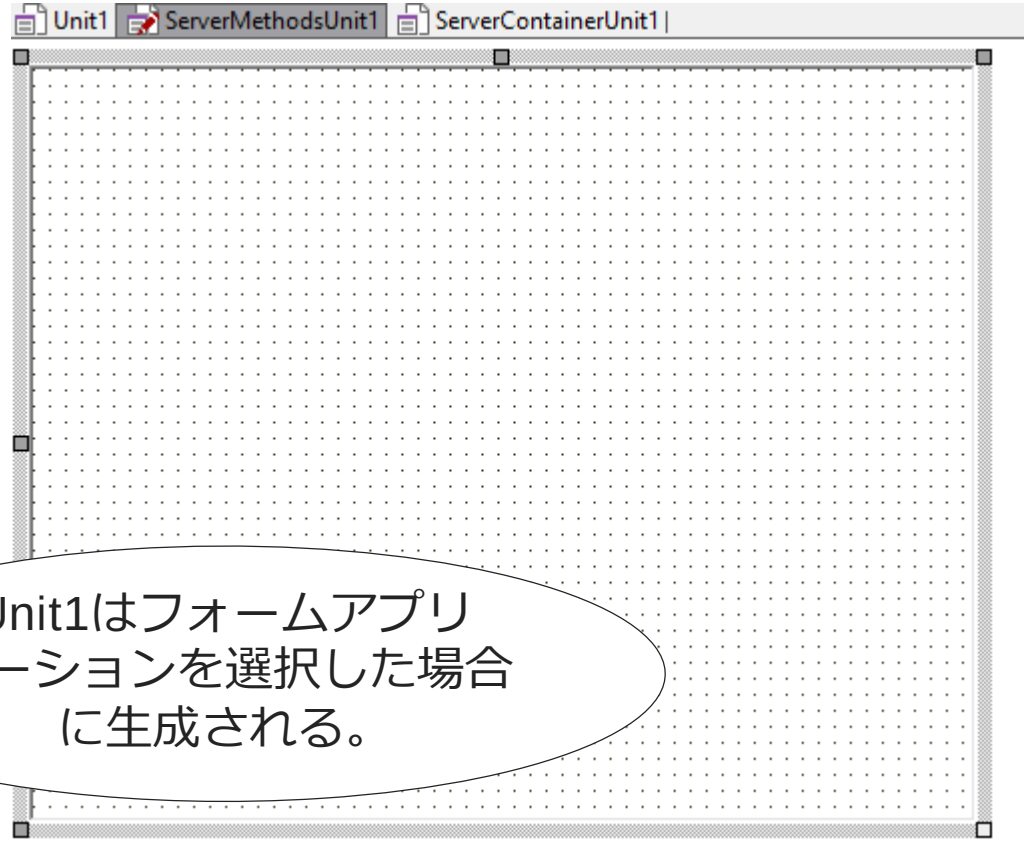
- 実装手順3
  ウィザードの指定



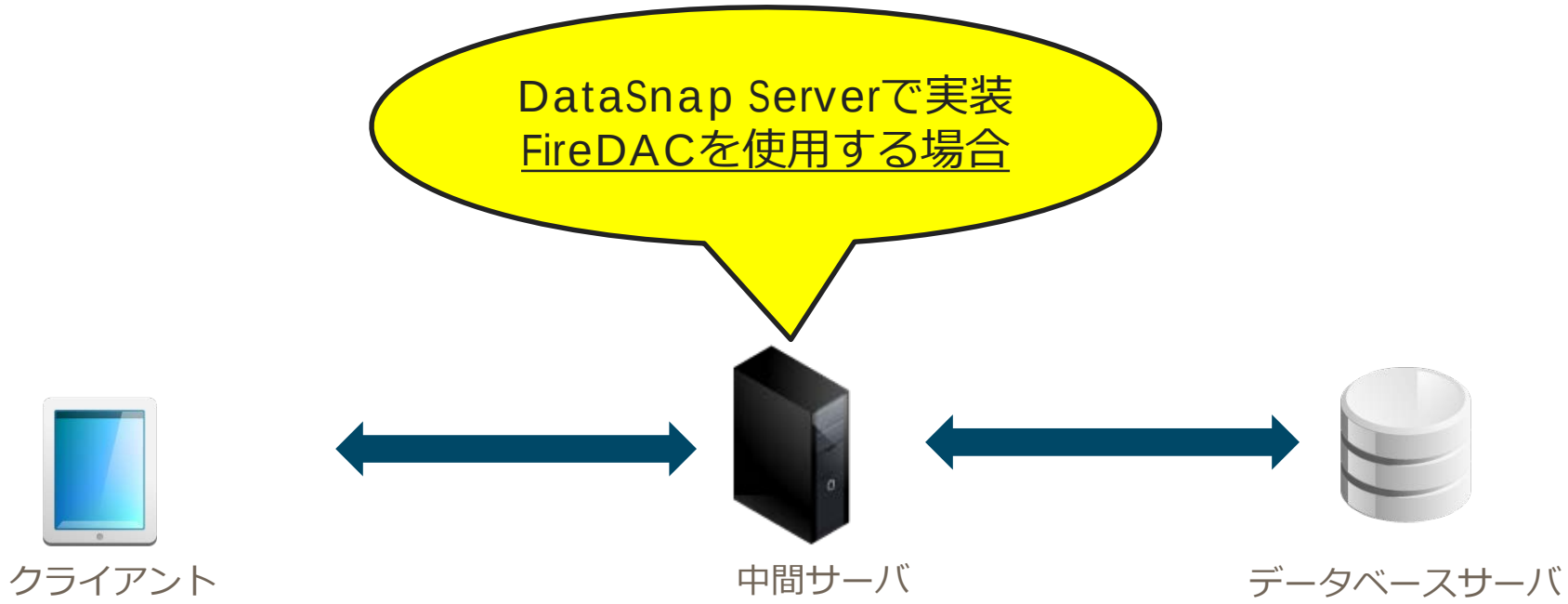
DataSnap Serverを使った実装手順（ウィザード）

- 実装手順4
  自動生成されるユニット





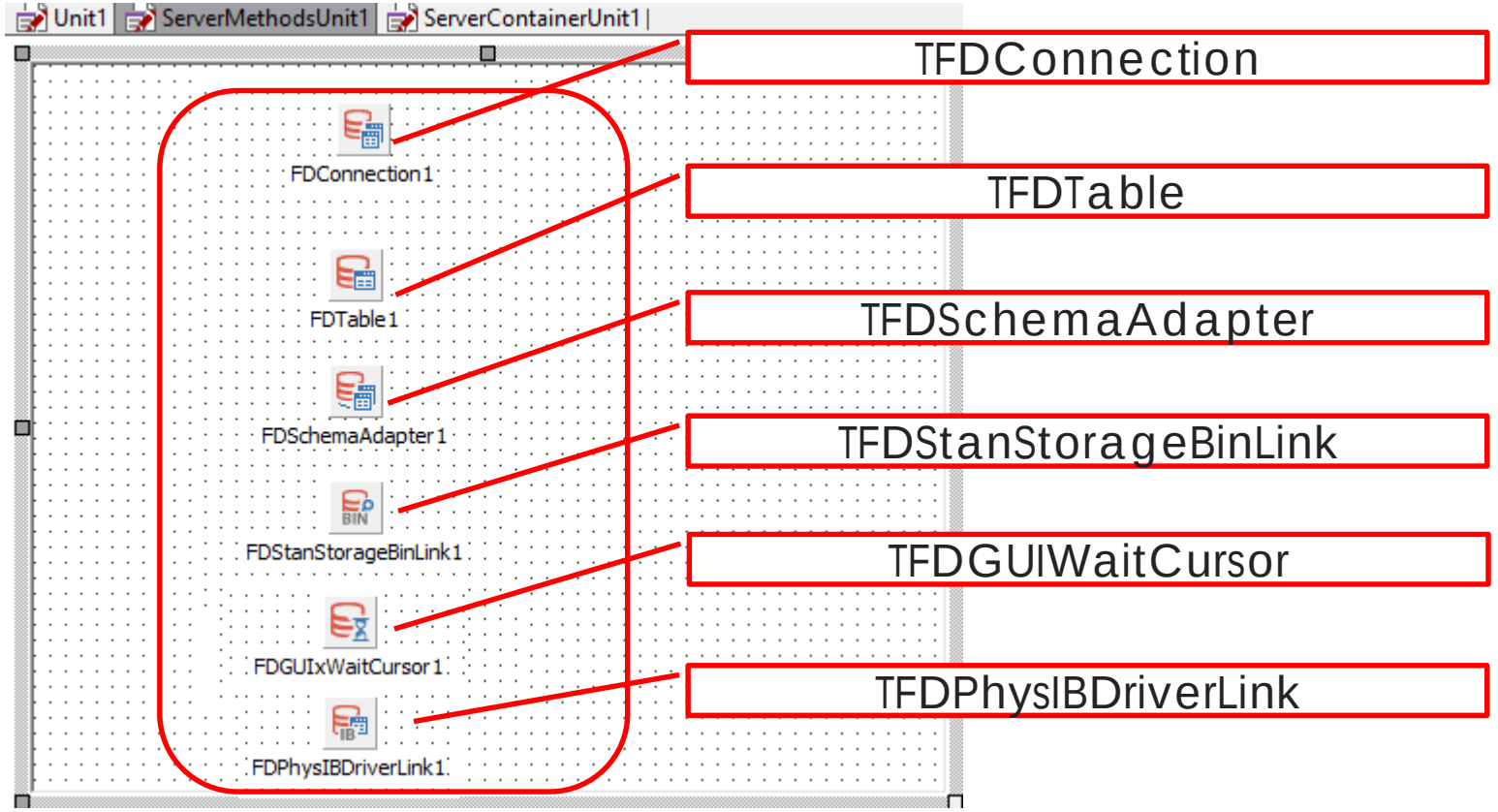
DataSnap Serverを使った実装手順（ FireDAC ）



DataSnap Serverを使った実装手順（ FireDAC ）

- 実装手順5
  サーバ機能のコンポーネント配置

ServerMethodsユニットに機能を実装

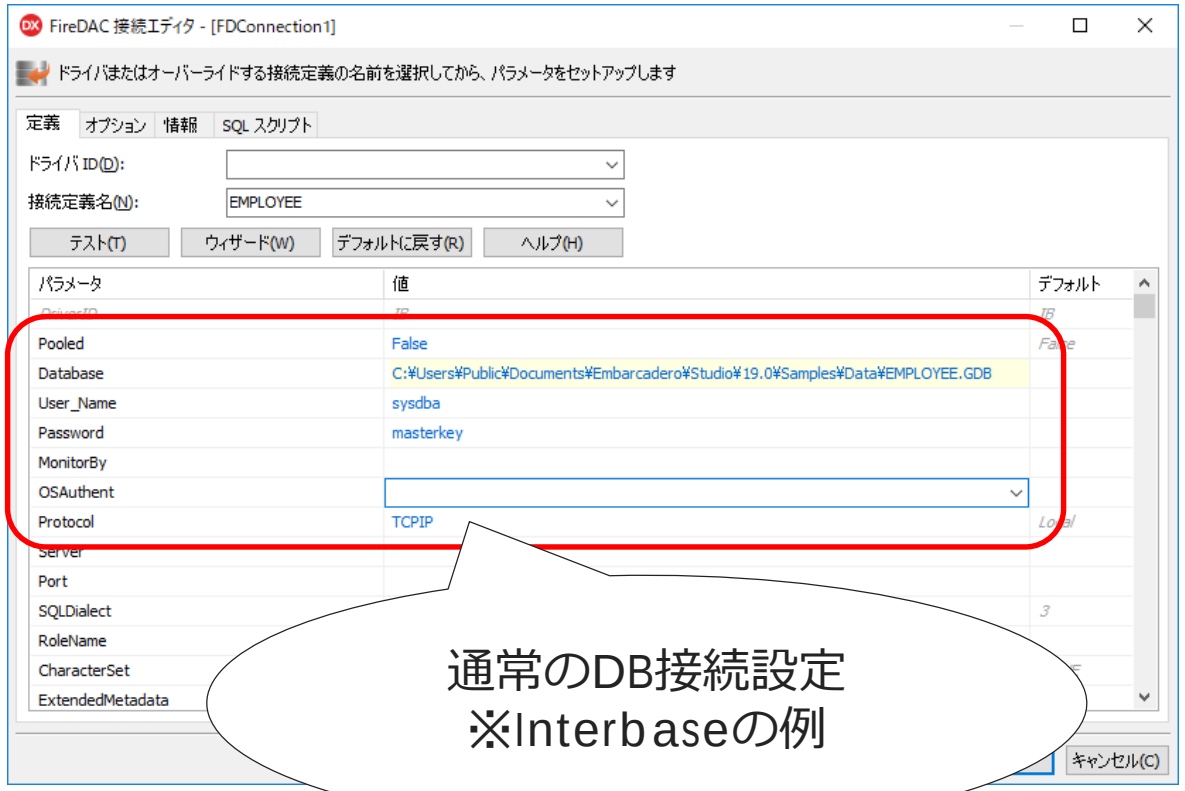


DataSnap Serverを使った実装手順（ FireDAC ）

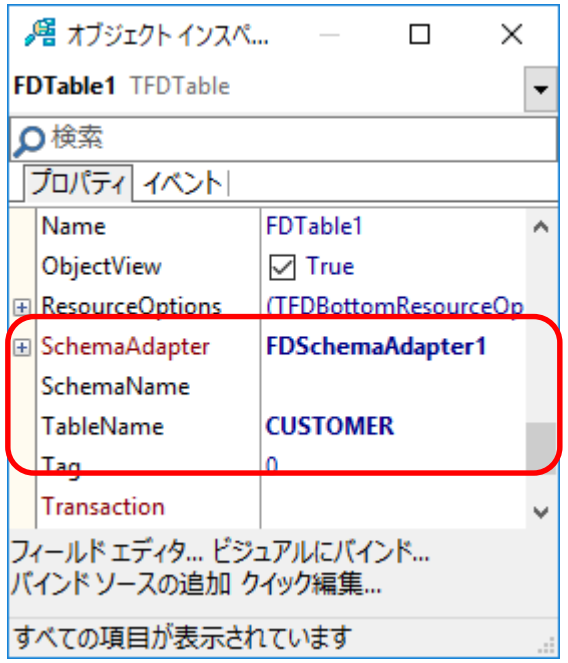
- 実装手順6

コンポーネントの設定

TFDConnection



TFDTable



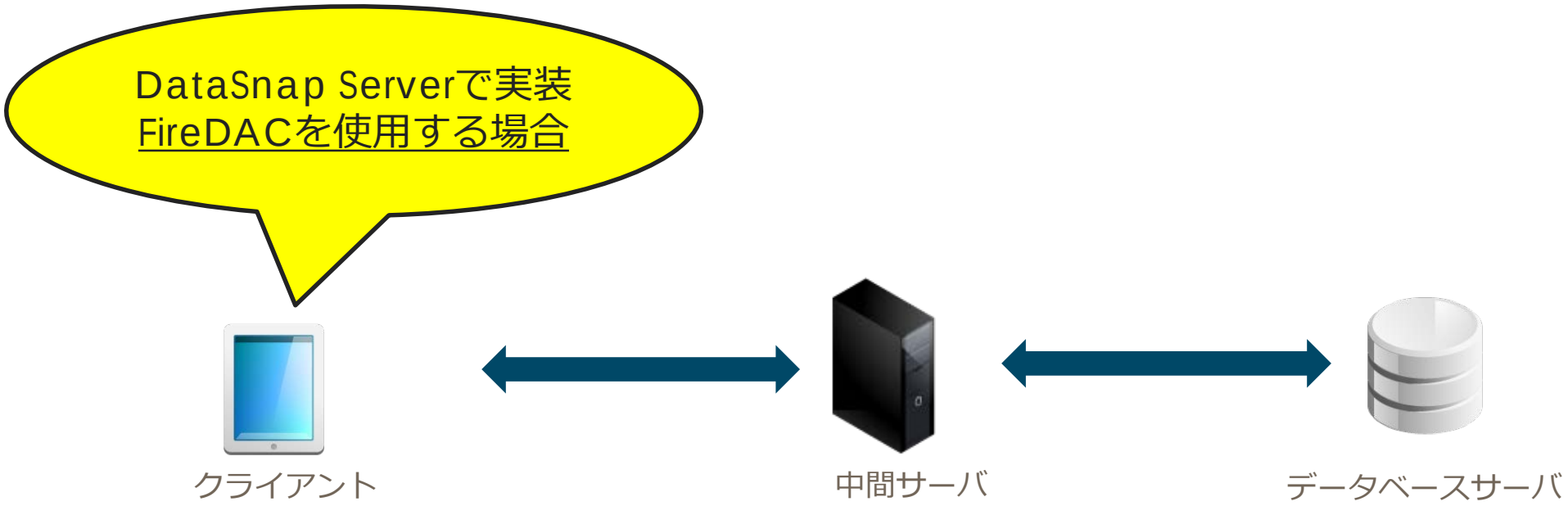
DataSnap Serverを使った実装手順（ FireDAC ）

- 実装手順7
  データ取得の機能を実装

usesにIPPeerClientを追加しておく。

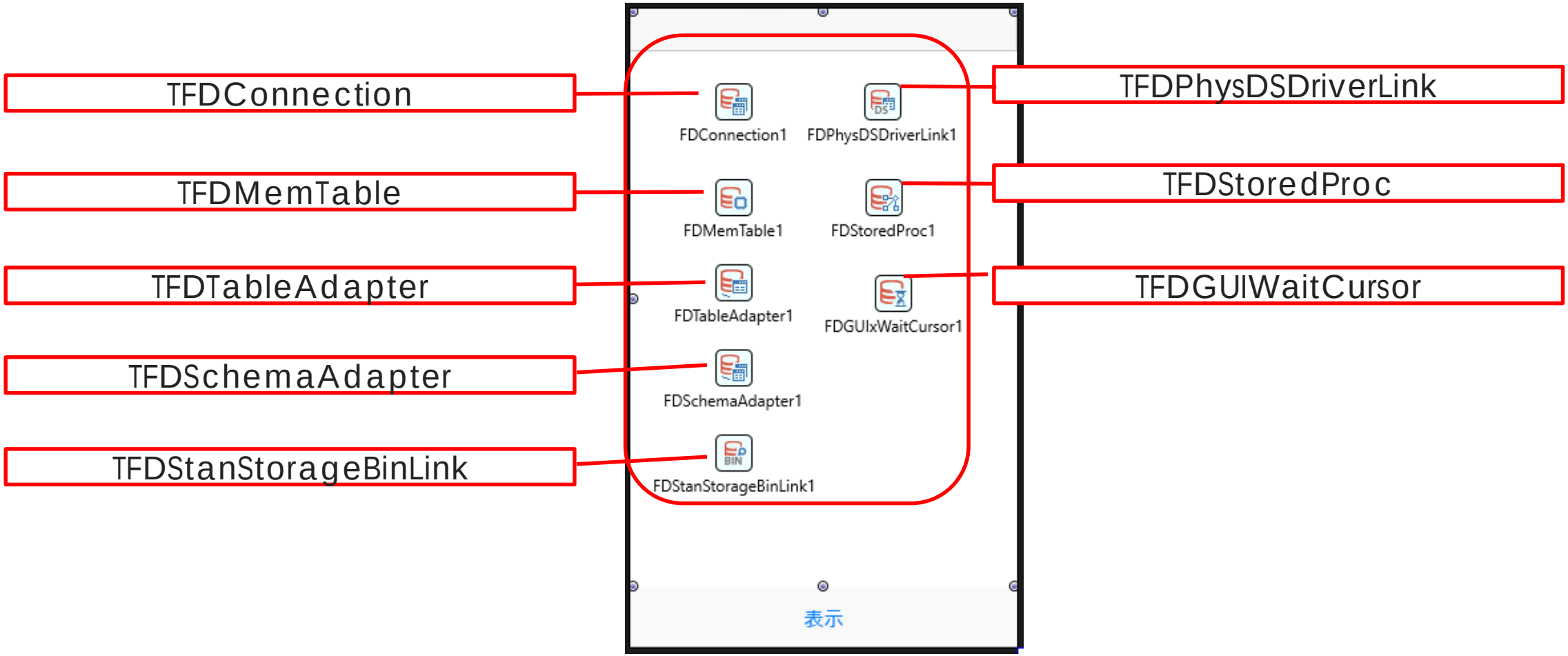
```
function TServerMethods1.GetTable: TStream;
begin
  Result := TMemoryStream.Create;
  try
    FDTable1.Close;
    FDTable1.Open;
    //TFDSchemaAdapterを経由してStream形式で結果をセットする
    FDSchemaAdapter1.SaveToStream(Result, TFDStorageFormat.sfBinary);
    Result.Position := 0;
  except
    raise;
  end;
end;
```

DataSnap Serverを使った実装手順（ FireDAC ）



DataSnap Serverを使った実装手順（ FireDAC ）

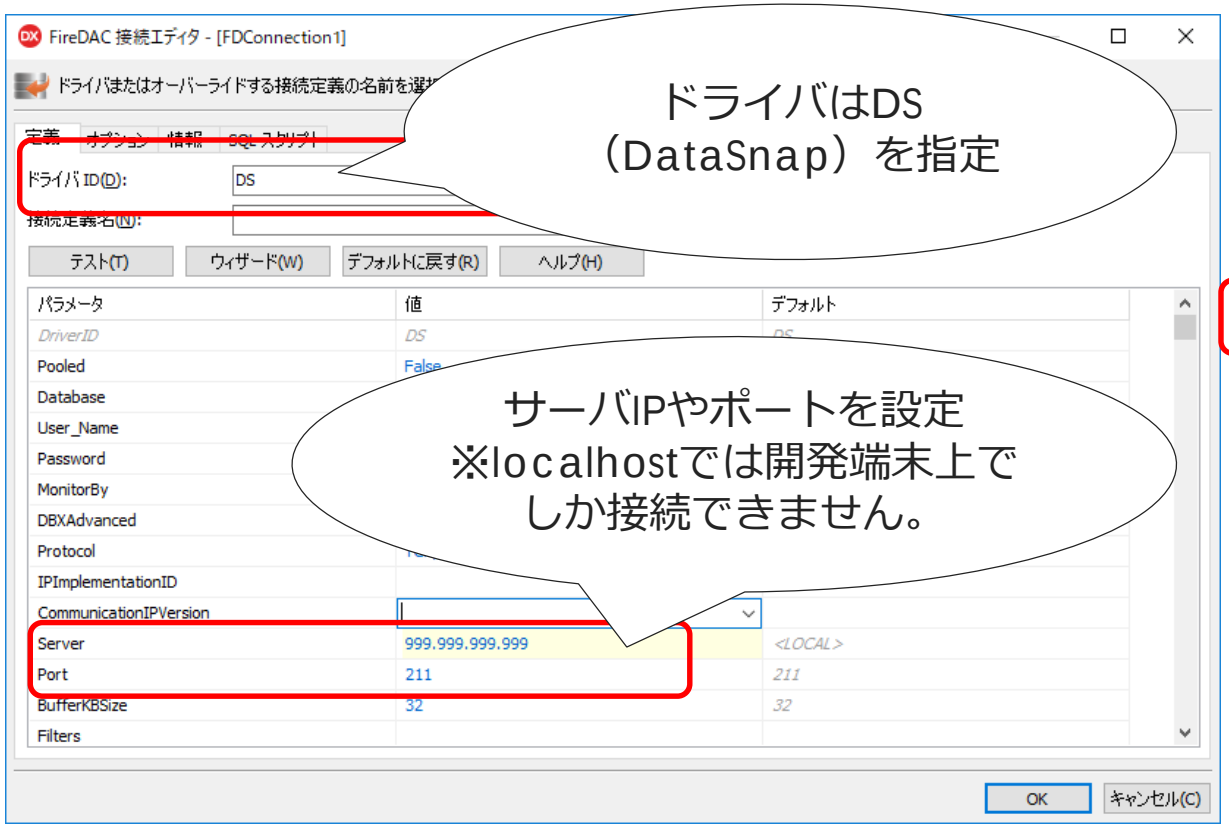
- 実装手順8
  クライアントアプリのコンポーネント配置



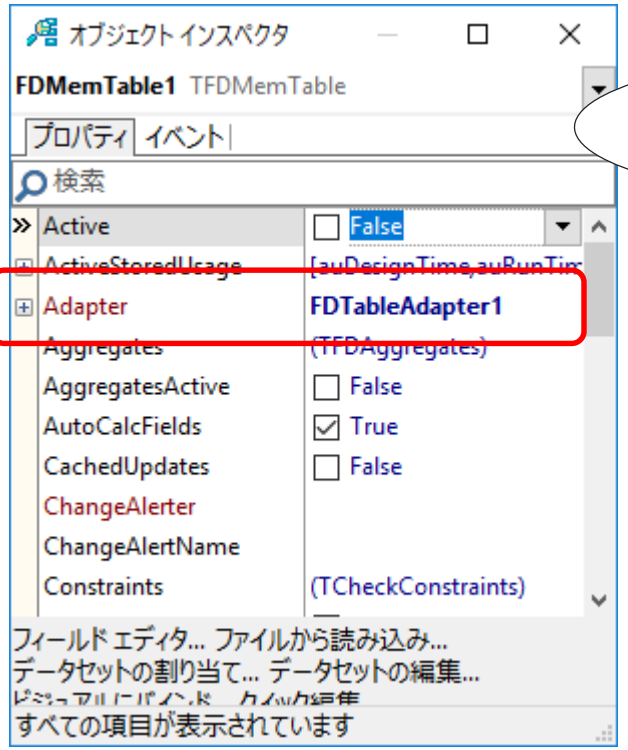
DataSnap Serverを使った実装手順（ FireDAC ）

- 実装手順9
  コンポーネントの設定

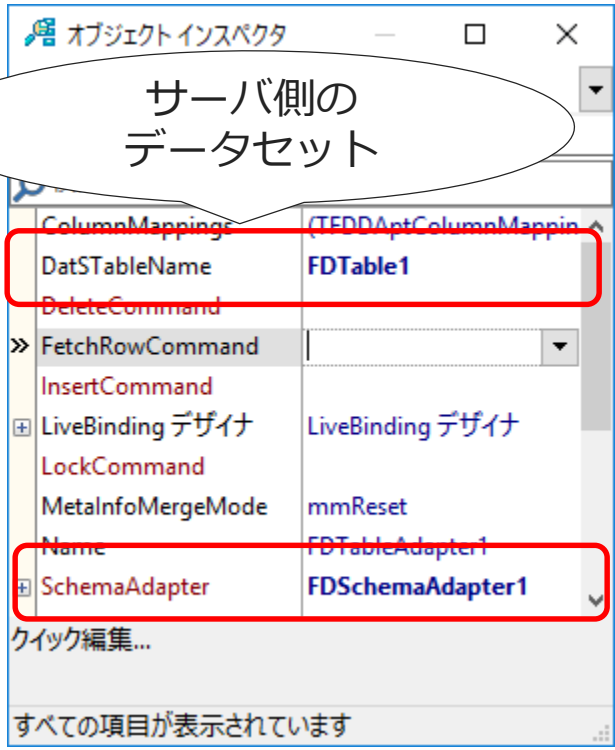
TFDConnection



TFDMemTable



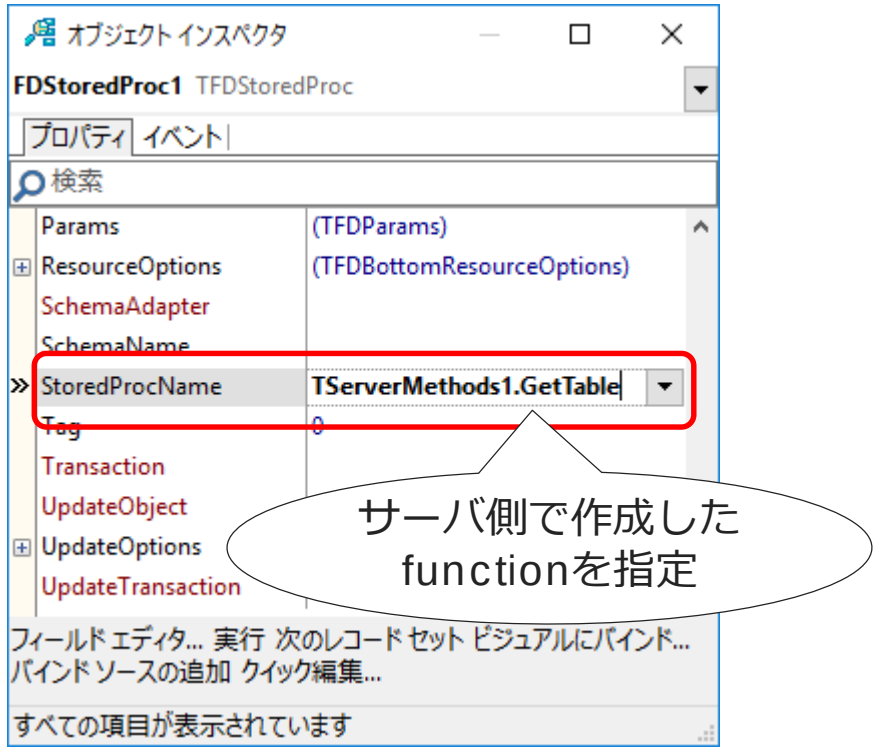
TFDTableAdapter



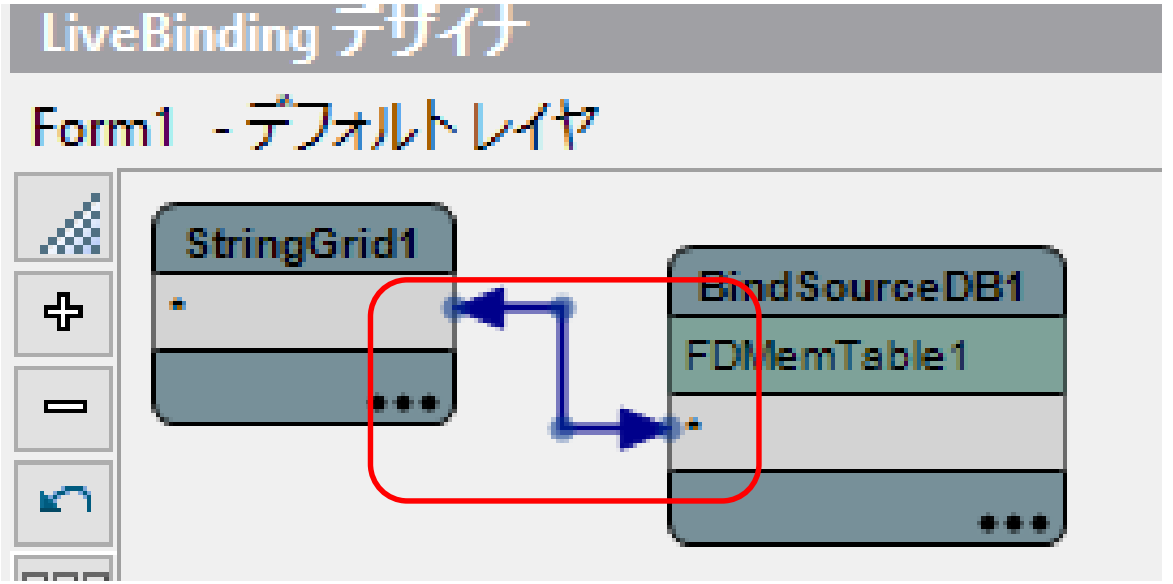
DataSnap Serverを使った実装手順（ FireDAC ）

- 実装手順10
  コンポーネントの設定

TFDStoredProc



ライブバインディング設定



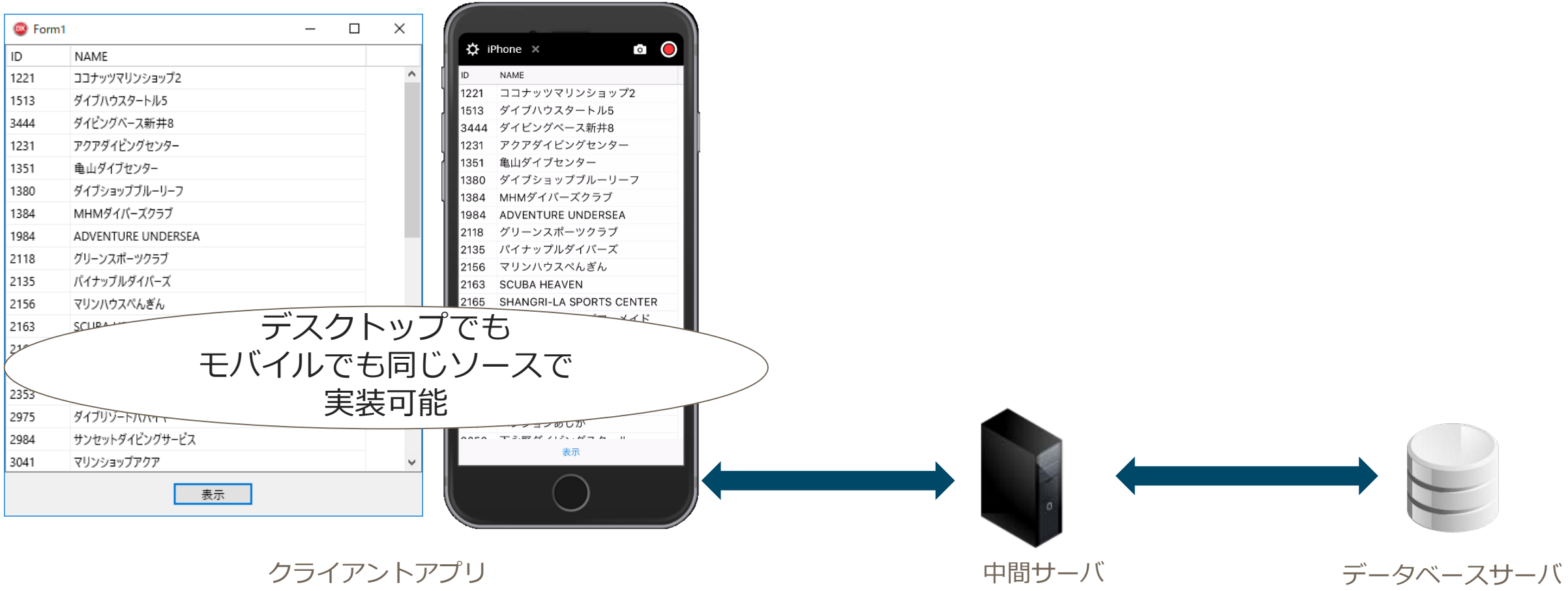
DataSnap Serverを使った実装手順（ FireDAC ）

- 実装手順11
  データ取得の機能を実装

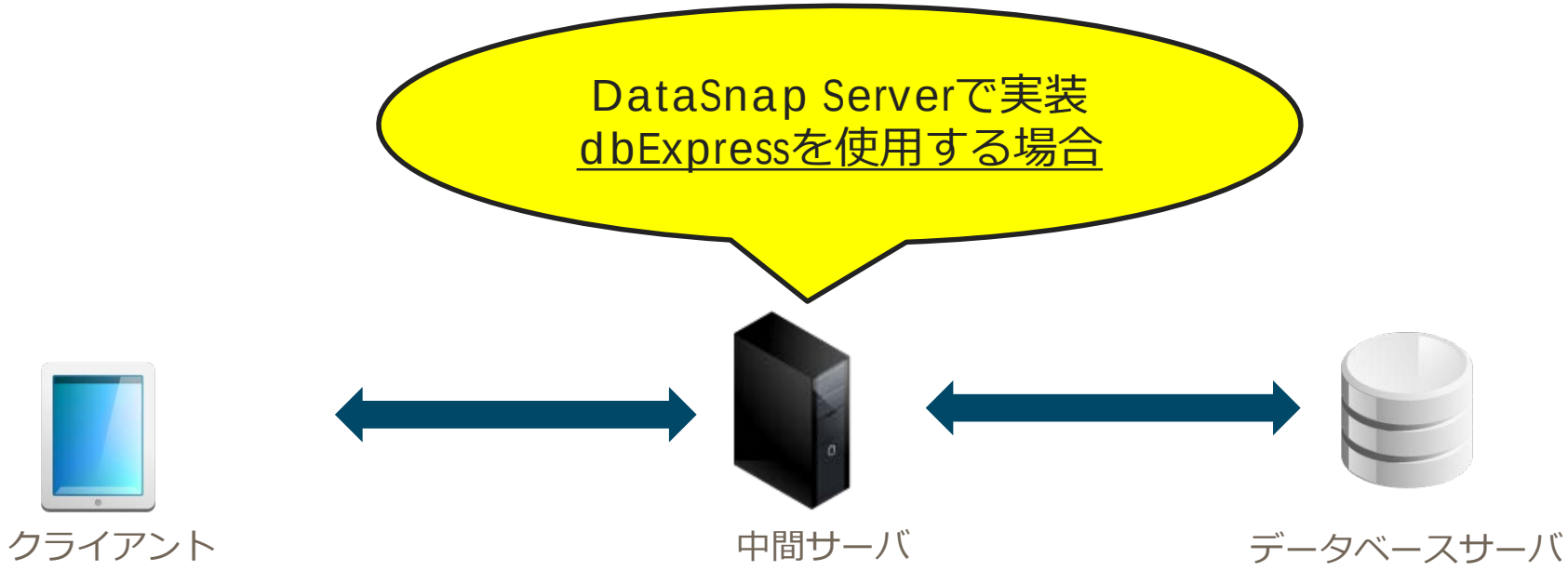
```
procedure TForm1.Button1Click(Sender: TObject);
var
  LStringStream: TStringStream;
begin
  //TFDStoredProcでサーバで作成したfunctionをCallしてストリームを受け取る
  FDStoredProc1.ExecProc;
  LStringStream := TStringStream.Create(FDStoredProc1.Params[0].asBlob);
  try
    if LStringStream <> nil then
    begin
      LStringStream.Position := 0;
      FDSchemaAdapter1.LoadFromStream(LStringStream, TFDStorageFormat.sfBinary);
    end;
  finally
    LStringStream.Free;
  end;

end;
```

DataSnap Serverを使った実装手順（ FireDAC ）

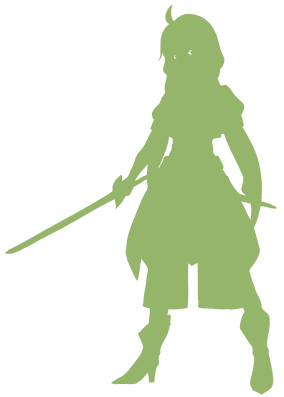
- 実行



DataSnap ServerでdbExpressを使う実装方法は、P.77以降の補足資料を参考ください。

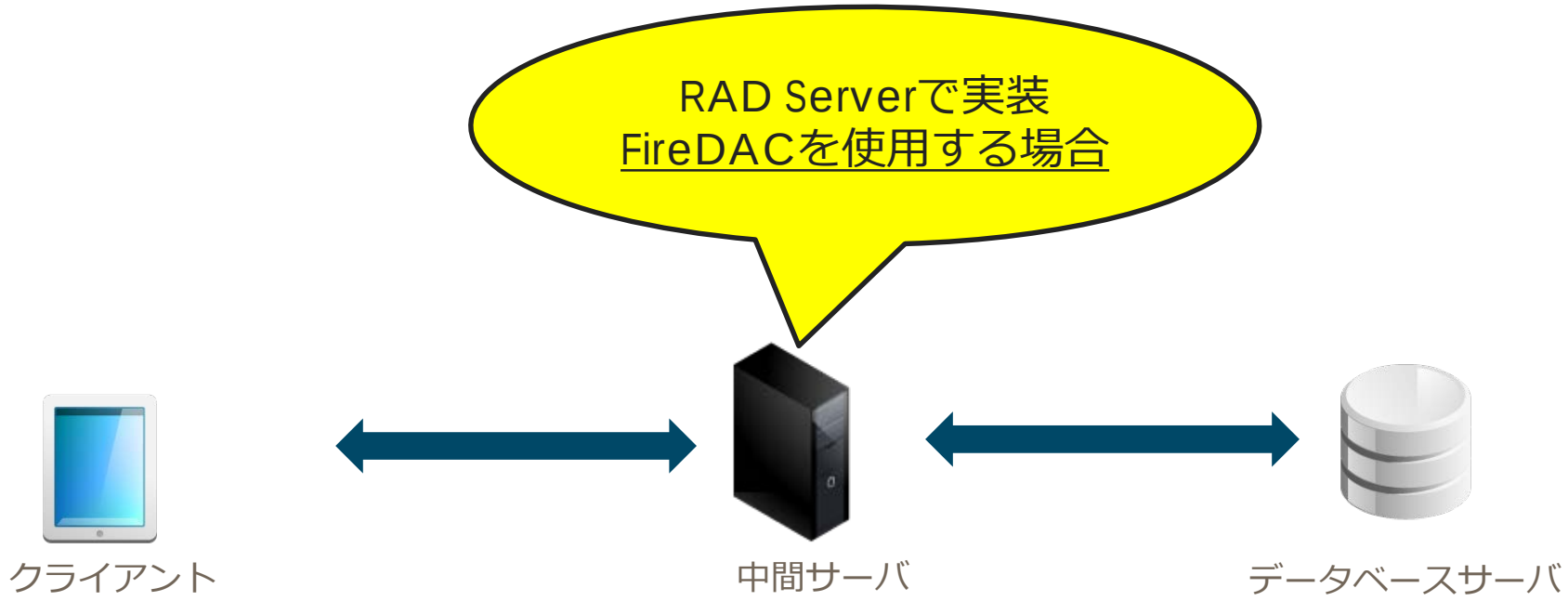


RAD Serverを使った実装手順



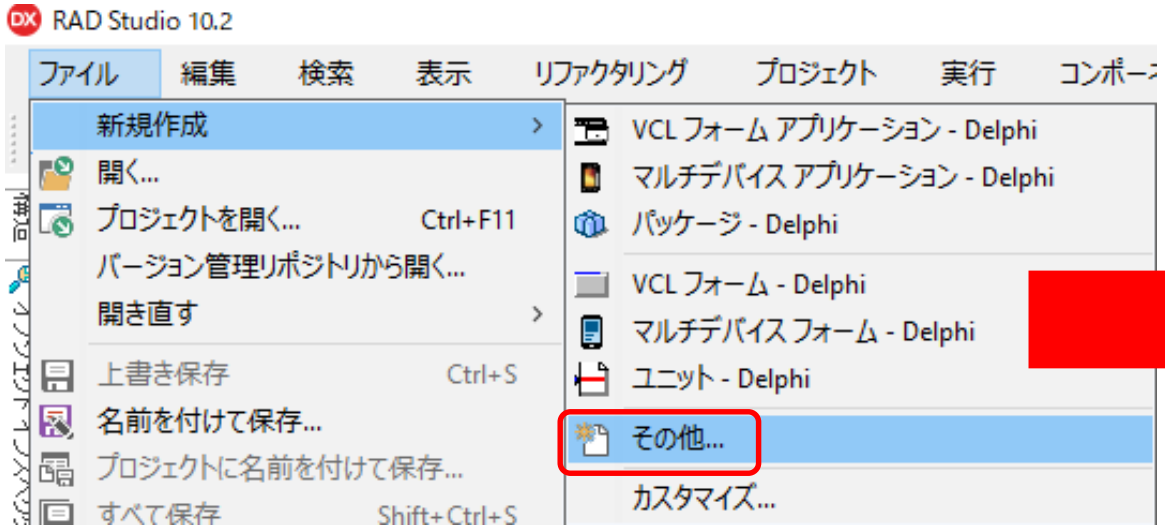
embarcadero®
DEVELOPER CAMP

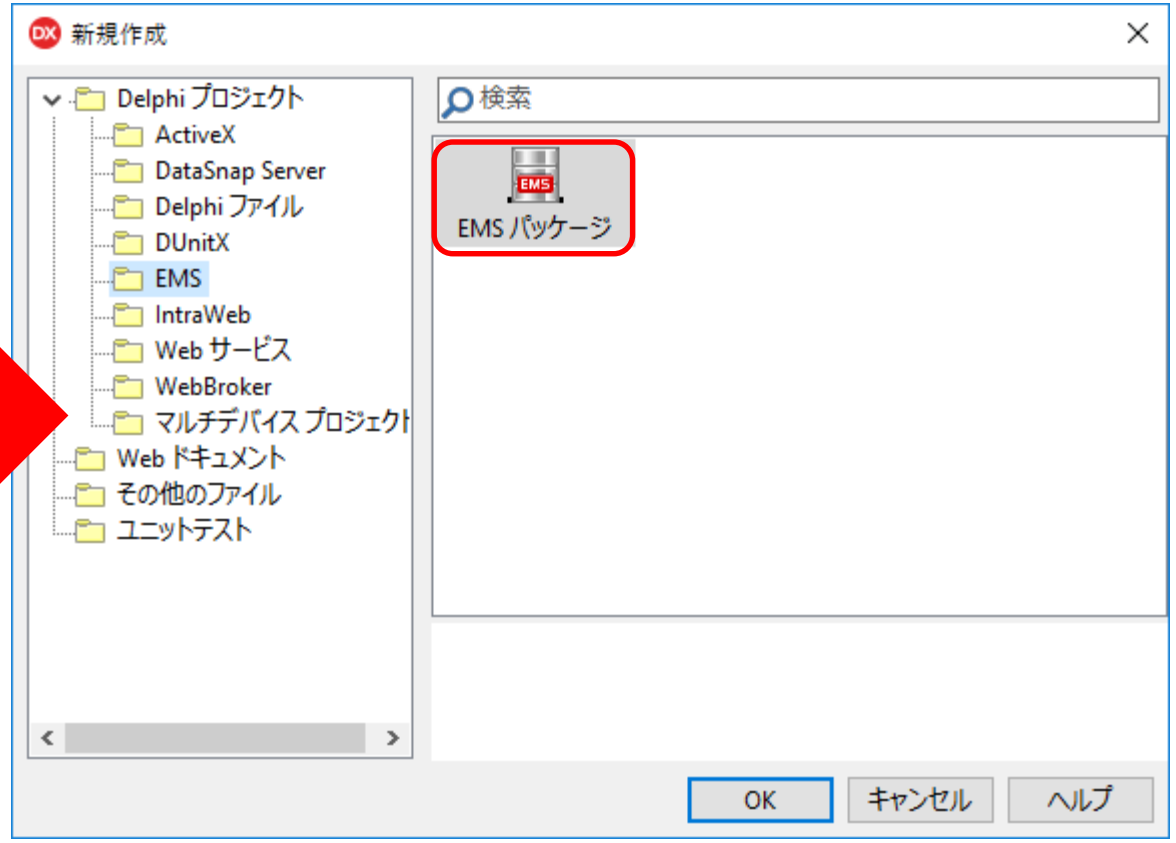
RAD Serverを使った実装手順（ FireDAC ）



RAD Serverを使った実装手順（ FireDAC ）

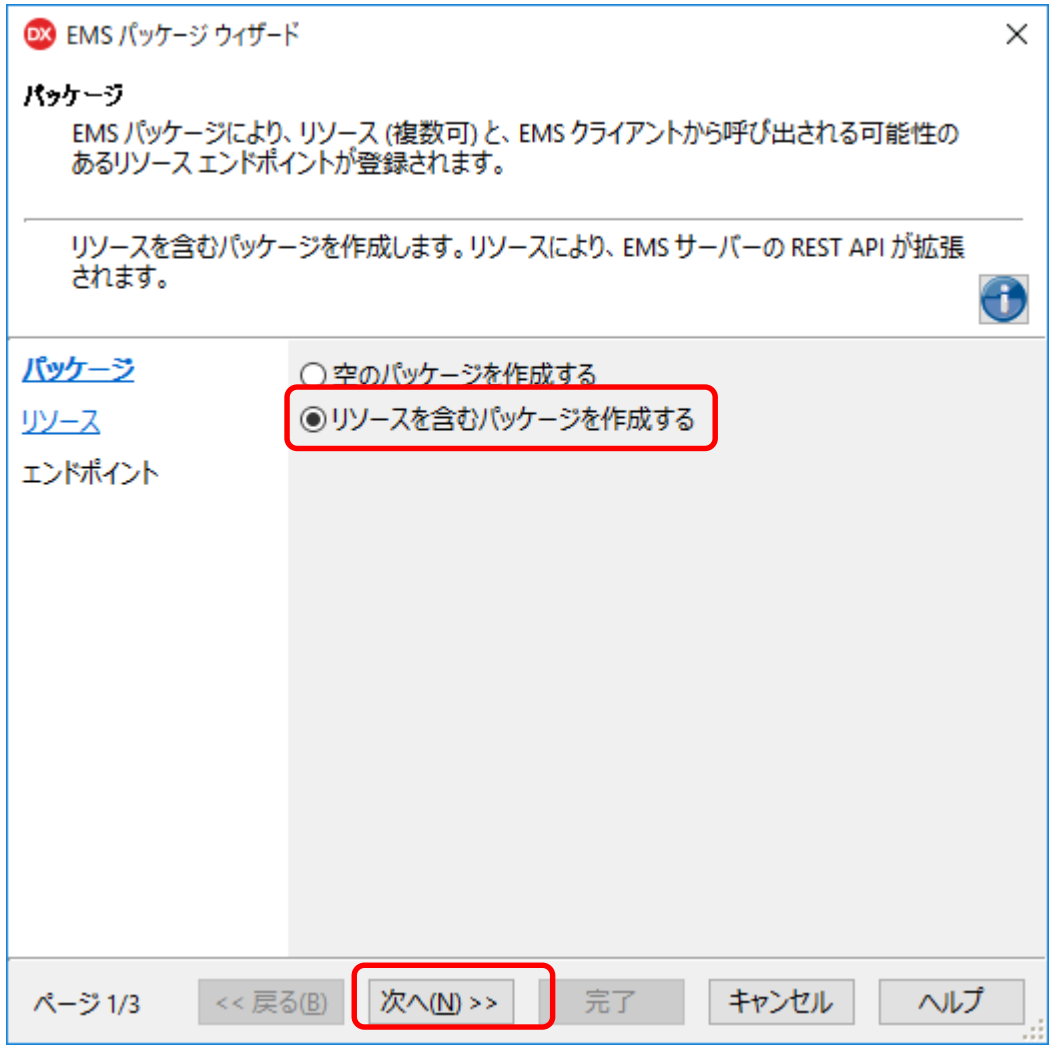
- 実装手順1
  プロジェクトの作成





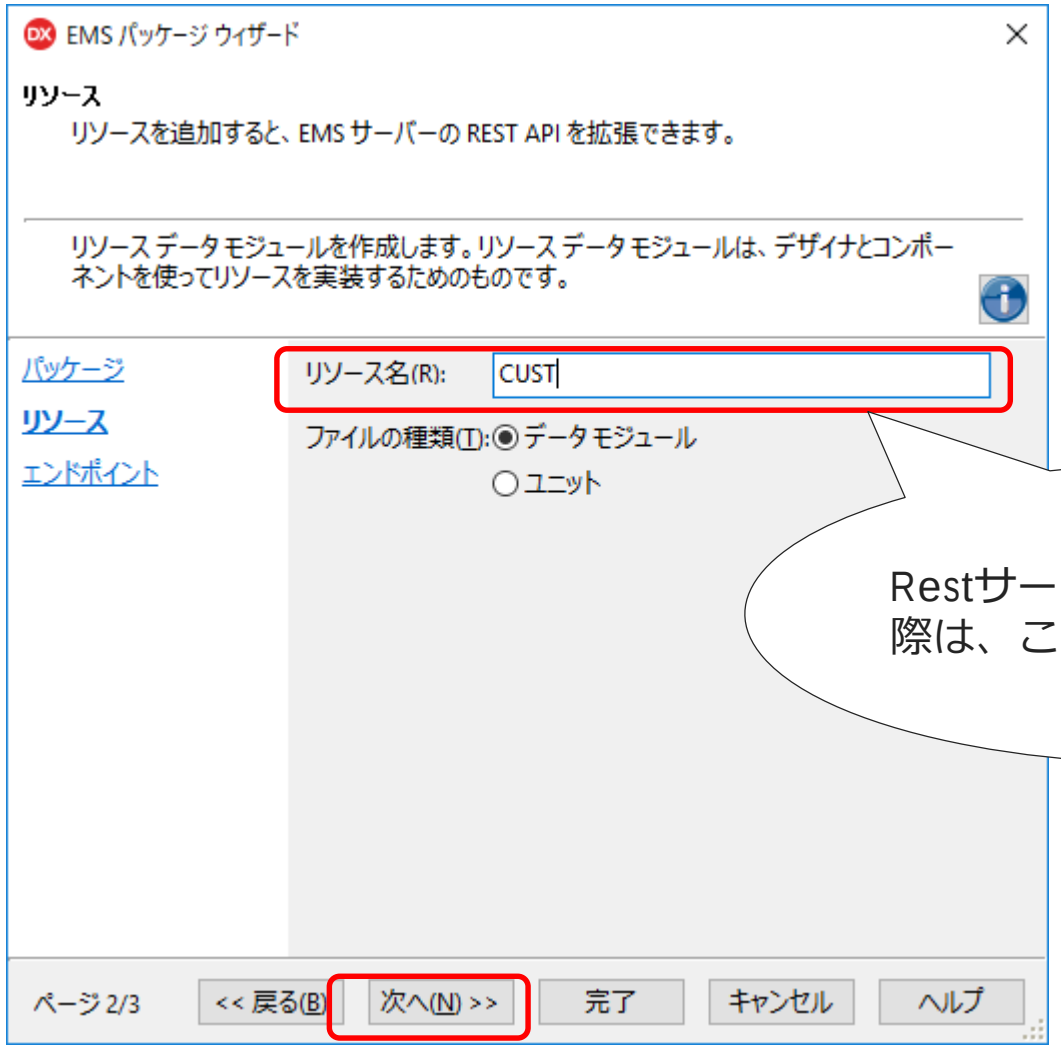
RAD Serverを使った実装手順（ FireDAC ）

- 実装手順2
  ウィザードの指定



RAD Serverを使った実装手順（ FireDAC ）

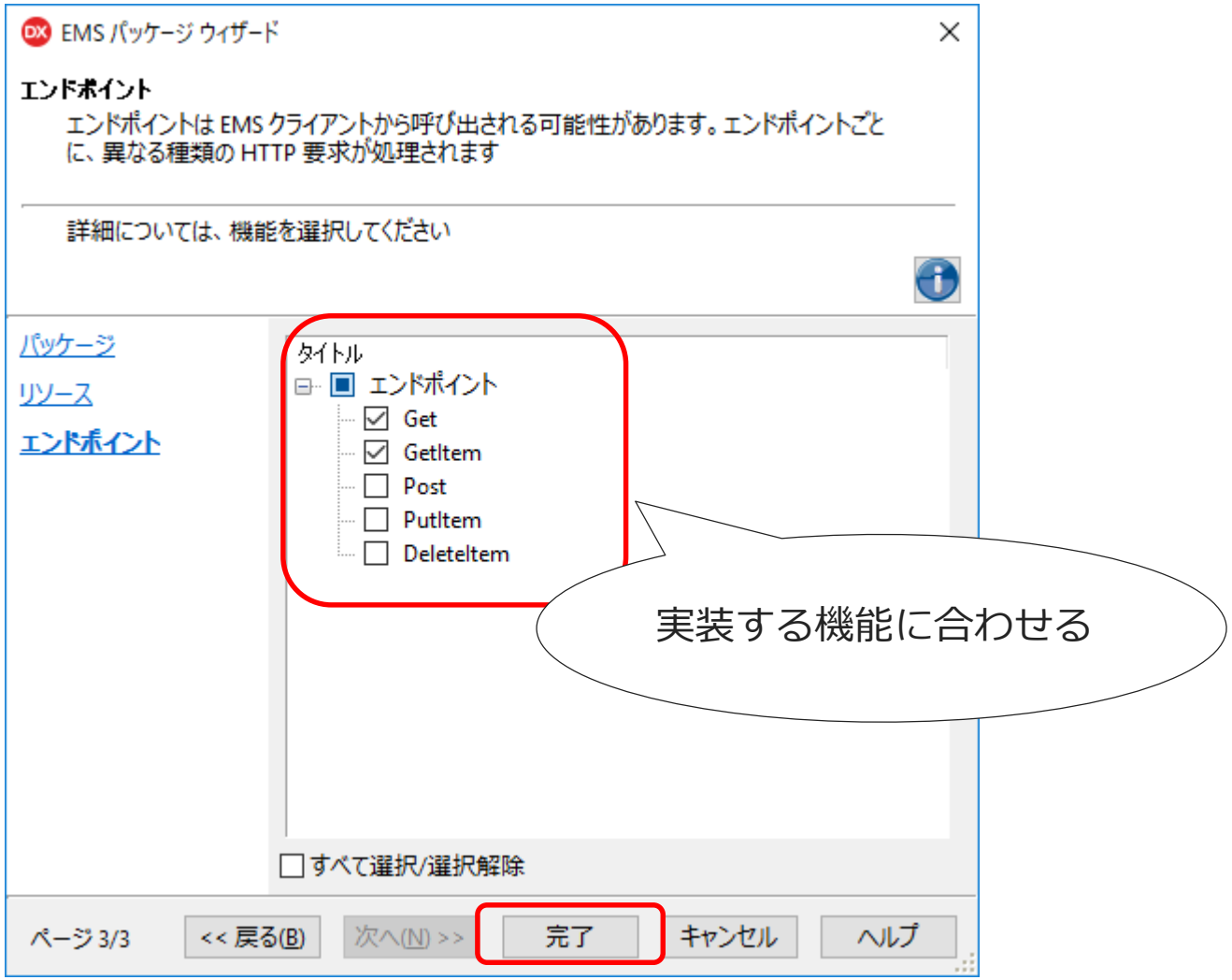
- 実装手順3
  ウィザードの指定



Restサービスとして呼び出す際は、このリソース名を使用

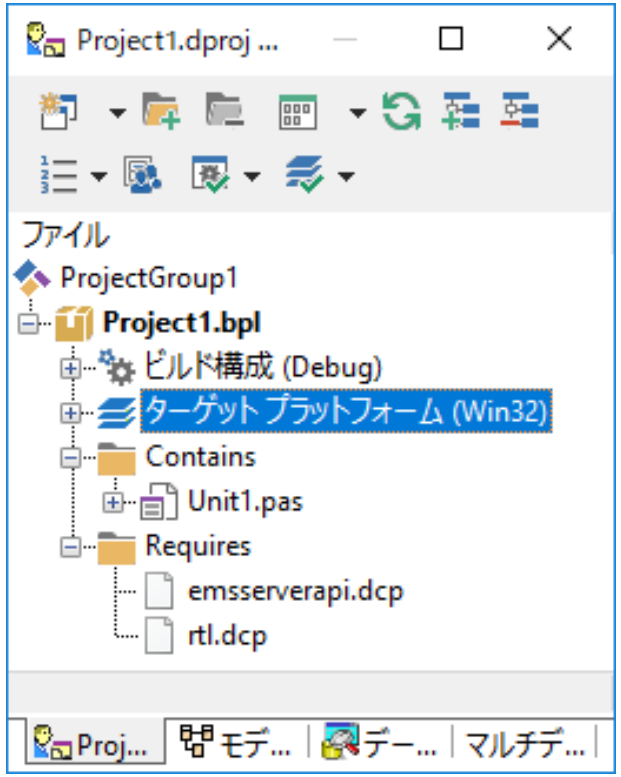
RAD Serverを使った実装手順（ FireDAC ）

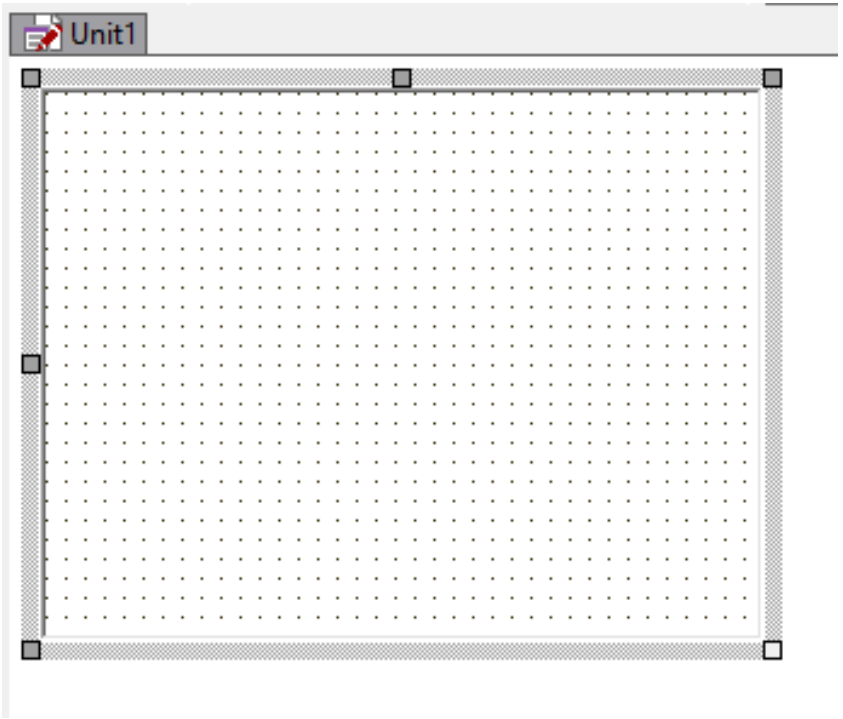
- 実装手順4
  ウィザードの指定



RAD Serverを使った実装手順（ FireDAC ）

- 実装手順5
  自動生成されるユニット

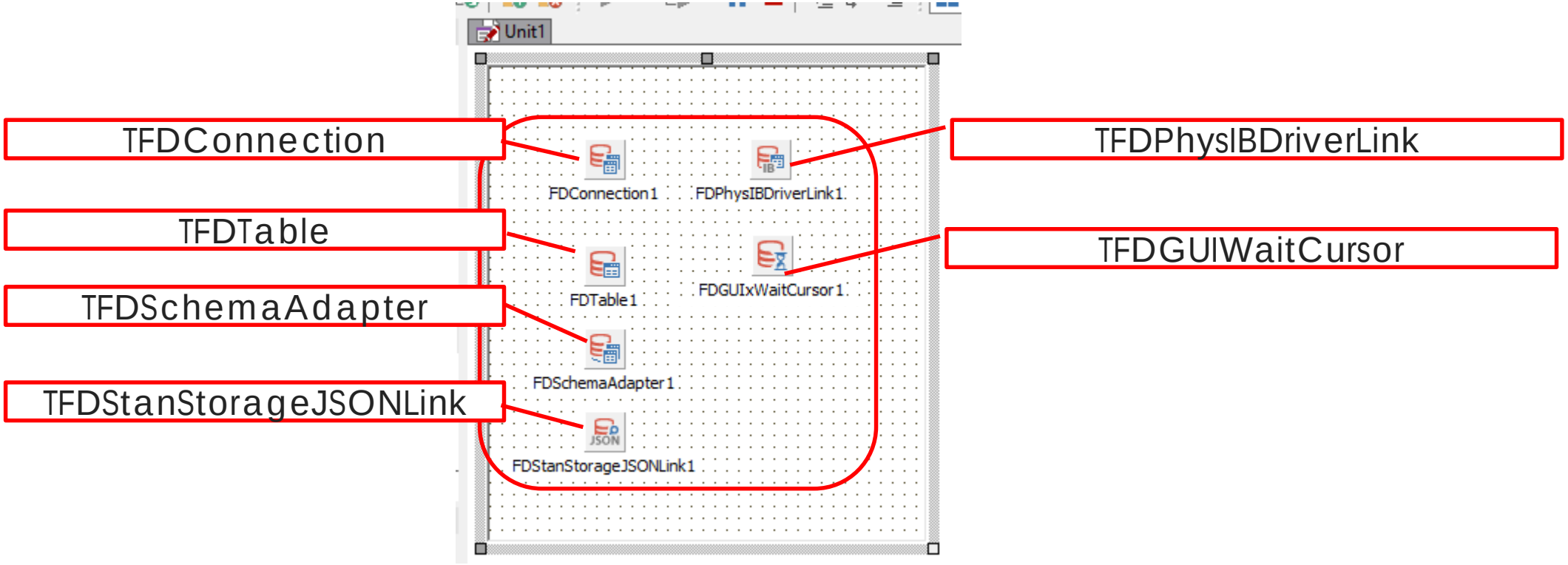




RAD Serverを使った実装手順（ FireDAC ）

- 実装手順6
  サーバ機能のコンポーネント配置

  リソースとして機能を実装

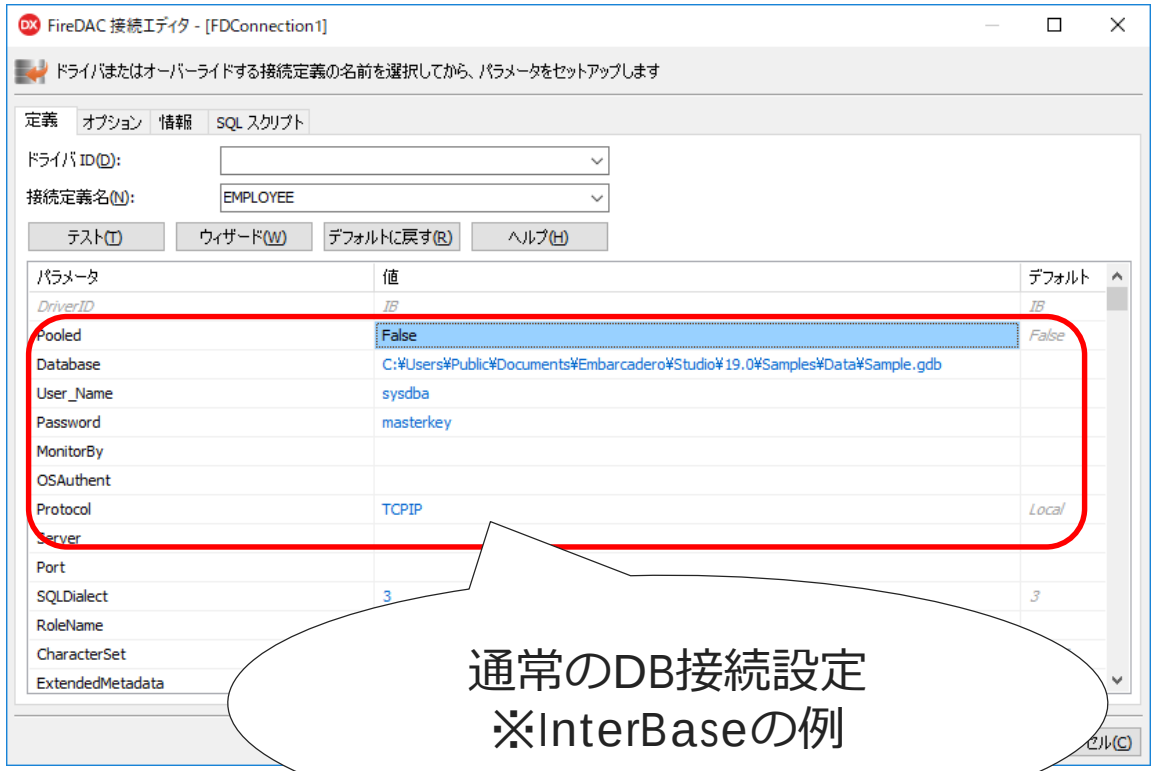


RAD Serverを使った実装手順（ FireDAC ）

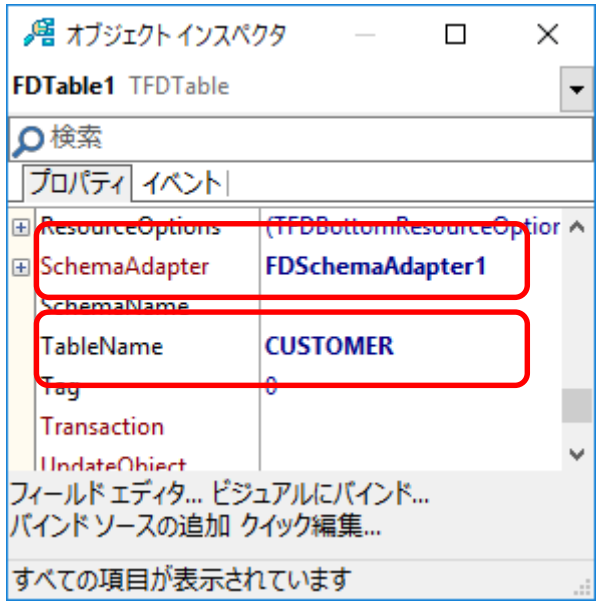
- 実装手順7

コンポーネントの設定

TFDConnection



TFDTable



RAD Serverを使った実装手順（ FireDAC ）

- 実装手順8
  データ取得の機能を実装

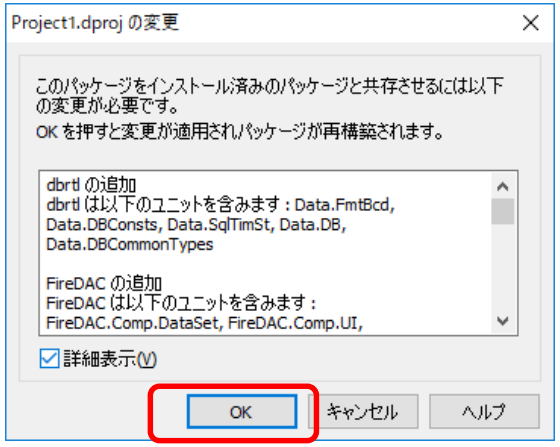
```
procedure TCUSTResource1.Get(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
var
  oStr: TMemoryStream;
begin
  oStr := TMemoryStream.Create;

  // クエリの実行結果をスキーマアダプタからメモリストリーム経由で返す
  FDTable1.Open;
  FDSchemaAdapter1.SaveToStream(oStr,TFDStorageFormat.sfJSON);
  AResponse.Body.SetStream(oStr,'application/json', True);
end;
```

Restでパラメータを受け取ることも可能
ARequest.Params.Values['XXXX']
※QueryのSQLでバインド変数として
ParamBy値で利用すると便利

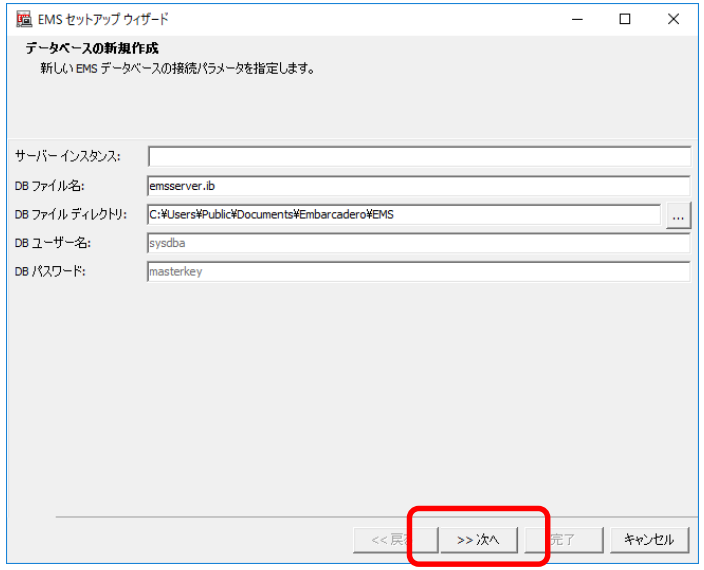
RAD Serverを使った実装手順（ FireDAC ）

- 実装手順9
  コンパイル・実行
  初回コンパイルや実行時はinterBaseの設定も含め、ダイアログがいくつか出るので応答が必要。

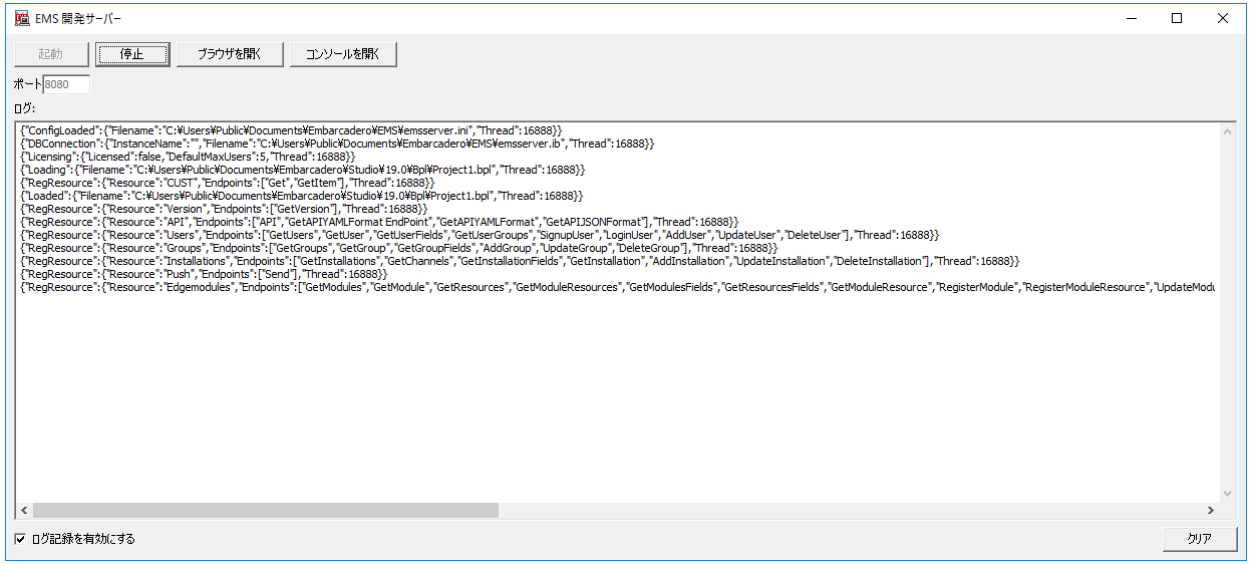
パッケージの参照



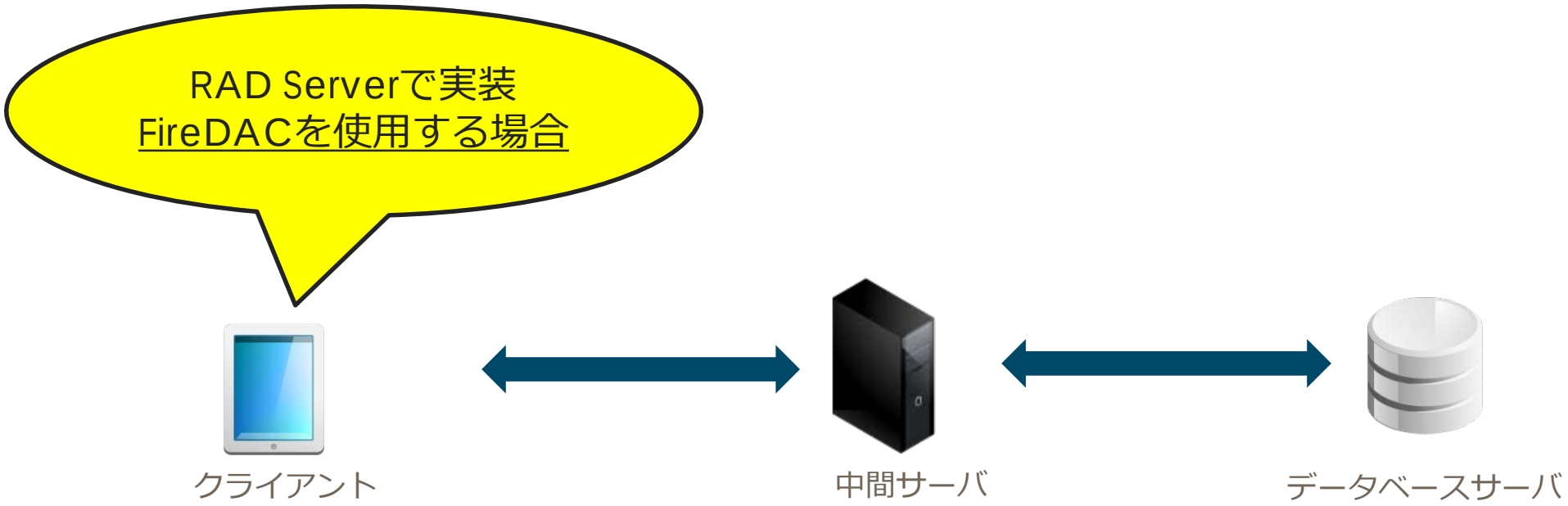
InterBaseの設定
（サーバ管理データ）



起動画面

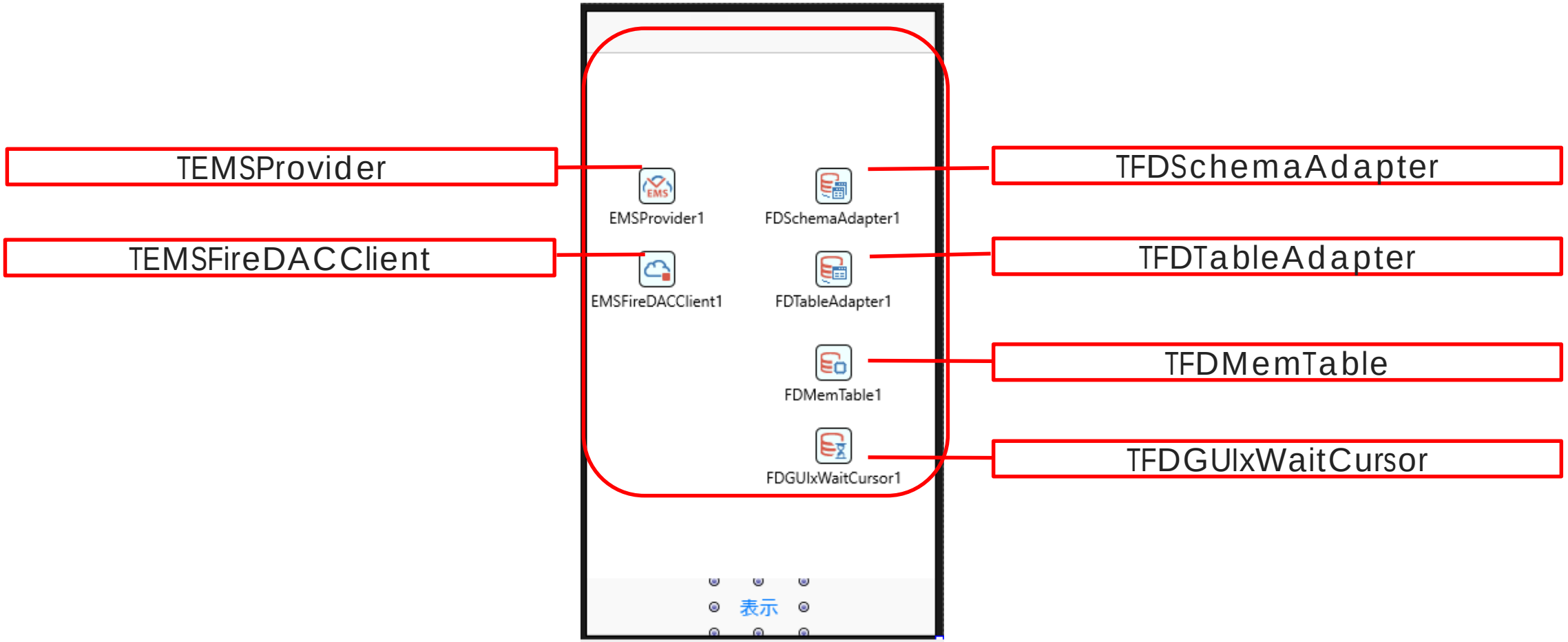


RAD Serverを使った実装手順（ FireDAC ）



RAD Serverを使った実装手順（ FireDAC ）

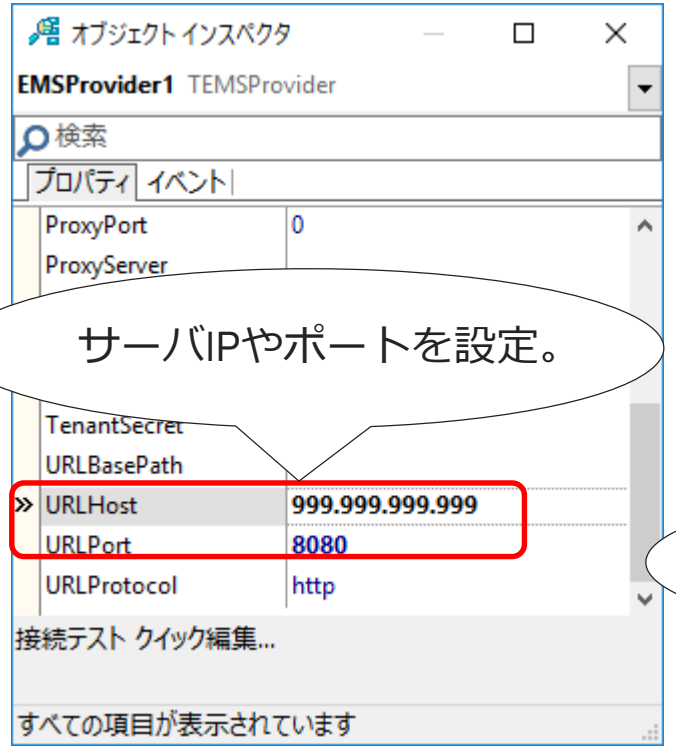
- 実装手順10
  クライアントアプリケーションのコンポーネント配置



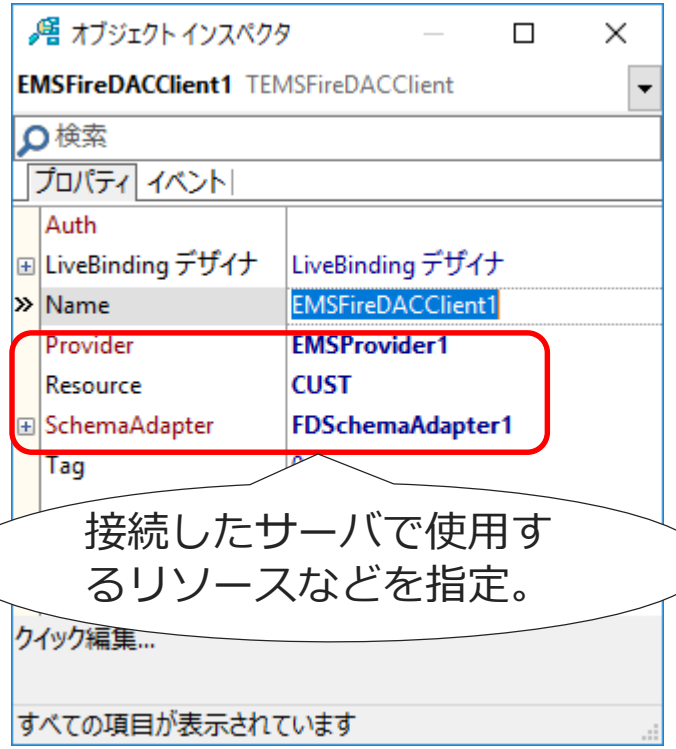
RAD Serverを使った実装手順（ FireDAC ）

- 実装手順11
  コンポーネントの設定

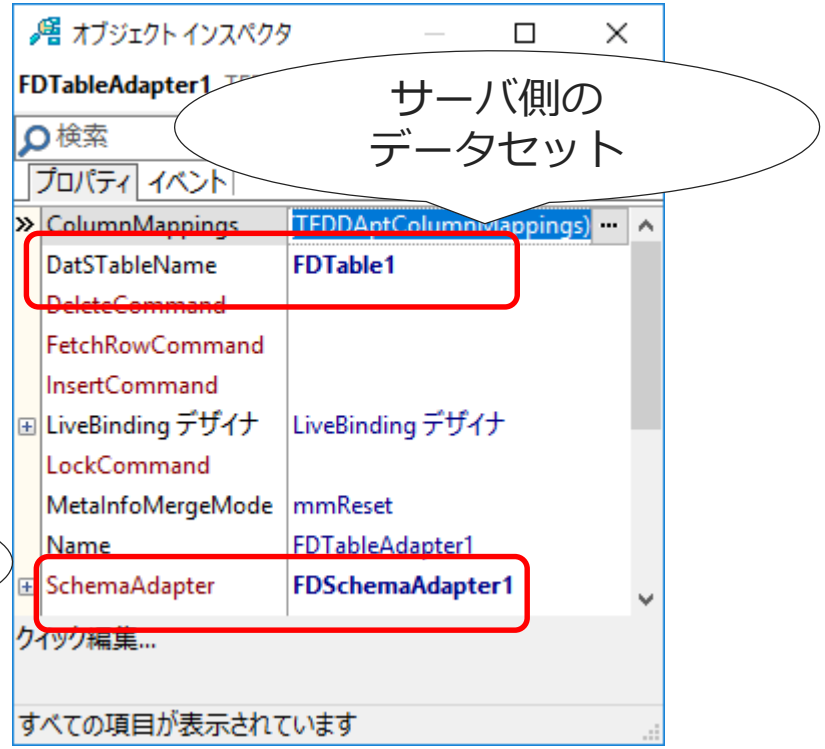
TEMSProvider



TEMSFireDACClient



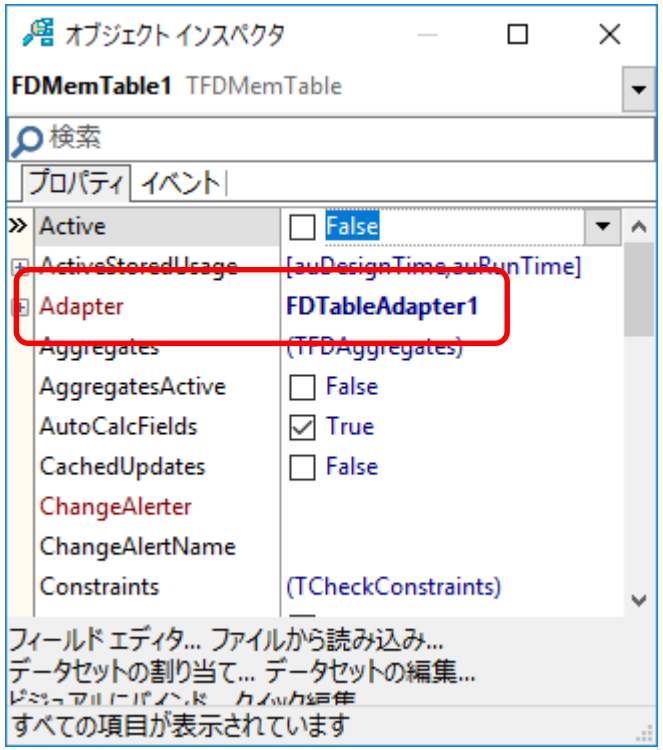
TFDTableAdapter



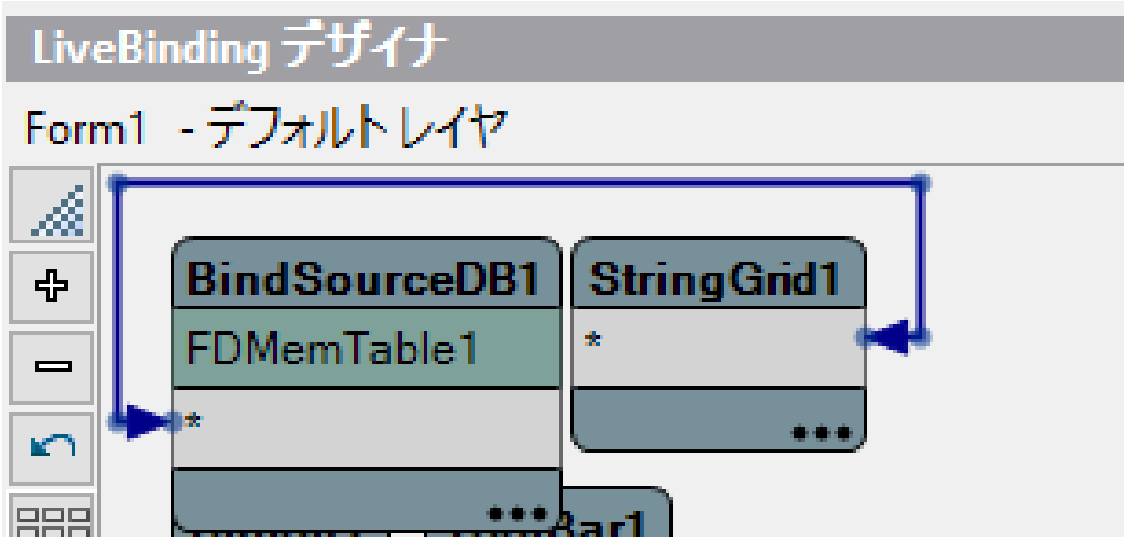
RAD Serverを使った実装手順（ FireDAC ）

- 実装手順12
  コンポーネントの設定

TFDMemTable



ライブバインディング設定



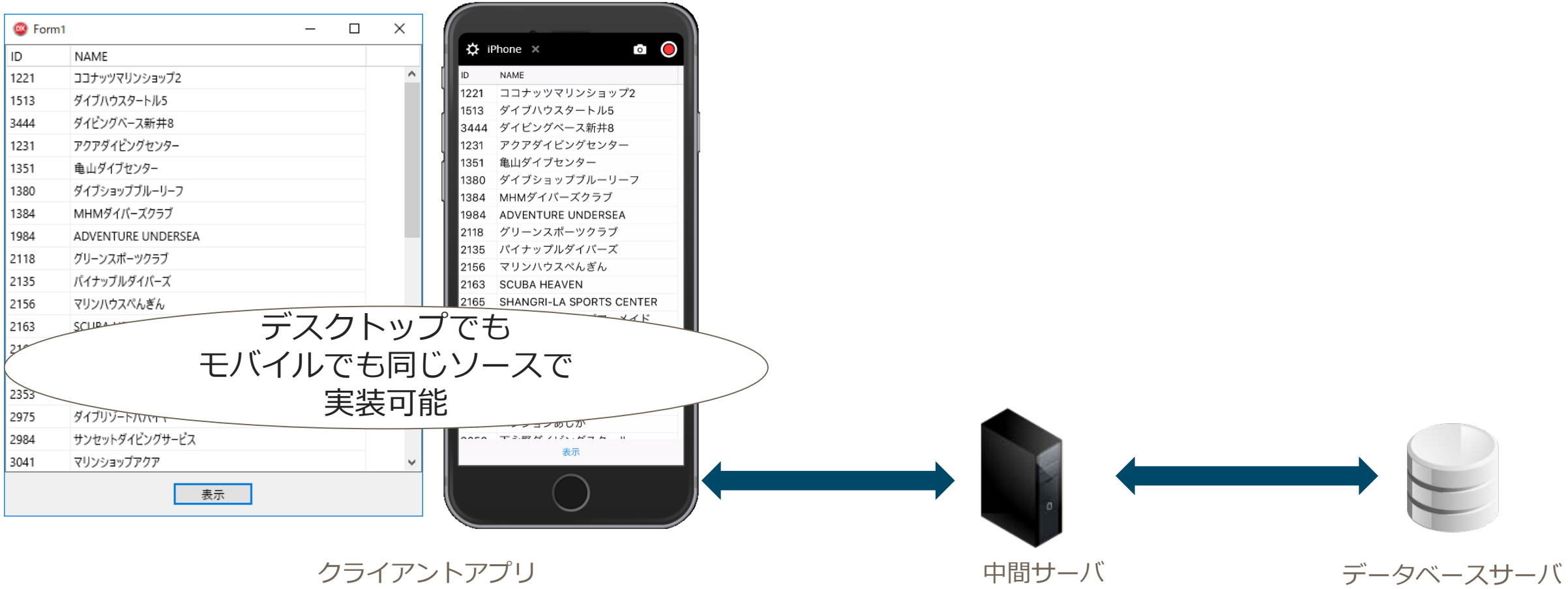
RAD Serverを使った実装手順（ FireDAC ）

- 実装手順13
  データ取得の機能を実装

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //設定しているリソースのGetDataを呼び出す
  EMSFireDACClient1.GetData;
end;
```

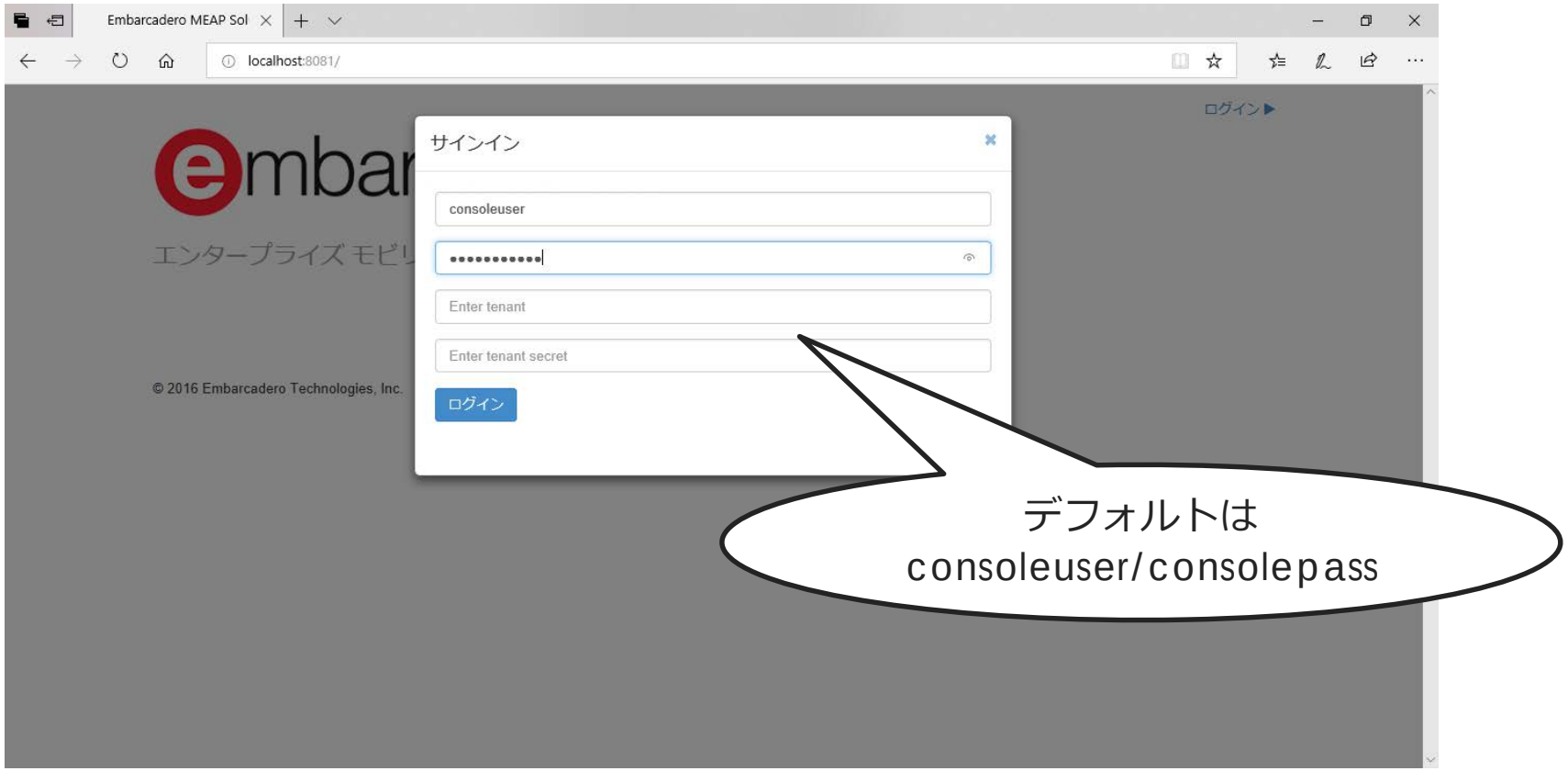
RAD Serverを使った実装手順（ FireDAC ）

- 実行



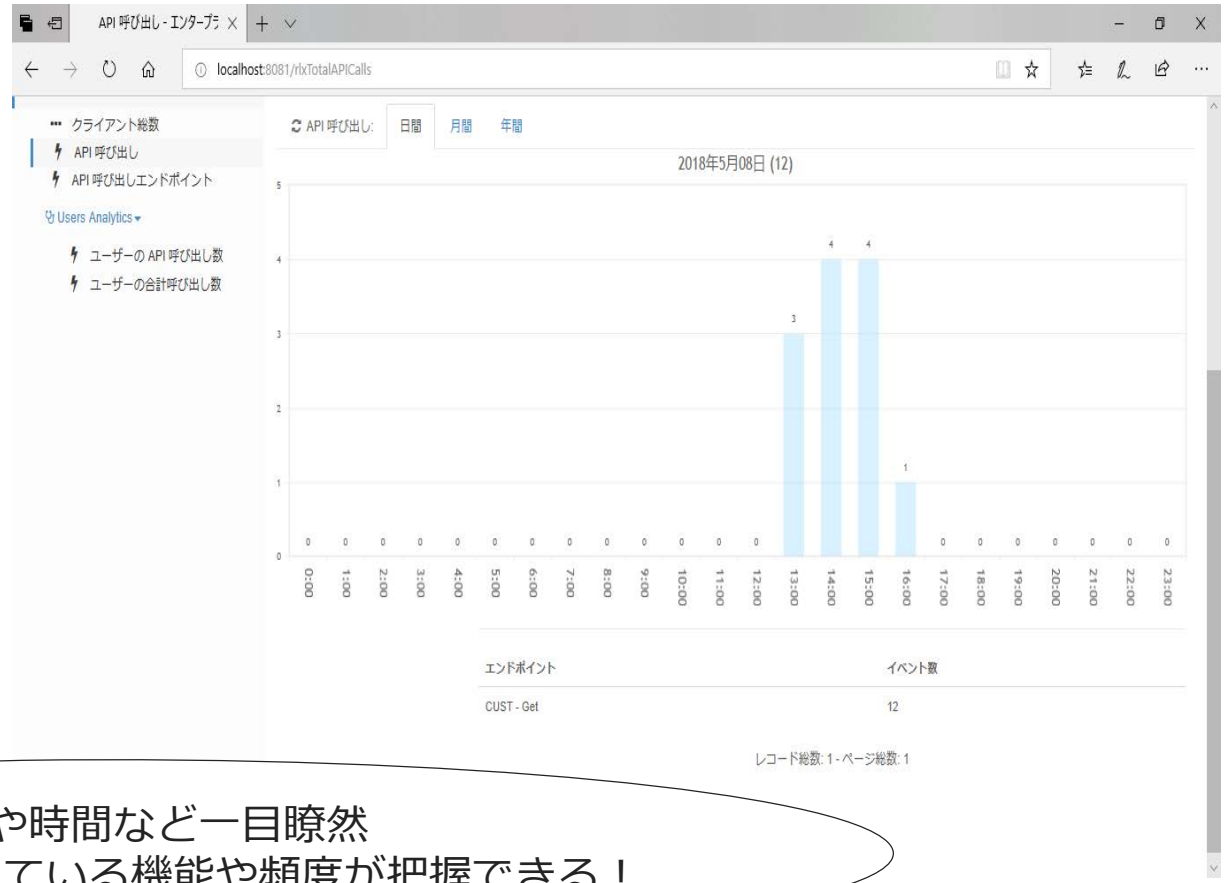
RAD Serverのコンソール

- コンソールのログイン
  ログインアカウント情報はiniファイルに 設定されています。
  C:¥Users¥Public¥Documents¥Embarcadero¥EMS¥emsserver.ini



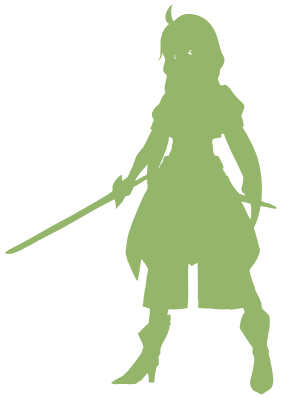
RAD Serverのコンソール

- コンソールで使える分析メニュー（アナリティクス）



APIが利用された回数や時間など一目瞭然
ユーザーが実際に使っている機能や頻度が把握できる！

中間サーバの活用範囲を拡張



embarcadero®
DEVELOPER CAMP

Enterprise Connectorsを利用したクラウドサービス等の連携

- Enterprise Connectorsとは？
  DelphiからSalesforceやAWS、SAP、ERP、Office 365、Googleドキュメント、ビッグデータDB、決済、ソーシャルサービスなど、80種類を超えるクラウドデータやエンタープライズサービスへのアクセス機能を追加できるパッケージソリューション

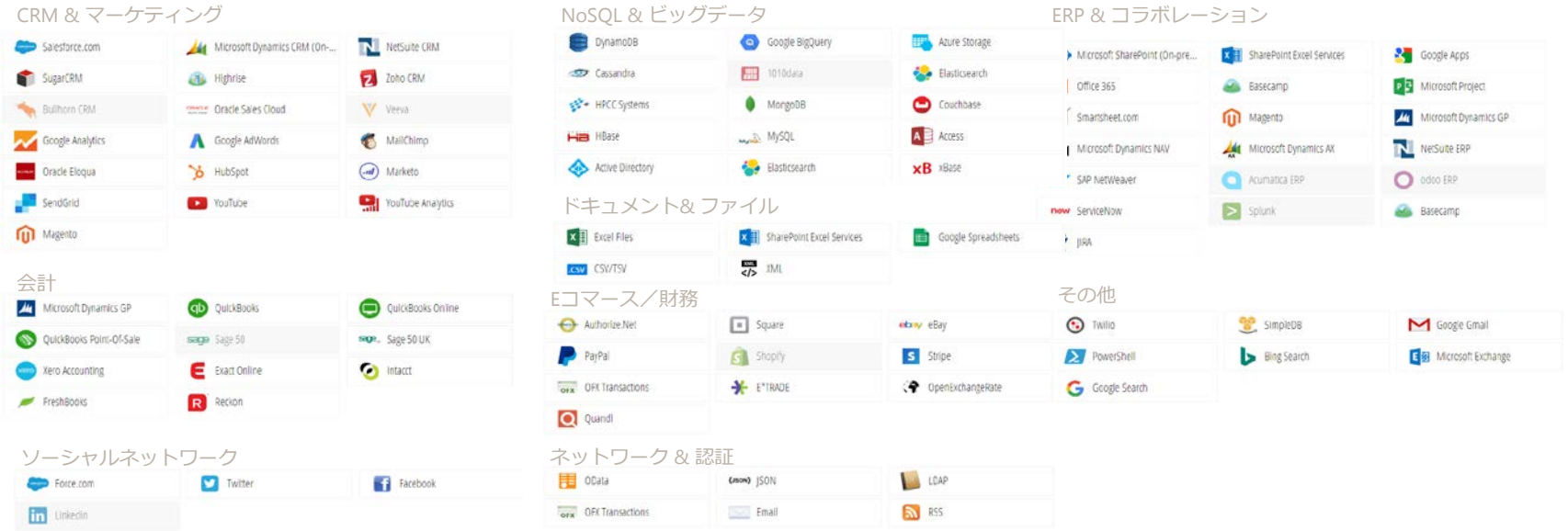
従来のDBアプリ開発スキルで簡単に実装が可能

Delphiに標準搭載されたDBエンジン「FireDAC」に対応しており、各サービスを利用するためのAPIは、コンポーネントによって提供される。
開発者はこれまでのデータベースアプリケーション開発スキルでクラウドデータ等を簡単に扱うことが可能。

Enterprise Connectorsを利用したクラウドサービス等の連携

- Enterprise Connectorsを中間サーバで利用するメリット
  - 社内DBだけではなく、クラウドサービスやパッケージのデータが連携できる
  - FireDACのコンポーネントで開発できるので、習得や調査が不要
  - クラウドサービスやパッケージのAPI変更にプログラム対応が不要
  - 中間サーバに実装すれば各クライアントでの導入や設定が不要

開発者の負担となる
工数・コストが大幅軽減して
アプリの拡張ができる



Enterprise Connectorsを利用したクラウドサービス等の連携

【n層アプリケーション】
Client/Serverアプリケーション

クライアント

Webアプリケーション

Webサーバ

クライアント（ブラウザ）

中間サーバ

モバイルアプリケーション

クライアント

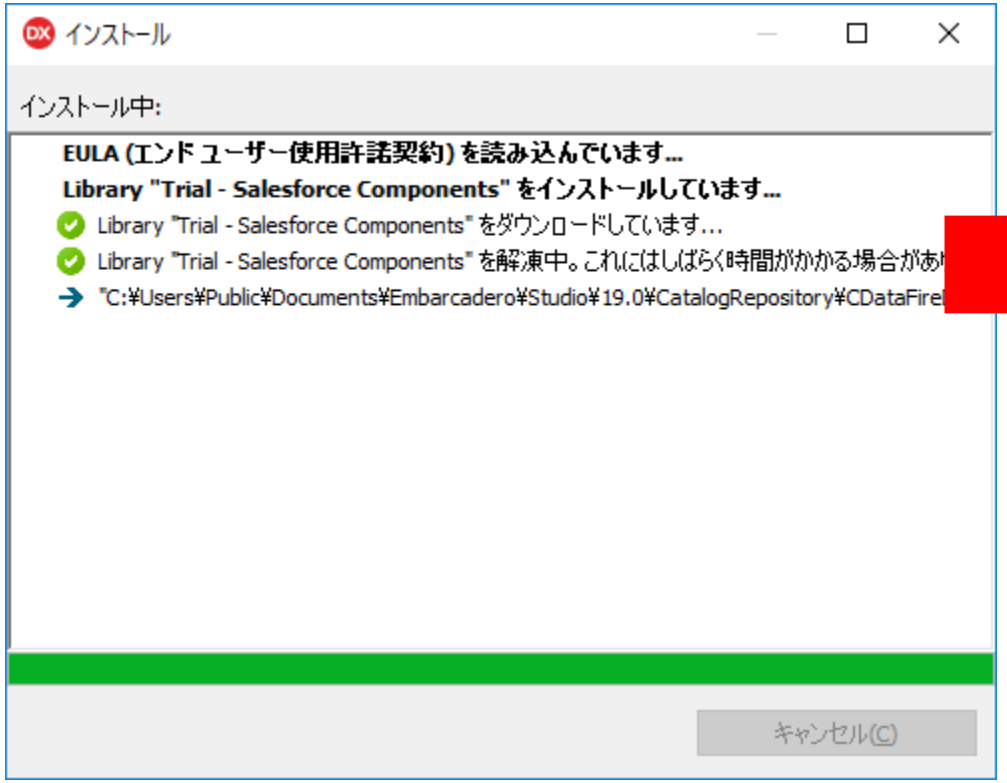
データベースサーバ

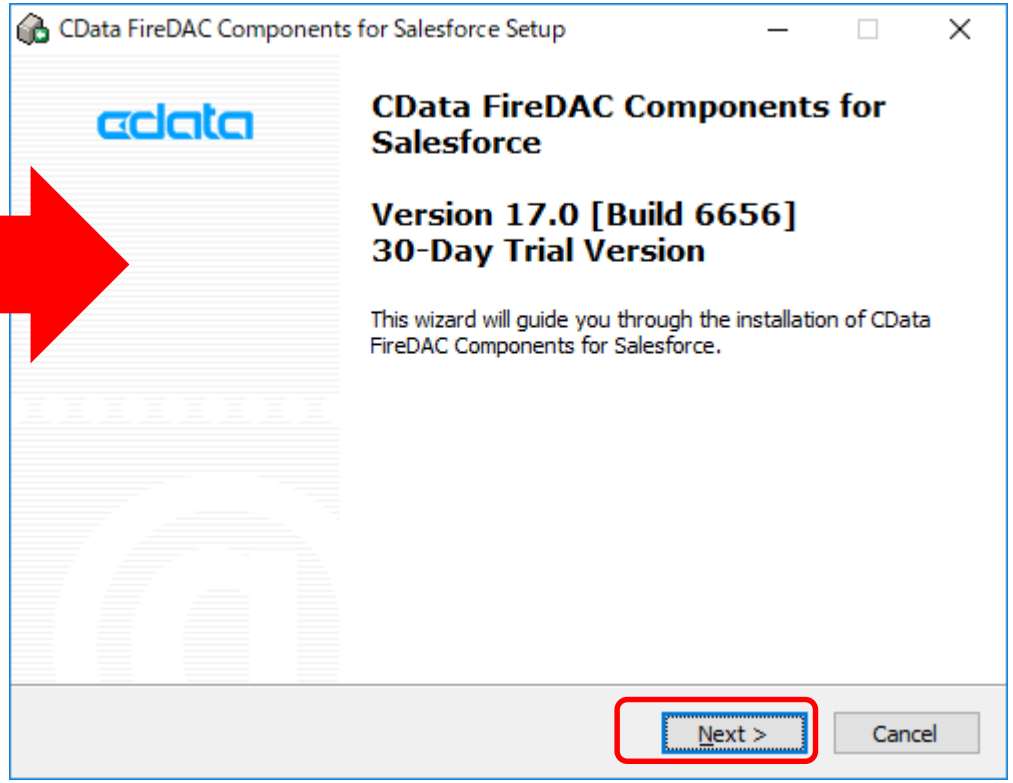
Enterprise Connectorsの組み込み例（Salesforce）

- 組み込み手順1　（※製品購入版はCData社サイトよりインストーラをDLしてインストールします）
  [ツール|GetItパッケージマネージャ]から対象のECを検索して選択

Enterprise Connectorsの組み込み例（Salesforce）

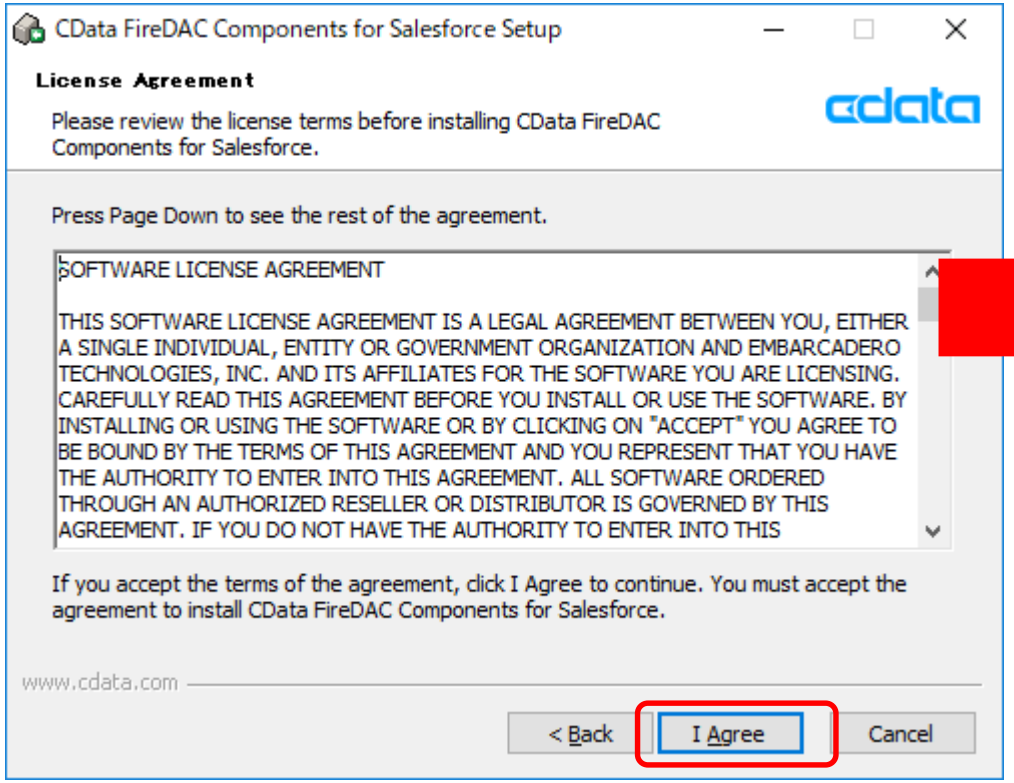
- 組み込み手順2
  自動ダウンロード・配置

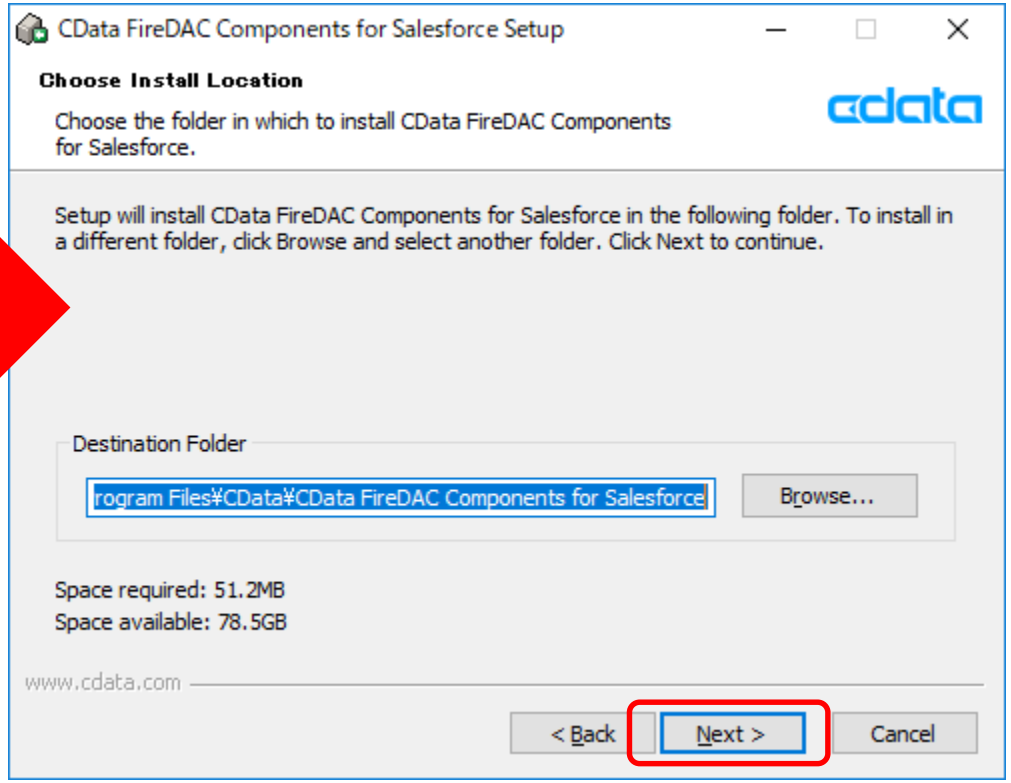




Enterprise Connectorsの組み込み例（Salesforce）

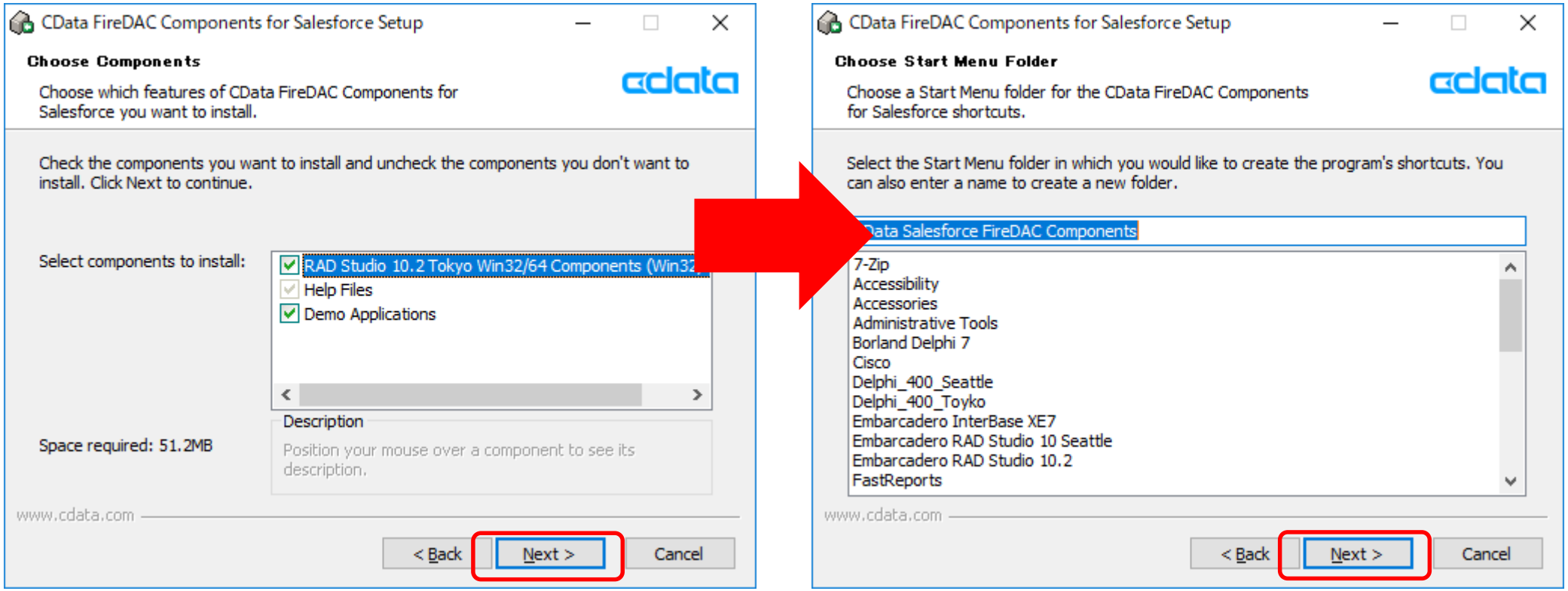
- 組み込み手順3
  インストール先を設定





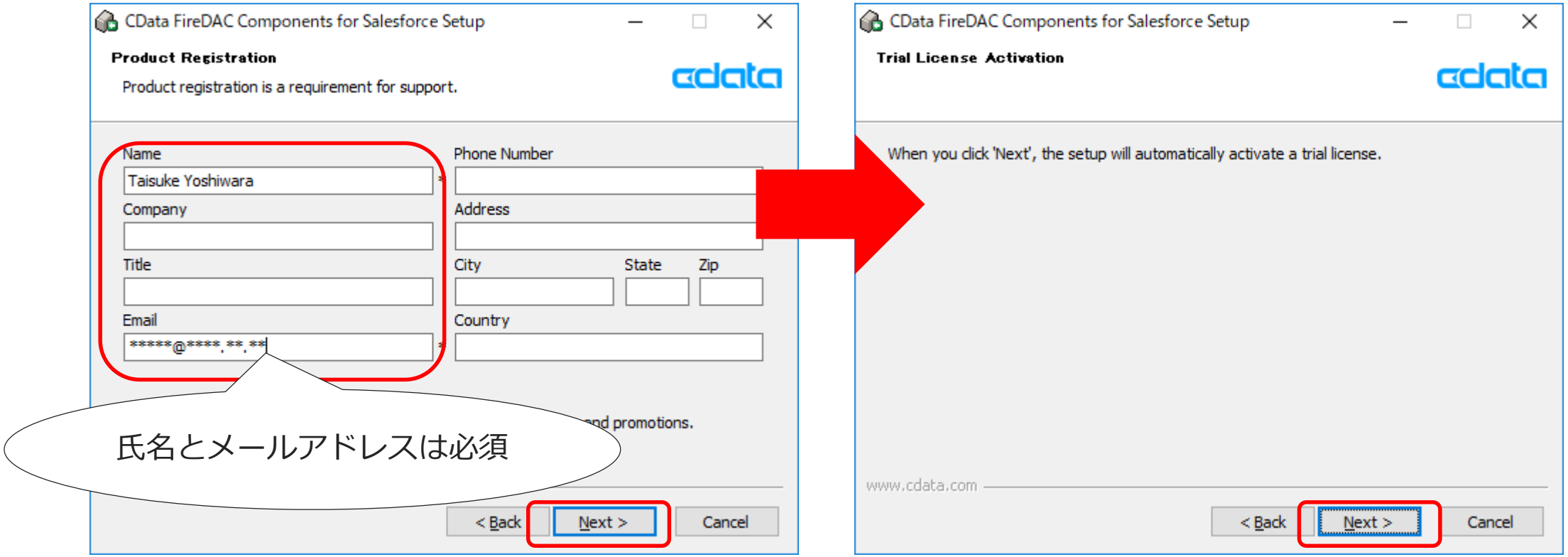
Enterprise Connectorsの組み込み例（Salesforce）

- 組み込み手順4
  インストールオプションの選択



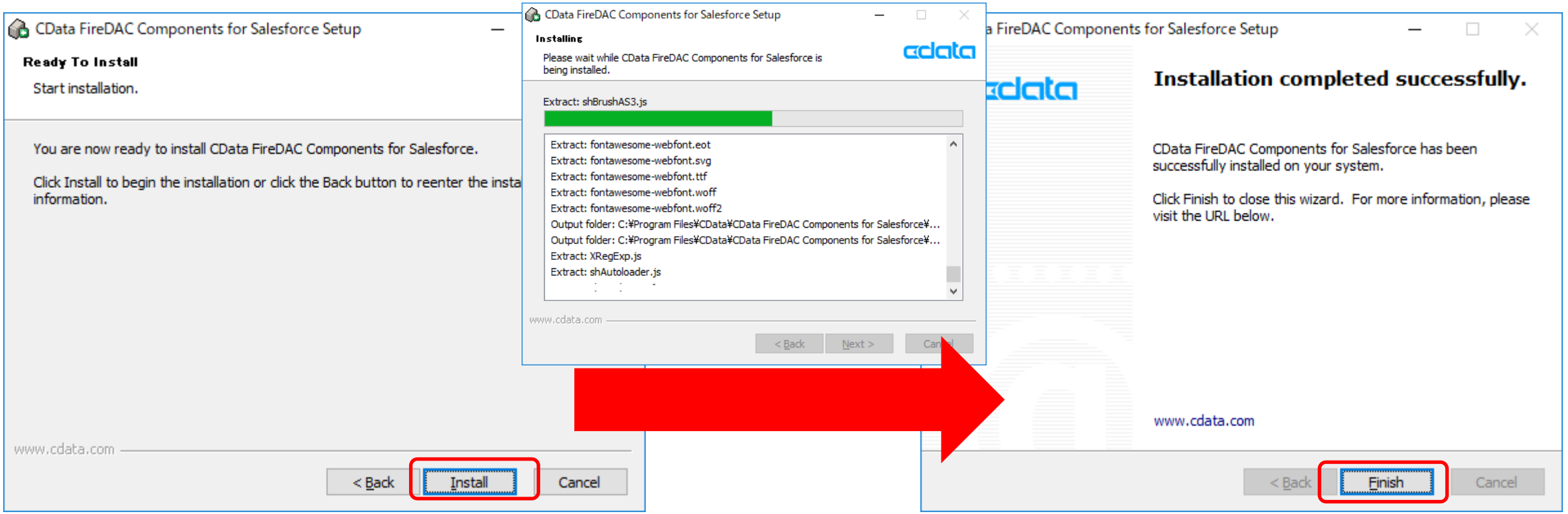
Enterprise Connectorsの組み込み例（Salesforce）

- 組み込み手順5
  導入ユーザーの登録



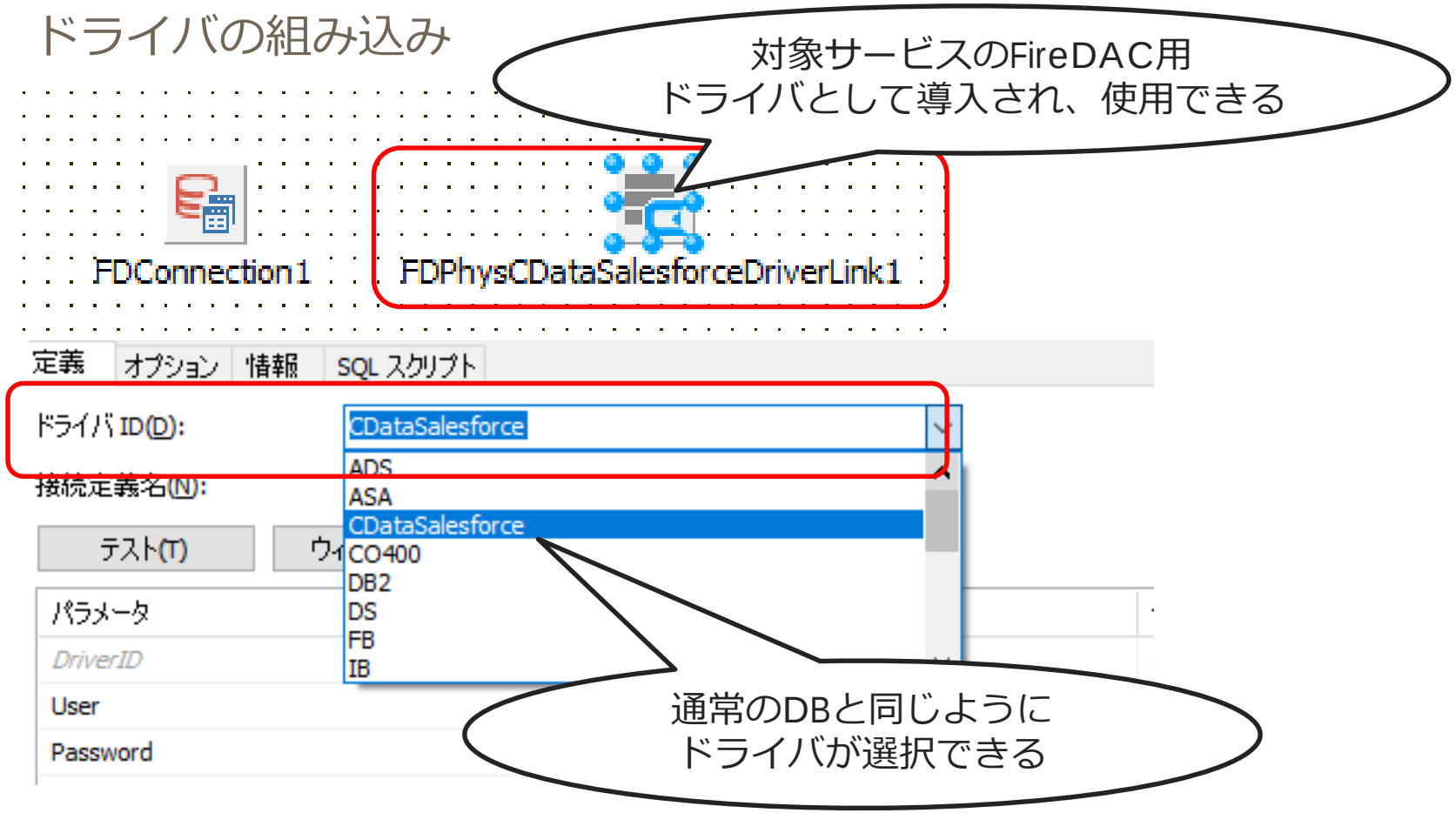
Enterprise Connectorsの組み込み例（Salesforce）

- 組み込み手順6
  インストールの実行



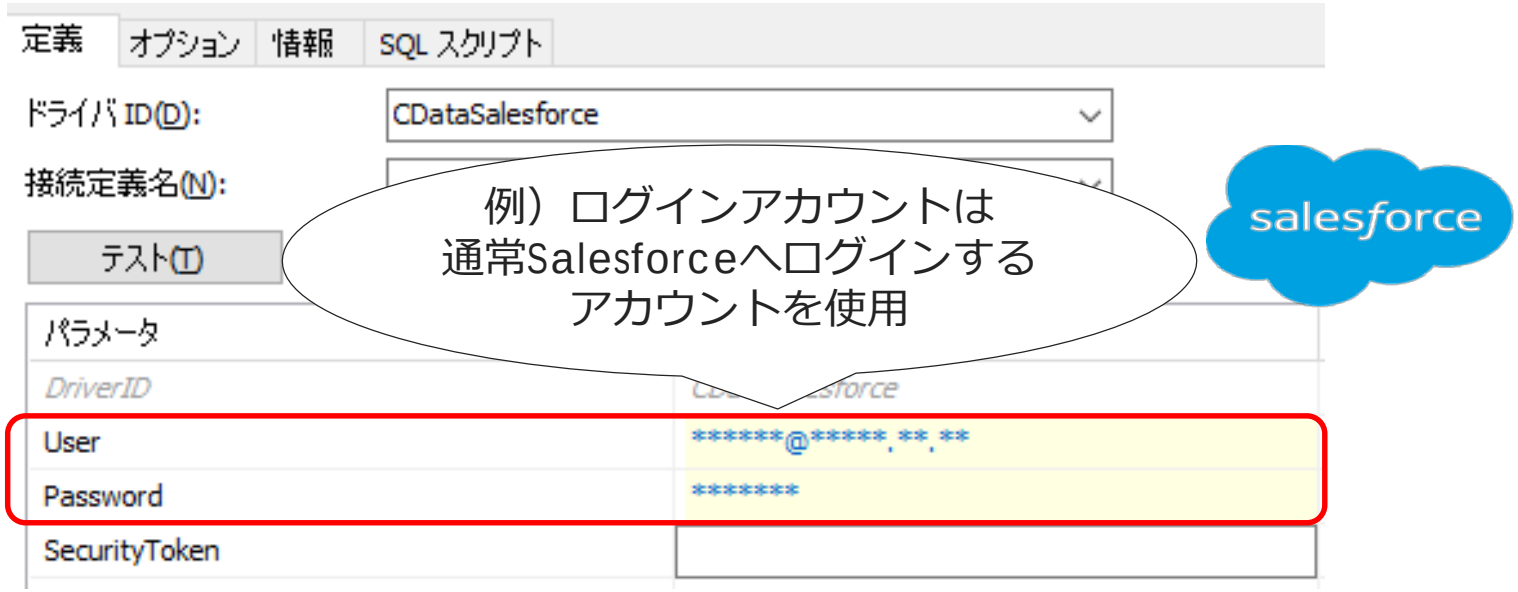
Enterprise Connectorsの組み込み例（Salesforce）

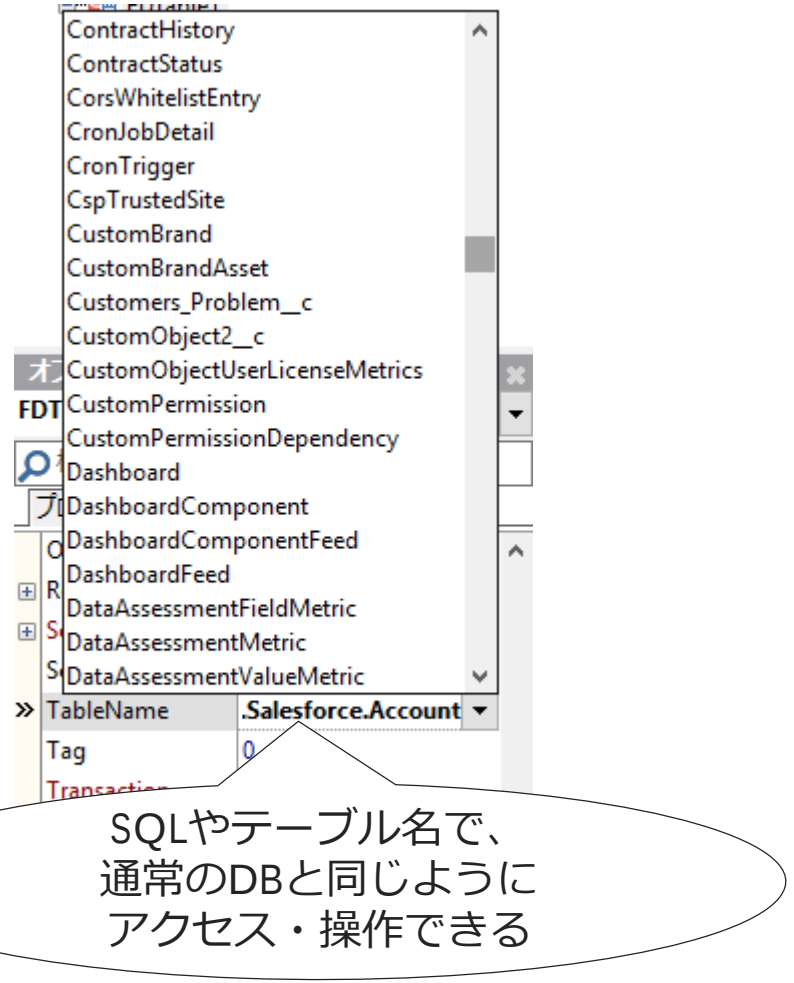
- 組み込み手順7
  ドライバの組み込み



Enterprise Connectorsの組み込み例（Salesforce）

- 組み込み手順8
  通常のFireDACのDB操作と同じ



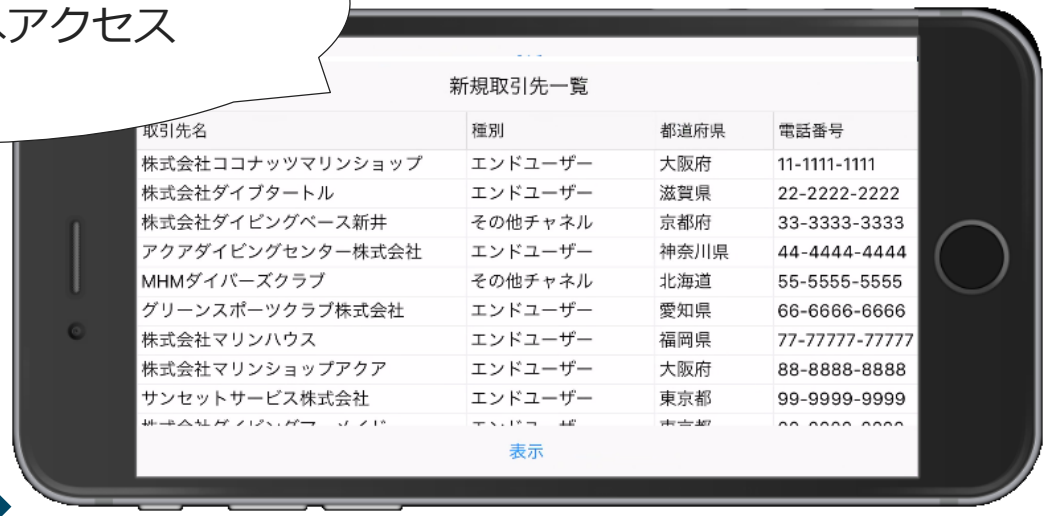


Enterprise Connectorsの組み込み例（Salesforce）

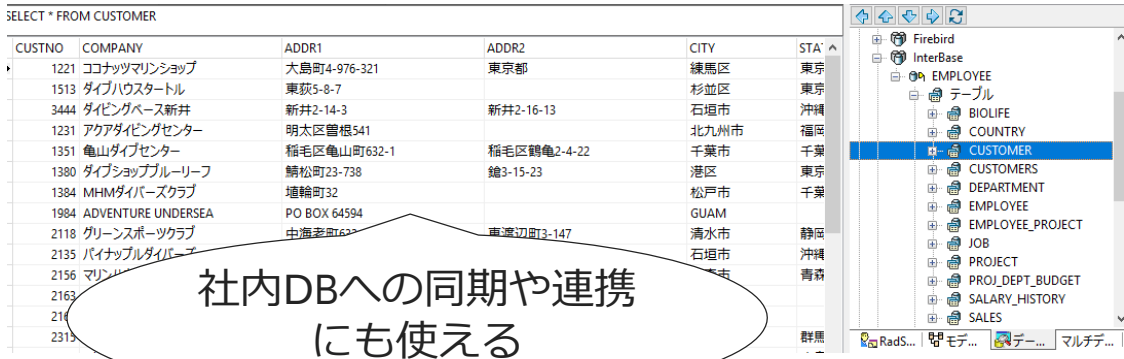
- 実行

実際のsalseforceデータ

Delphiアプリでは自由に
Saleseforceへアクセス



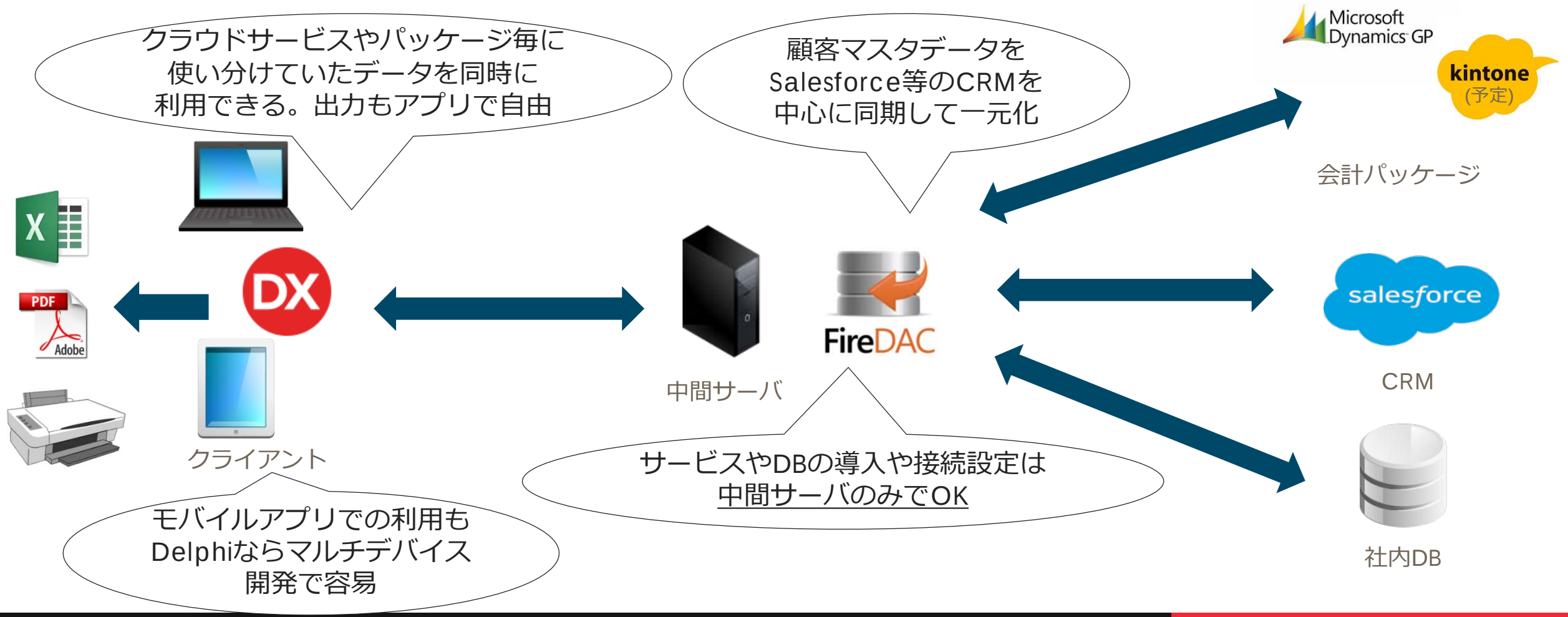
社内DB



社内DBへの同期や連携
にも使える

Enterprise Connectorsを利用した拡張

- クラウドサービスやパッケージとの連携・同期例



Enterprise Connectors利用時の補足

- 補足：通常のRDBMSと異なる点

  ・ビューやプロシージャなど、用意されているものしか基本使えない。
  （自由に作成はできない）

  ・トランザクションは基本使えない。

  ・ネットワークやサービス側の負荷が応答パフォーマンスに影響する。

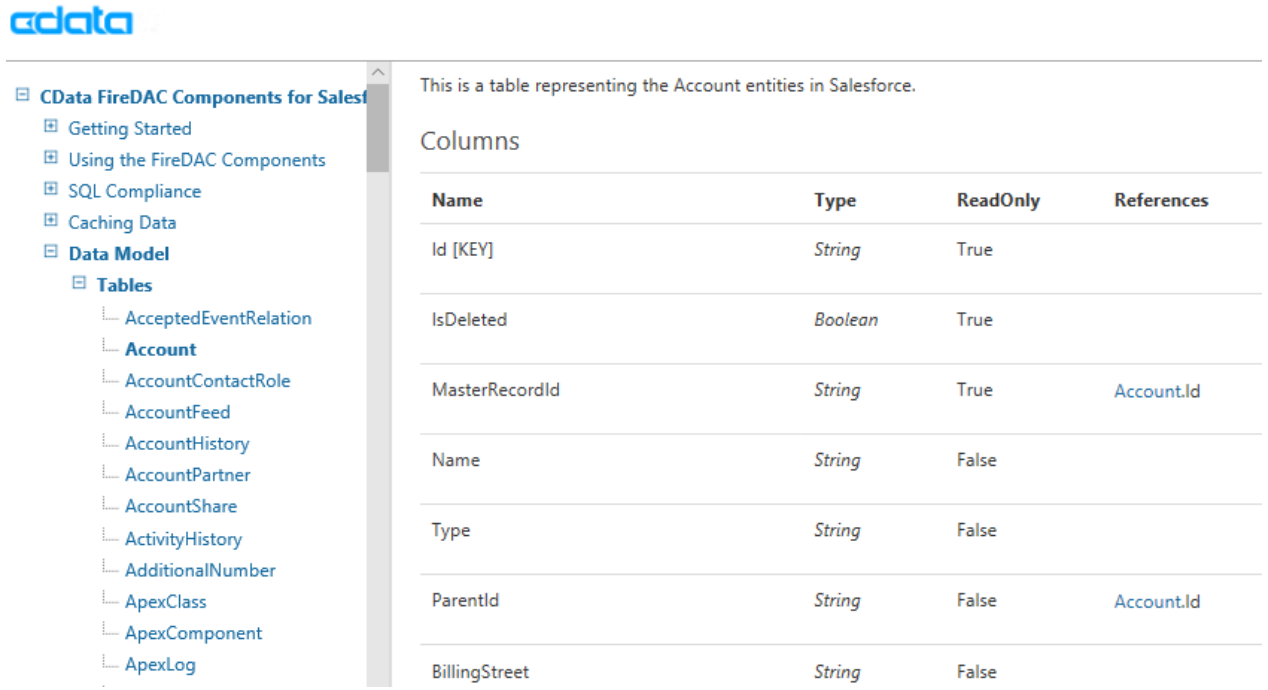
  ・必要なデータだけを絞り込む(select＊的な検索は行わない)

  Enterprise Connectors の select 実行時に where で使う検索条件は、接続先サービスの検索仕様で指定する

Enterprise Connectors利用時の補足

- 補足：テーブル等の定義
  アクセスできるテーブル（の形で用意された）定義等はローカルのヘルプに詳しく掲載されている。
  定義だけで分からない不明なものは実データを見て判断する方法も有効。

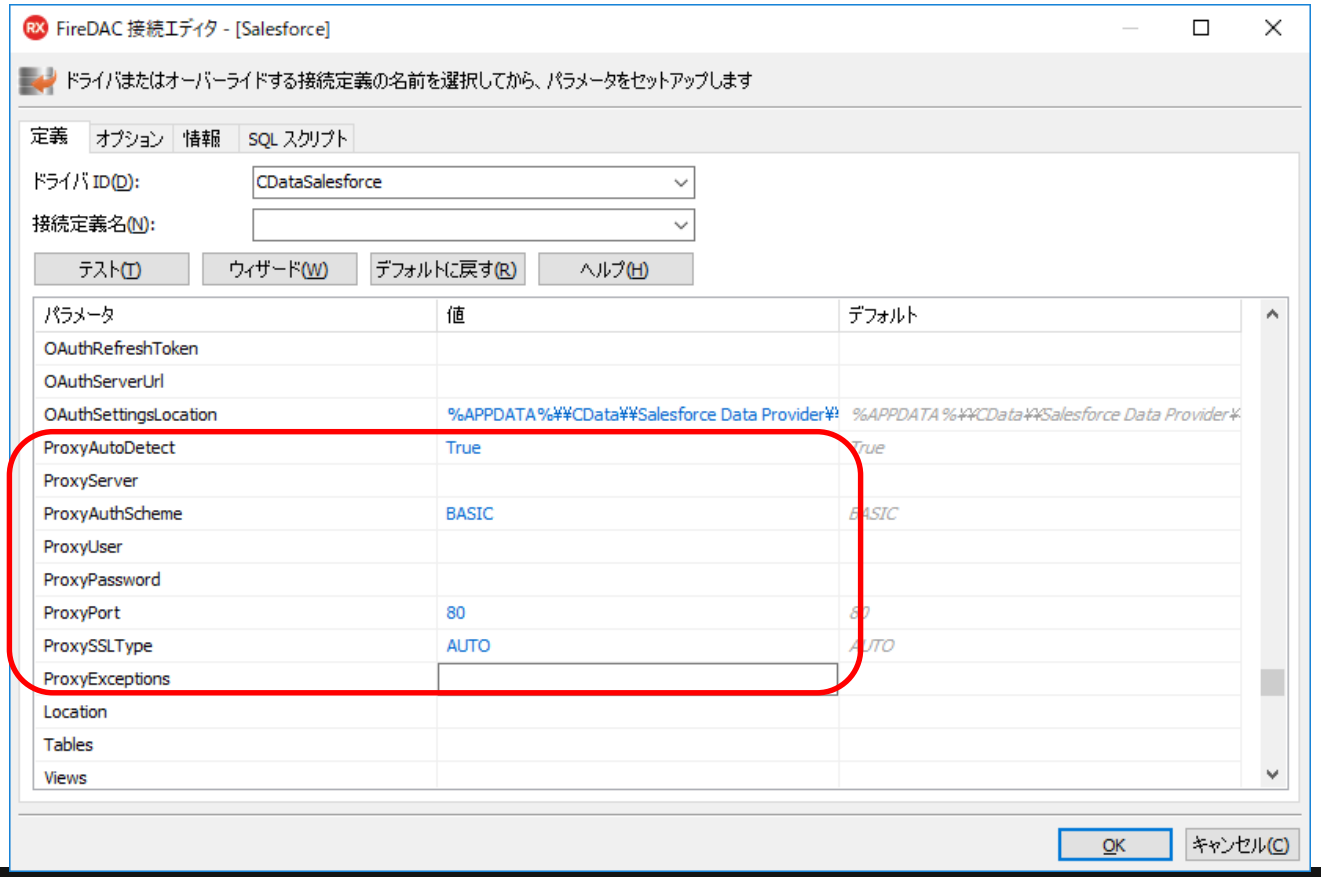
ローカルのHelp



Enterprise Connectors利用時の補足

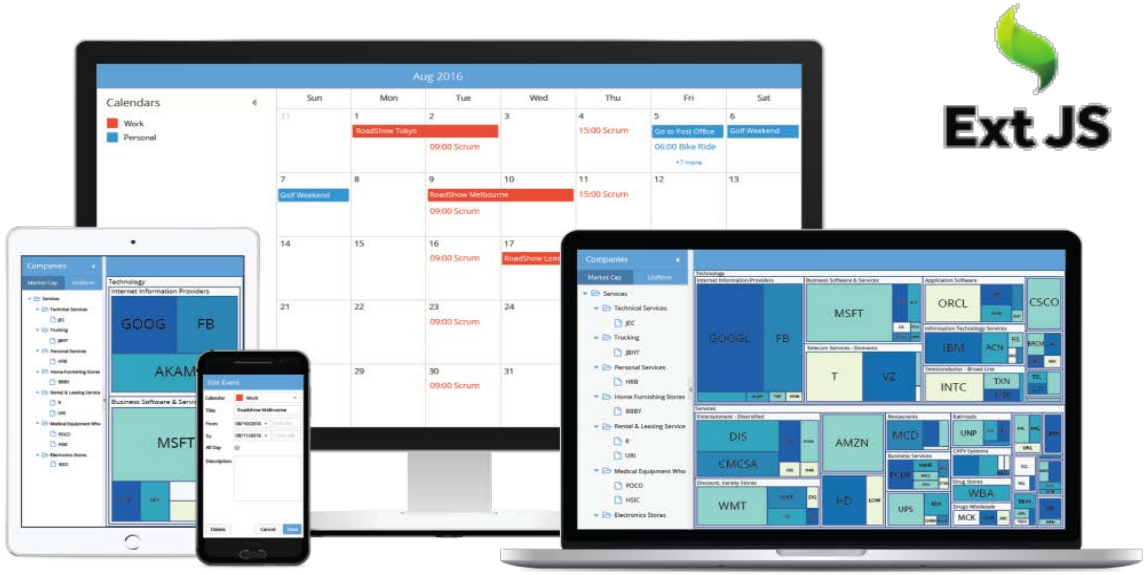
- 補足：プロキシ対応

  プロキシの設定もTFDConnectionのパラメータで対応可能
  （サーバを配置する環境によっては便利）



Sencha等のWebアプリケーションへの連携

- Delphi以外のアプリケーションでの利用
  中間サーバから一般的なJSON形式でRESTの結果を提供すれば、
  Sencha等で作成したWebアプリケーションで利用することも可能。
  (Senchaとの連携については午後のセッションをご参考ください)



Ext JS

Sencha等のWebアプリケーションへの連携

- 一般的なJSON形式でRESTの結果を戻す

```
procedure TCUSTResource1.Get(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
var
  JSONResponse: TJSONObject;
  JSONArray: TJSONArray;
  JSONRecord: TJSONObject;
  aField: TField;
  count: Integer;
begin
  JSONResponse := TJSONObject.Create;
  JSONArray    := TJSONArray.Create;
  count := 0;
```

Sencha等のWebアプリケーションへの連携

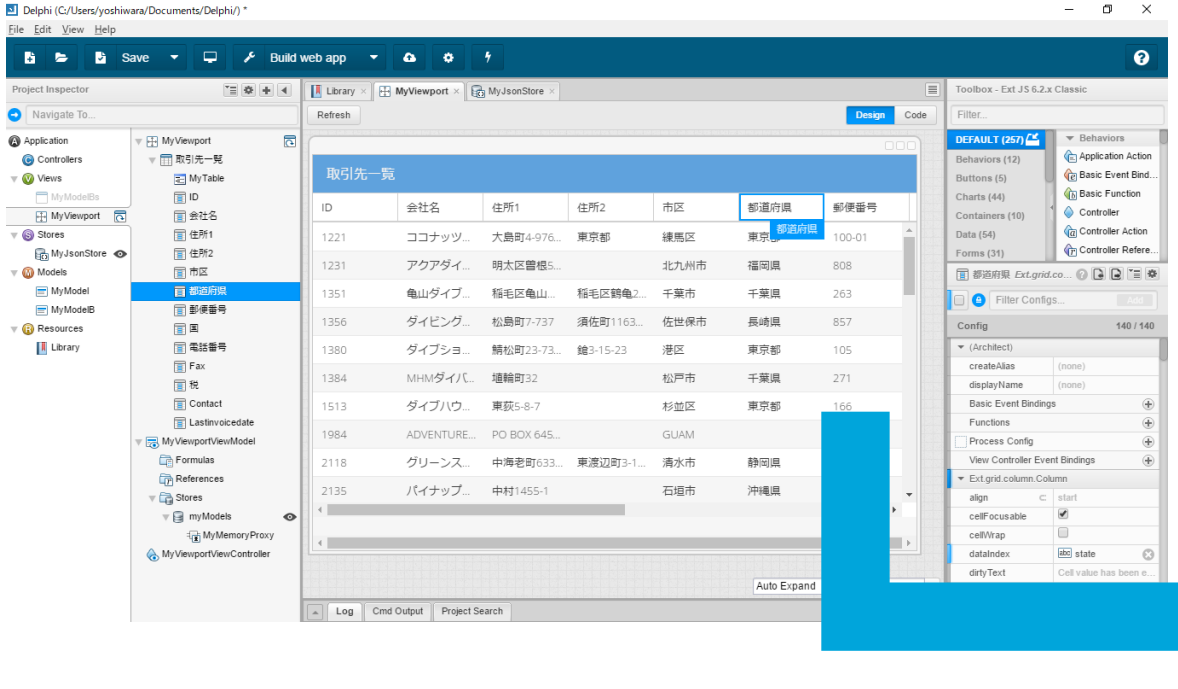
- 一般的なJSON形式でRESTの結果を戻す

```
  // データセットの結果を１件づつ取得してJSONArrayに追加する
  FDTable1.Open;
  while (not(FDTable1.Eof)) do
  begin
    JSONRecord := TJSONObject.Create;

    for aField in FDTable1.Fields do
    begin
      JSONRecord.AddPair(LowerCase(aField.FieldName),aField.AsString);
    end;

    JSONArray.Add(JSONRecord);
    FDTable1.Next;
    inc(count);
  end;

  AResponse.Body.SetValue(JSONArray,True);
end;
```

Sencha等のWebアプリケーションへの連携

- SenchaからのアクセスでもDelphiと同じようにデータを利用

Sencha Architect



ブラウザで実行

クロスドメインがネックになる場合は、RAD Serverのemsserver.iniで設定するか、JSONPで返却する等の工夫が必要
Marco Tech Blogが参考になります
http://blog.marcocantu.com/blog/2017-october-ajax-jsonp-radserver-extjs.html

まとめ

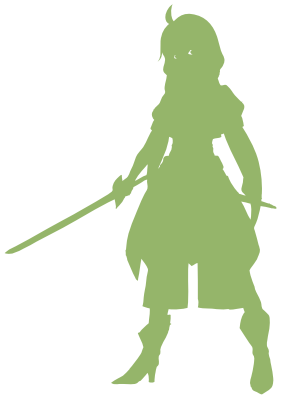
- 多層アプリケーションによる機能の分離はメリットが多く、
  マルチデバイスでシステムを展開していく上では非常に有効。
- DataSnap Serverはサーバ開発用のSDK。開発できる自由度が高い。
- RAD Serverは標準で豊富な機能が提供されるサーバ。
- Enterprise Connectorsを中間サーバで利用すれば、
  クラウドサービスやパッケージなどの連携が容易。
- 中間サーバのサービスで返却するJSON形式を配慮すれば
  Delphi以外のアプリケーションでも活用できる。

THANKS!

www.embarcadero.com/jp

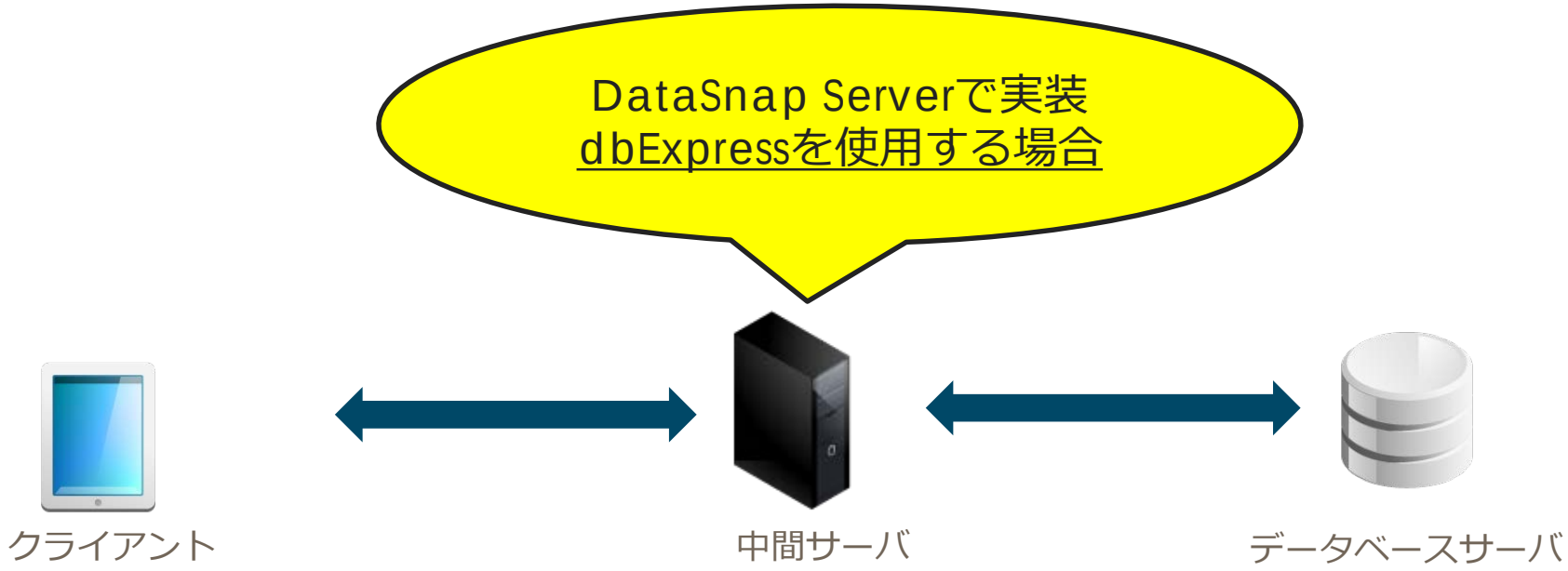
第35回 エンバカデロ・デベロッパーキャンプ

補足：DataSnap Serverを使った実装手順（dbExpress）



embarcadero®
DEVELOPER CAMP

補足：DataSnap Serverを使った実装手順（dbExpress）

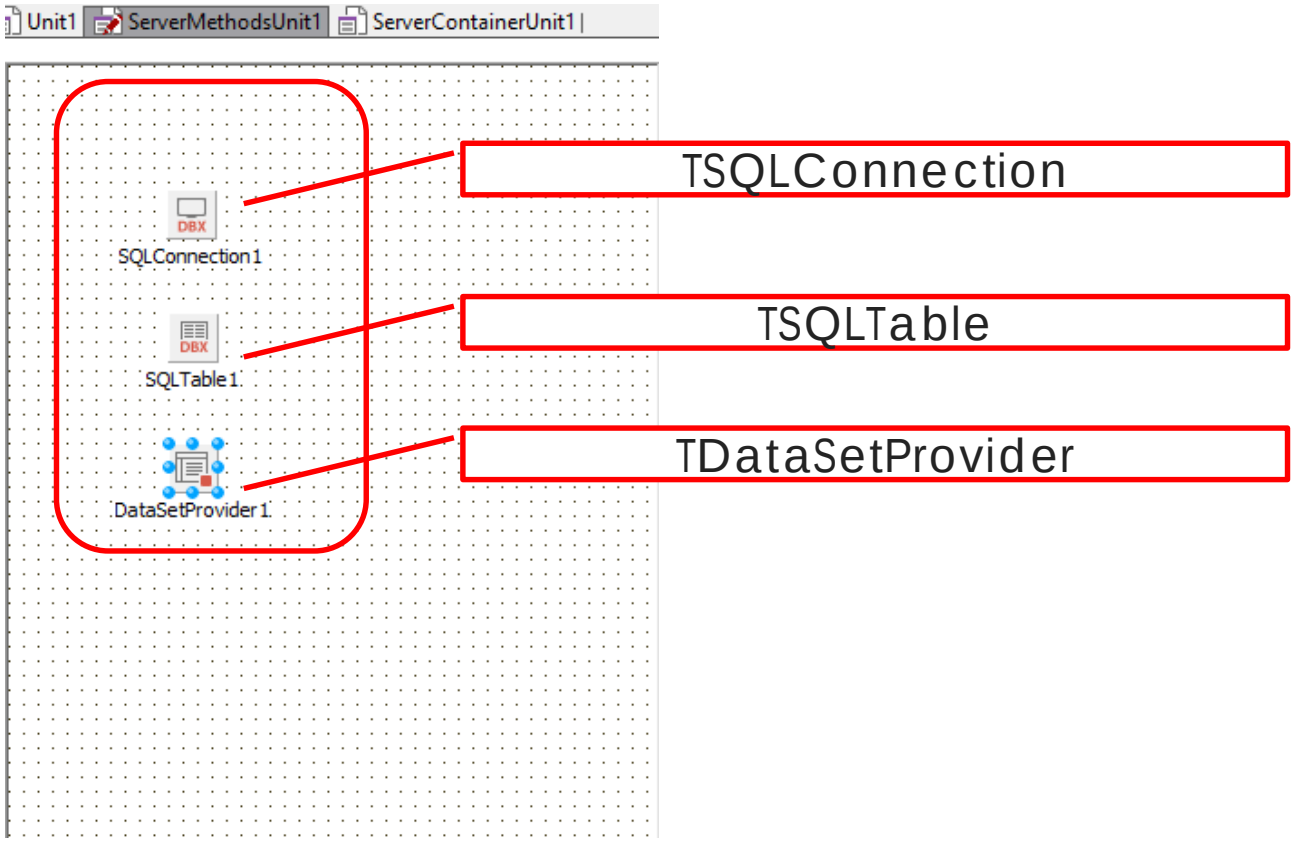


補足：DataSnap Serverを使った実装手順（dbExpress）

- 実装手順1

サーバ機能のコンポーネント配置

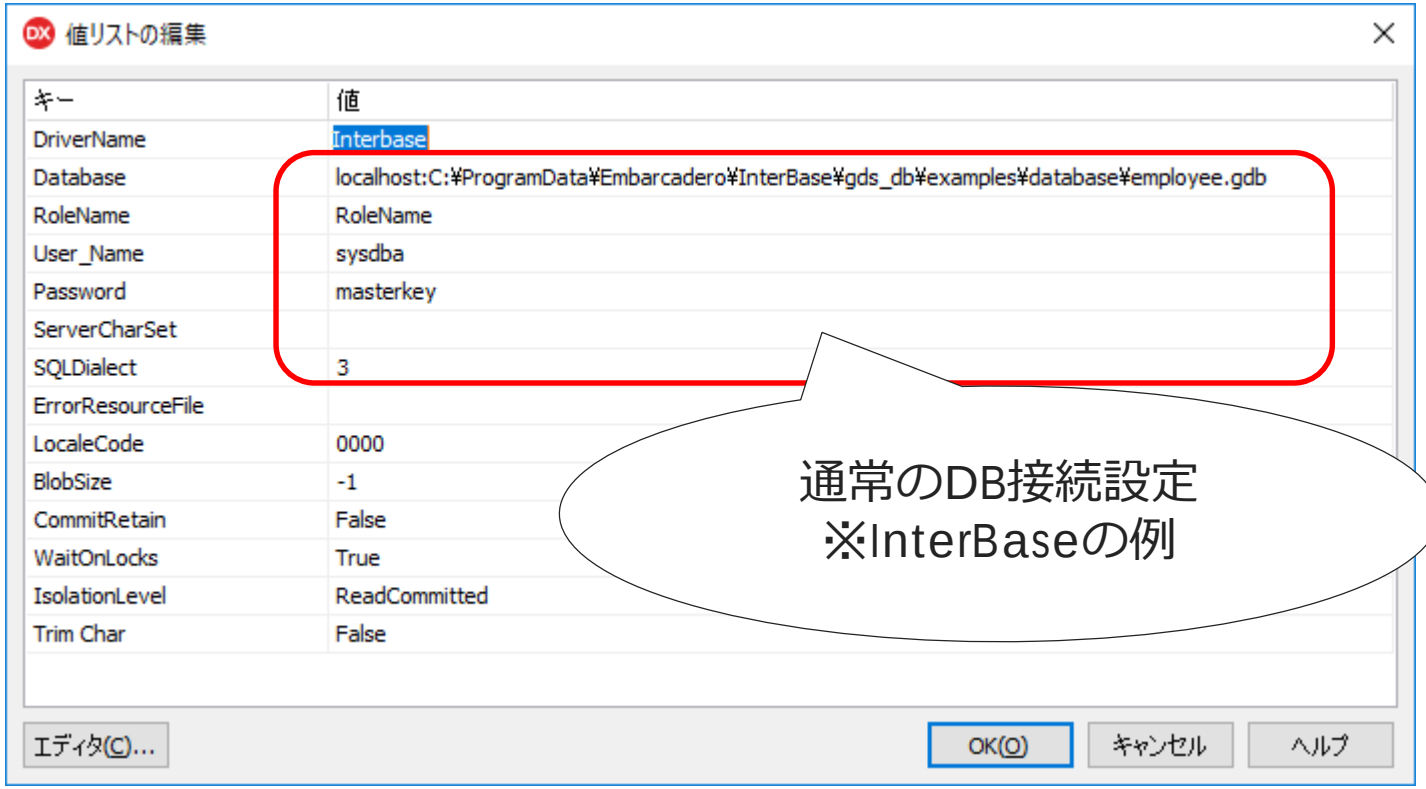
ServerMethodsユニットに機能を実装



補足：DataSnap Serverを使った実装手順（dbExpress）

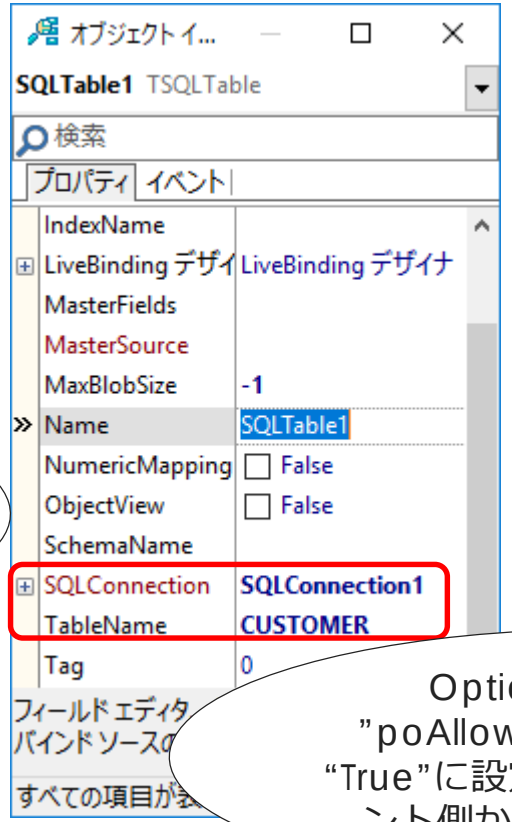
- 実装手順2
  コンポーネントの設定

TSQLConnection

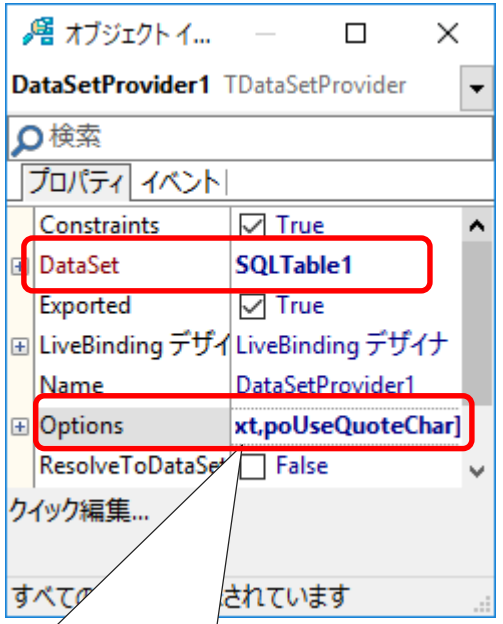


通常のDB接続設定
※InterBaseの例

TSQLTable

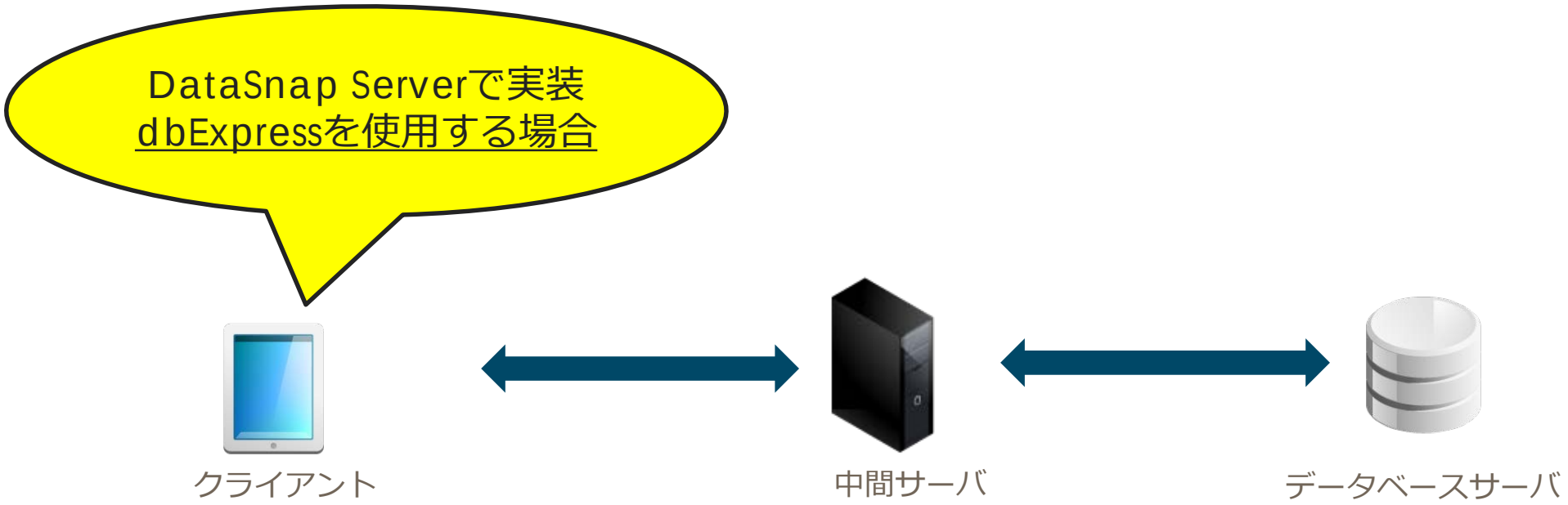


TDataSetProvider



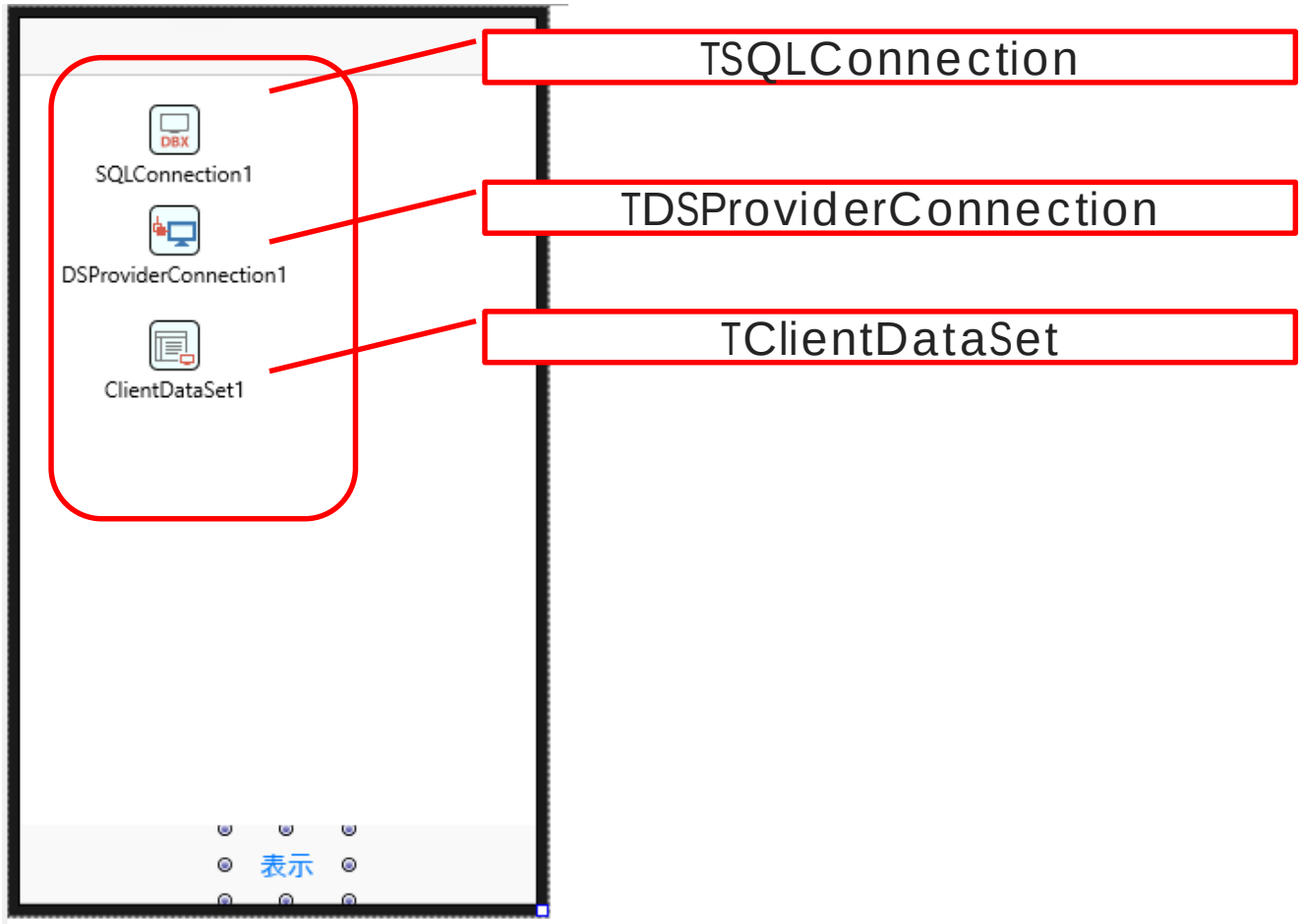
Optionプロパティの
”poAllowCommandText”を
“True”に設定しておくとクライアント側からSQLの指定も可能

補足：DataSnap Serverを使った実装手順（dbExpress）



補足：DataSnap Serverを使った実装手順（dbExpress）

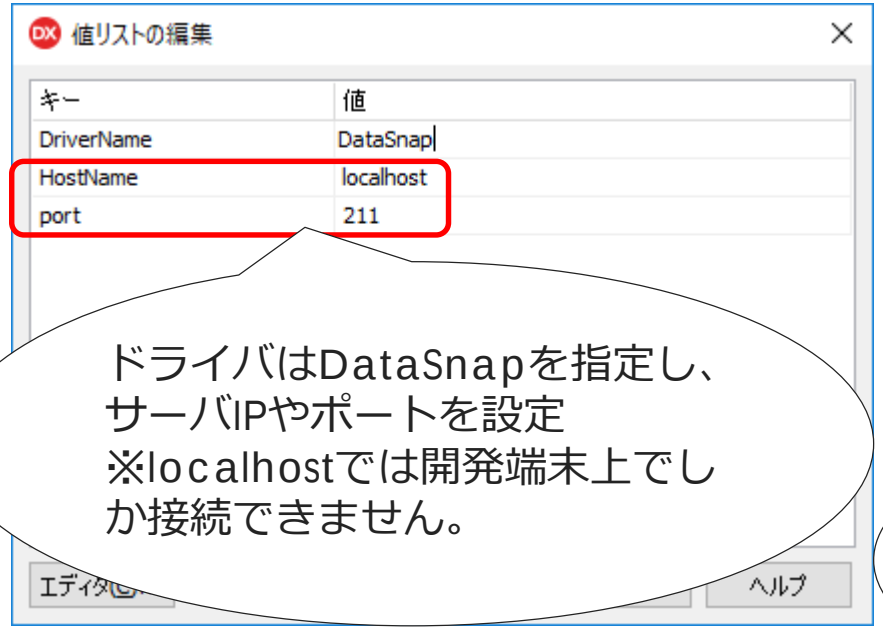
- 実装手順3
  クライアントアプリケーションのコンポーネント配置



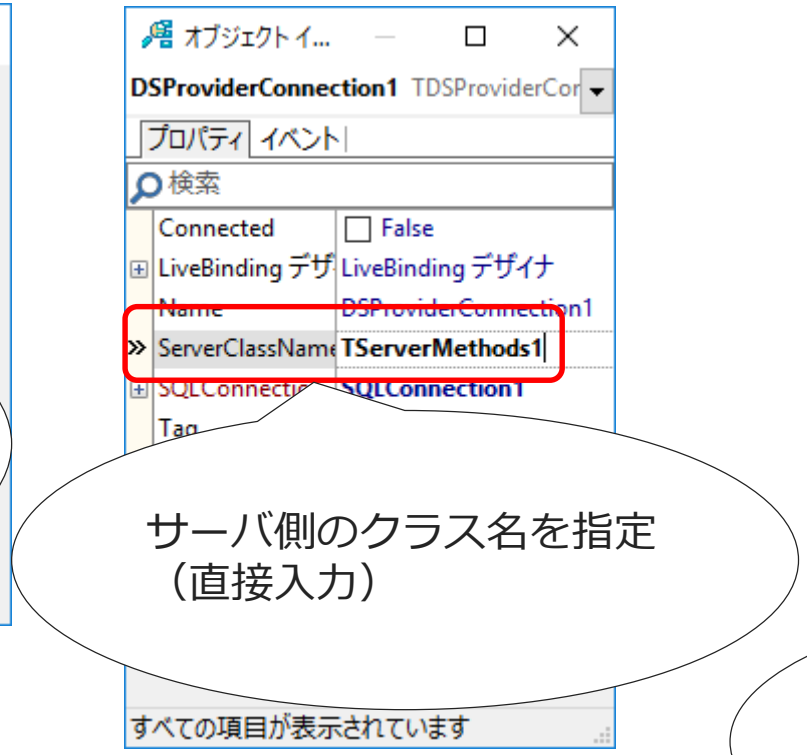
補足：DataSnap Serverを使った実装手順（dbExpress）

- 実装手順4
  コンポーネントの設定

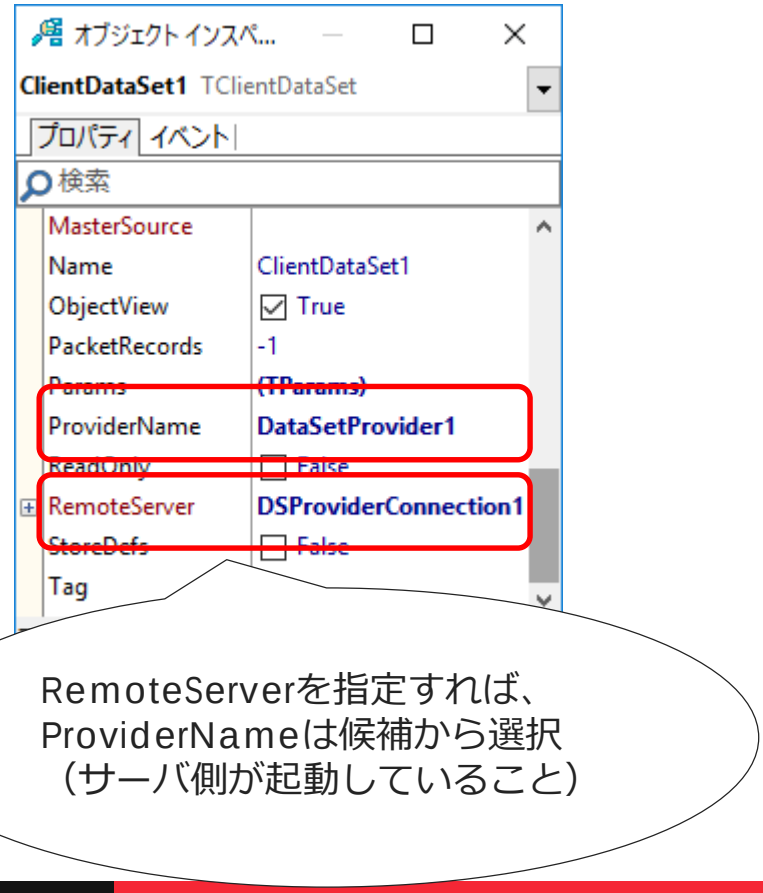
TSQLConnection



TDSProviderConnection



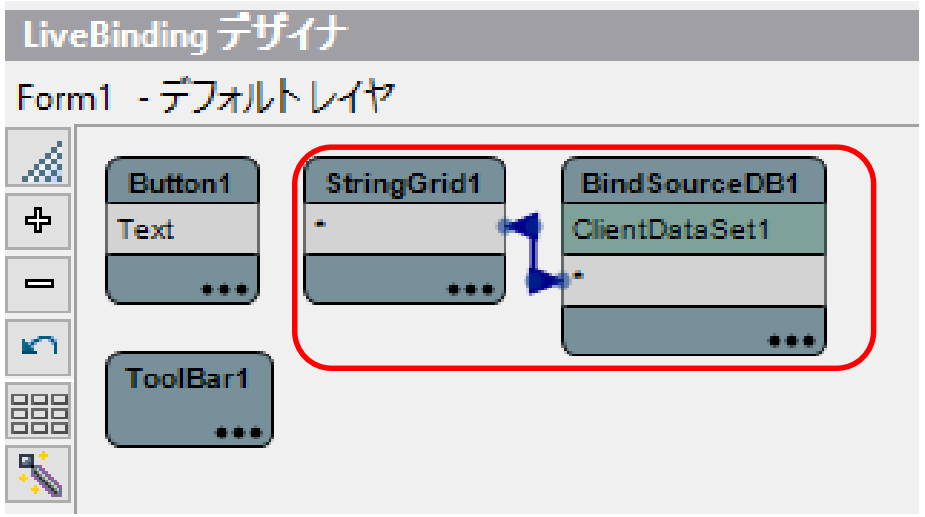
TClientDataSet



補足：DataSnap Serverを使った実装手順（dbExpress）

- 実装手順5
  コンポーネントの設定

ライブバインディング設定



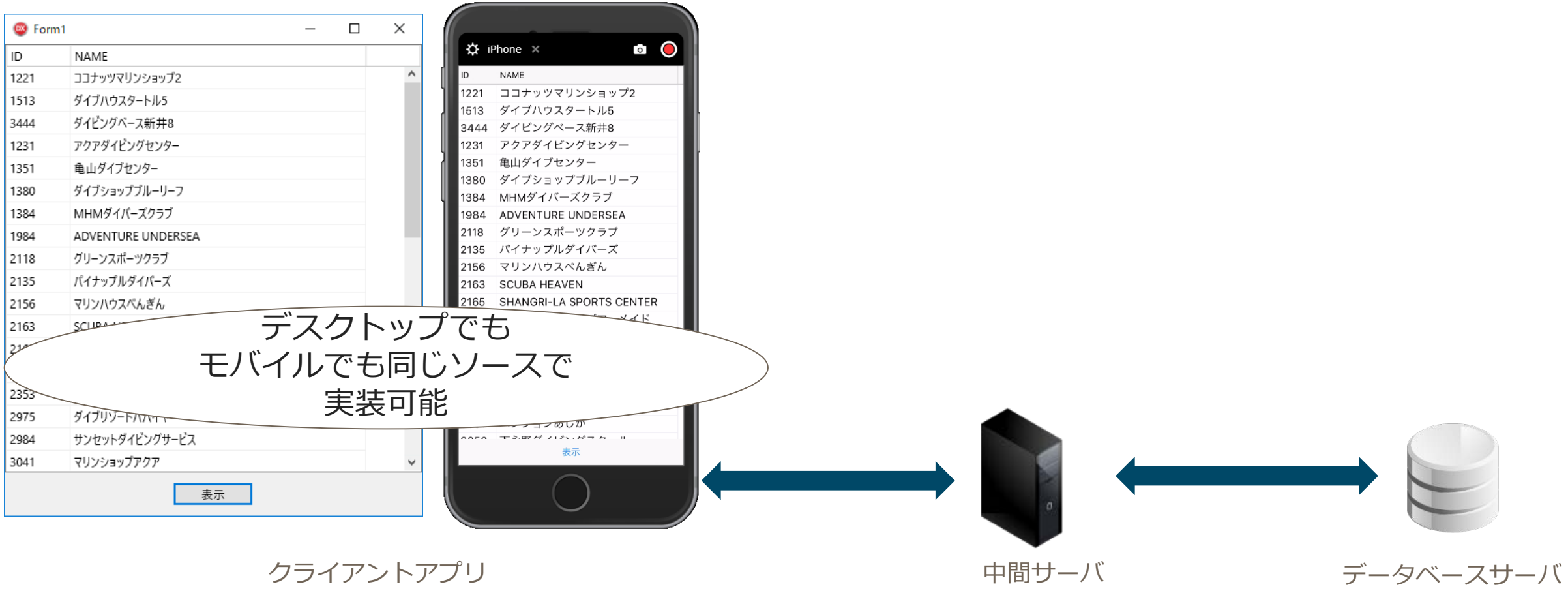
補足：DataSnap Serverを使った実装手順（dbExpress）

- 実装手順6
  データ取得の機能を実装

```
procedure TForm1.Button1Click(Sender: TObject);
Begin
  //設定しているClientDataSetを開くだけ
  ClientDataSet1.Open;
end;
```

補足：DataSnap Serverを使った実装手順（dbExpress）

- 実行



補足：DataSnap Serverを使った実装手順（dbExpress）

- DBエンジンコンポーネントの組み合わせ

サーバ側をFireDACのコンポーネントを使い、クライアント側をdbExpressの
コンポーネントで開発することも可能。
実装内容はコンポーネントが変わるだけでdbExpressの作り方とほぼ同じです。

例）第29回のエンバカデロ・デベロッパーキャンプの
「こんなに簡単！Delphi によるiOS/Android業務アプリケーション開発！」
のセッション内容がこの構成になっています。アーカイブの資料等をご参考ください。
https://www.embarcadero.com/jp/events/developer-camp/archive

DataSnap Serverで可能な組み合わせ

組み合わせ	
クライアント側（サーバ接続）	サーバ側（DB接続）
FireDAC	FireDAC
dbExpress	FireDAC
dbExpress	dbExpress