FireDAC

人と技術をつなぐ

【E4】Delphi/C++テクニカルセッション

今さら聞けない!?「FireDAC入門」

～ インーメモリーデータベース編 ～

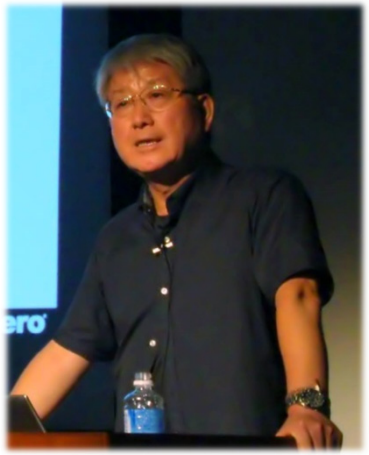
2015年5月21日

田中 芳起

Ver.1.0.2

自己紹介

名前：田中 芳起（たなか よしき）



- 中堅SIerでパッケージシステムの開発／プロジェクト管理／品質管理 等の仕事に従事
- 26th デブキャンプで「はじめてのFireDAC」の講師を担当
- 29th デブキャンプで「Visual NAVI」の講師を担当
- Delphiとは 1.0US版 からの付き合い

- ホームページ　：http://www.avsoft.jp/
- ブログ　：http://avsoft.typepad.jp/blog/
- Facebook　：https://www.facebook.com/yoshiki.tanaka.942/
  ：https://www.facebook.com/VisualNavi
  ：https://www.facebook.com/groups/864814060228437/

アジェンダ

➢ はじめに

- スマートデバイスと「データ連携」するための技術
- 技術のポジショニング
- イン－メモリデータベース
- JSONってなにもの?

➢ FireDACのおさらい

- FireDACとは?
- FireDACを使用するメリットは‥
- FireDACアプリケーションの構造
- Delphi XE5での変更点
- Delphi XE7 での変更点
- BDEとFireDAC 主要コンポーネントの比較
- データベースクラスの継承関係

アジェンダ

FDMemTable

- テーブルを生成する
- 一般的な「Xml ファイル」は読込めない！
- 出力した「Xml ファイル」の構造は‥
- 自力で「Xml ファイル」を 読み／書き してみると‥
- 拙作クラス (TDataMigration) で読み／書きしてみると‥
- 別のデータセットから、全てのレコードをコピーする
- 別のデータセットのデータを共有する
- データセットの状態を調べる
- 取得するデータを絞り込む
- TFDMemTable のまとめ

SQLite

- 接続の設定
- テーブルを生成する
- 高速にデータを挿入する
- 動的問い合わせ
- データベースをバックアップする
- SQLをトレースする

アジェンダ

- **EMSサーバー／クライアント**
- Webアプリケーション／DataSnapの構成
- EMSサーバー／クライアントの構成
- EMS って、なに‥?
- REST って、なに‥?

- DEMONSTRATION ～DataSetを渡す～
- デモンストレーションの説明
- ウィザードを使ってRESTリソースを作成する
- ウィザードが作成した RESTリソース の内容は‥
- RESTメソッドの実装（Get／Post）
- クライアントからRESTメソッドを呼び出す
- コンポーネントの関係とデータの流れ
- クライアント側の実装ポイントを整理
- EMSサーバーウィンドウ

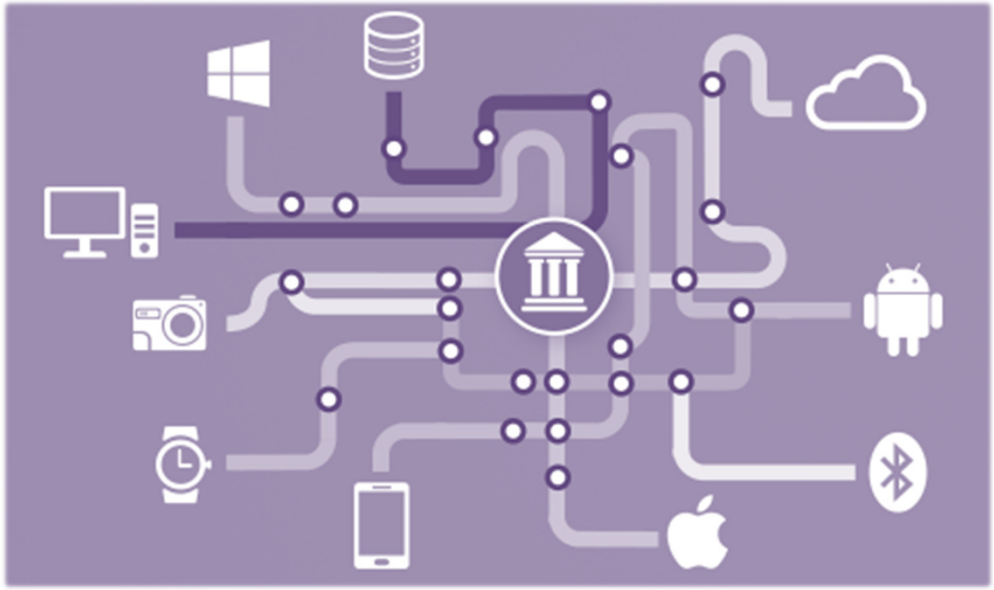
アジェンダ

- DEMONSTRATION ～パラメータを渡す～
- デモンストレーションの説明
- RESTメソッドの定義
- RESTメソッドの実装（Get）
- RESTメソッドの実装（Post）
- クライアントからRESTメソッド(Get)を呼び出す
- クライアントからRESTメソッド(Post)を呼び出す
- 受け渡すパラメータの設定
- コンポーネントの関係とデータの流れ

- EMSサーバーとの「接続」を確認する
- ブラウザからEMSサーバーへアクセスする
- RESTデバッガを使ってみる
- JSONテキストをインターセプトする
- XE7とXE8でEMSサーバーを共有させる
- XE8の新機能
- 動作環境／ライセンス について
- 参考情報／関連情報 (Docwiki)

FireDAC入門

はじめに



スマートデバイスと「データ連携」するための技術

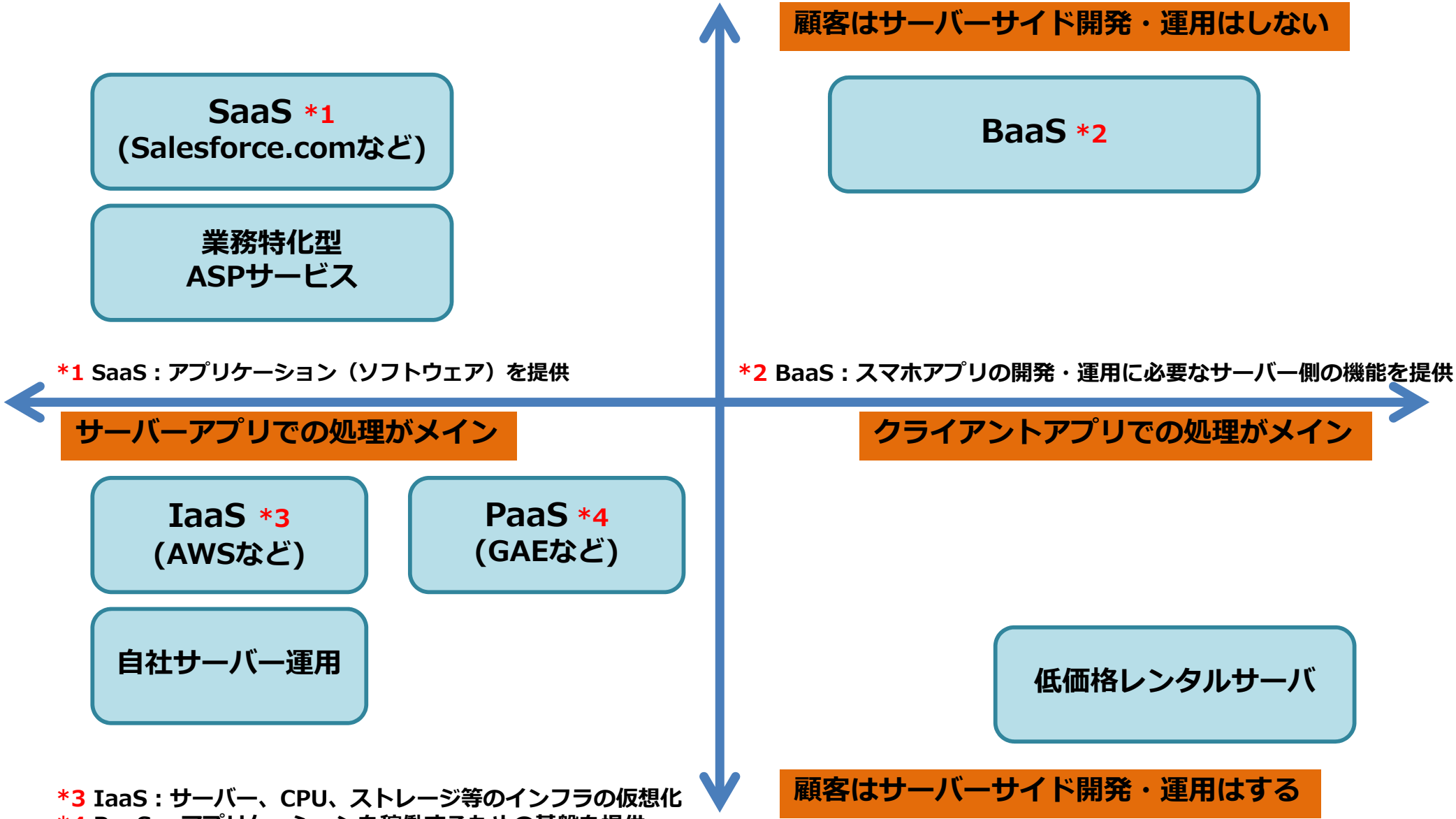
Delphi/
C++Builderで
実装可能

- **アプリケーション テザリング**
  アプリケーション間で、あらゆるアクションデータを共有する
- **DataSnap**
  バックエンドのデータベースをDataSnapでカプセル化する
- **BaaS(Backend as a Service/バース)サービスの活用**
  データストア、プッシュ通信、ユーザー管理、ソーシャル連係、ロケーション連係‥を
  モバイルアプリケーションからAPIで呼び出すことが可能
- **EMS(Enterprise Mobility Services)**
  データアクセスモジュールやカスタムAPIをホストできるミドルウェアサーバー機能

オープンかつ業界標準の技術がベース

REST　HTTP/S　JSON　…

技術のポジショニング



*3 IaaS：サーバー、CPU、ストレージ等のインフラの仮想化
*4 PaaS ：アプリケーションを稼働するための基盤を提供

インーメモリデータベース

◆ 定義

データを全て「メインメモリ上に格納する方式」で構築されたデータベース
（オンーメモリデータベースと同義）

◆ どんなアプリでインーメモリデータベースを使うのか‥

・データを次々に蓄積して、その中からデータを抽出するタイプのアプリケーション

・データを永続的に保持しておくタイプのアプリケーション

・大量のデータから任意のデータを探すタイプのアプリケーション

JSONってなにもの?

- JavaScript Object Notation の略
- XML等と同様、テキストベースのデータフォーマット
- XMLと比べると簡潔に構造化されたデータを記述することが可能
- 記述が容易で人間が理解しやすいデータフォーマット
- XMLには閉じタグが必要。JSONはカッコに対応する閉じカッコ以外は不要
- きちんとインデントされていれば可読性も高い

リスト1:XMLの例

```
<employees>
  <employee>
    <empId>000001</empId>
    <department>企画部</department>
    <name>山田 太郎</name>
  </employee>
  <employee>
    <empId>000002</empId>
    <department>営業部</department>
    <name>山田 次郎</name>
  </employee>
</employees>
```

リスト2:JSONの例

```
[
  {
    "empId": "000001",
    "department": "企画部",
    "name": "山田 太郎",
  },
  {
    "empId": "000002",
    "department": "営業部",
    "name": "山田 次郎",
  }
]
```

FireDAC入門

FireDACのおさらい

FireDACとは?

Delphi XE4 でAnyDACの「Embacadero版」として搭載

ユニバーサルデータアクセスが可能な、一体化されたコンポーネント群

・Oracle/MS-SQL Server/DB2/MySQL/PostgreSQL/InterBase/Firebird/SQLite/SQL Anywhere/Advantage DB/ Access/Informix/DataSnapなどに高速なネイティブアクセスが可能
・対応ドライバは、次のURLを参照
http://docwiki.embarcadero.com/RADStudio/XE8/ja/FireDAC

DelphiおよびC++Builder向け

・RAD Studio/ Delphi/ C++Builder XE4～XE8
・詳しくは、次のURLを参照
http://www.embarcadero.com/jp/products/rad-studio/firedac-faq

マルチデバイス

・Win32, Win64, Mac OS X, iOS, Android

ハイパフォーマンス

・BDEと同等、それ以上のデータアクセススピード

共通化されたAPI

・マクロ機能を使ってSQLの方言や微妙な違いを吸収
・修正可能なデータマッピング機能によるデータ型の統一化

FireDACを使用するメリットは‥

BDEとの互換性が高い

・データアクセスアーキテクチャが類似している
・従来のBDE DataSetとの互換性を備えたDataSetクラスがある
・CachedUpdatesモードも利用可能

ハイパフォーマンス

・BDEと同等、それ以上のデータアクセススピード

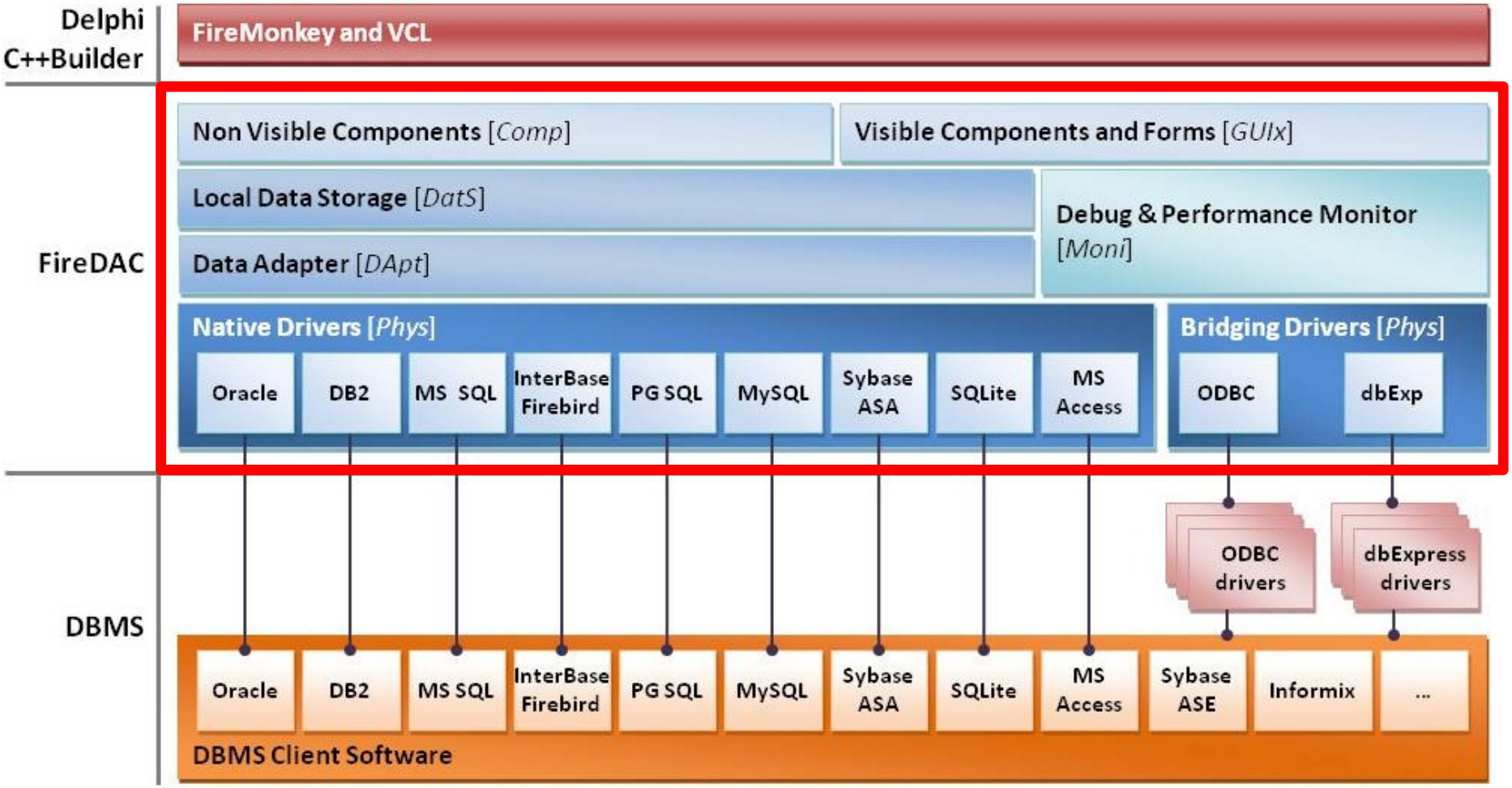
配布が簡単

・専用のデータベースドライバやインストーラが不要

FireDACアプリケーションの構造

・FireDACアプリケーションは、3層構造
・赤枠がFireDACコンポーネント本体
・開発には32ビットのクライアントライブラリが必須

注意！



Delphi XE5 での変更点

- **コンポーネント名が「TADxxx」から「TFDxxx」へ変更**
  - 「**reFind**」という名称変更ツールが提供されている。
  - フォームファイル(*.dfm)のAdpterが「 Fdpter 」と誤変換されるので注意が必要

- **InterBaseとFirebirdのDriverIDが次の通り変更（XE4では両方「IB」だった・・）**
  - InterBase → IB
  - Firebird → **FB**

- **usesのユニット名が次の通り変更**
  - uAD<レイヤ><ロール>.pas → **FireDAC.<レイヤ>.<ロール>.pas**
    ex. uADCompClient が「FireDAC.Comp.Client」に変更

- **次の3つのプログラムは、IDEのメニューから直接呼び出せない** *1
  - **FireDAC エクスプローラ** (FDExplorer.exe)
  - **FireDAC モニタ** (FDMonitor.exe)
  - **REST デバッガ** (RESTDebugger.exe)

- **詳細はFireDACヘルプ(DocWiki)を参照** *2

*1 エンバカデロの手違いでXE5では、メニューに登録されていません。プログラムの単独起動となる
詳しくは、こちらをご覧下さい。(http://blog.marcocantu.com/blog/new_tools_xe5_missing_menu.html)
*2 「FireDACへの移行」を参照
(http://docwiki.embarcadero.com/RADStudio/XE8/ja/FireDAC_%E3%81%B8%E3%81%AE%E7%A7%BB%E8%A1%8C)

Delphi XE7 での変更点

- **クラス名、ユニット名等のリファクタリング**
  旧バージョンのプロジェクトを移行する場合は注意が必要

- **JSON シリアル化をサポート**
  Bin/XML/JSONの形式ごとにTFDStanStorage**XXX**Link*1 を追加

- **データエクスプローラで FireDAC が利用可能**

- **TFDBatchMove の追加（TFDDataMove は非推奨）**

- **MSSQL FILESTREAMのサポート**

- **DBMSネイティブコマンドタイムアウト(ODBC API)のサポート**
  利用可能なDB: Advantage/Informix/SQL Server

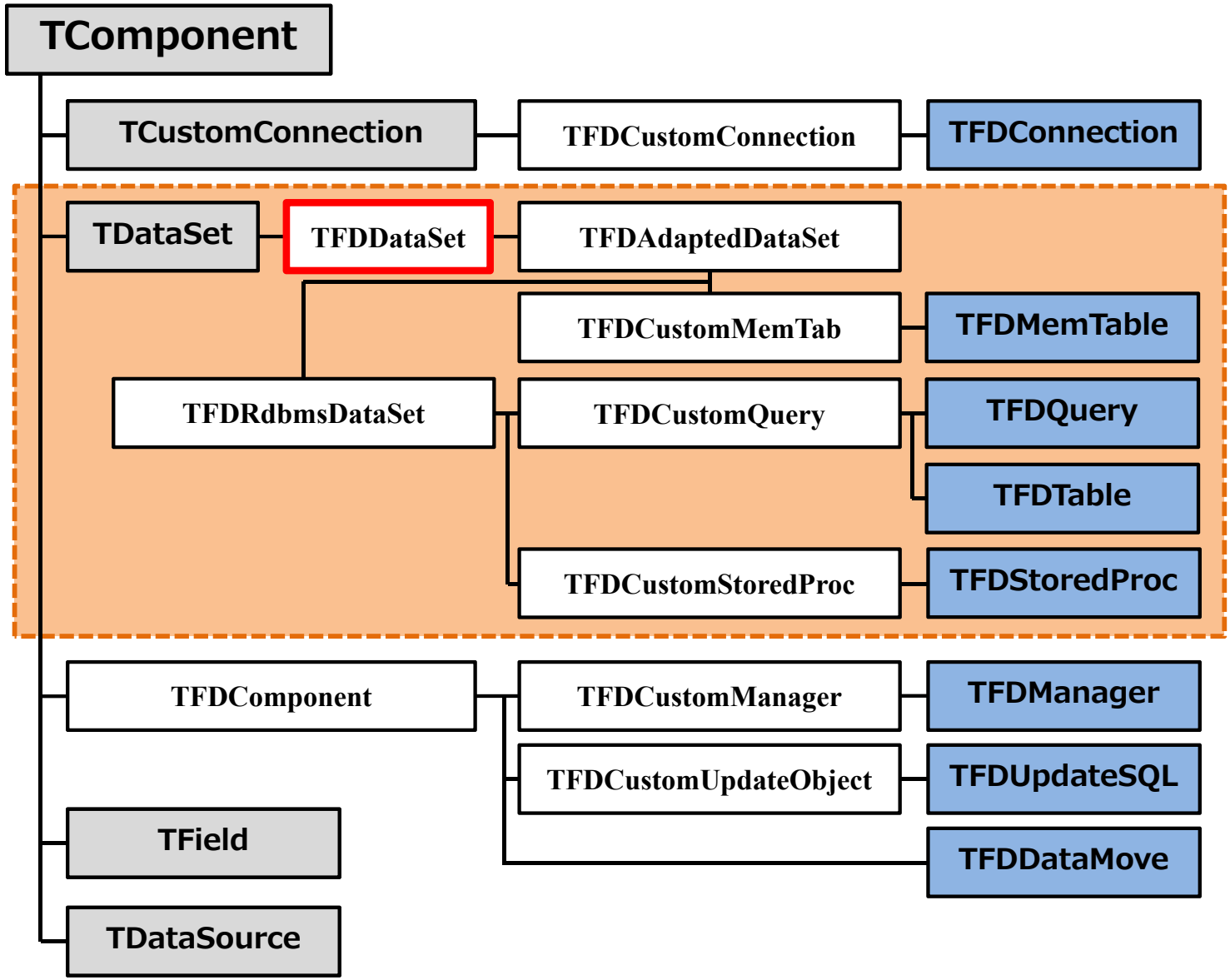
*1 TFDStanStorage**Bin**Link: バイナリ形式／ TFDStanStorage**XML**Link: XML形式／ TFDStanStorage**JSON**Link: JSON形式

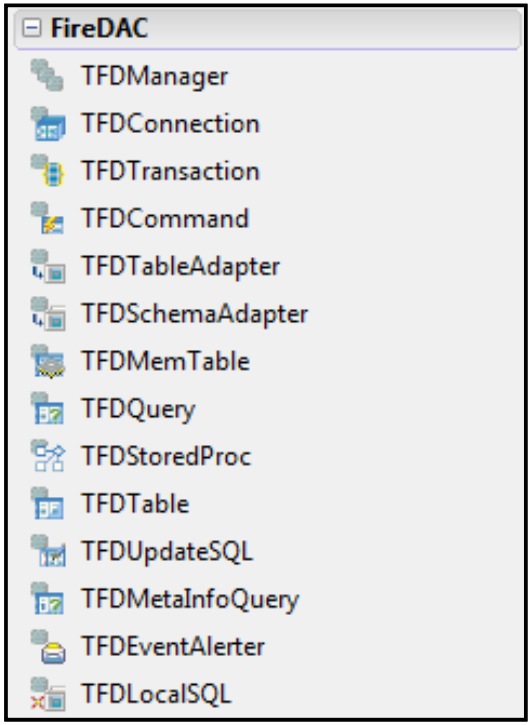
BDEとFireDAC 主要コンポーネントの比較

BDE	FireDAC	Unit
TDatabase	**TFDConnection**	FireDAC.Comp.Client
TSession	**TFDManager**	FireDAC.Comp.Client
TTable	**TFDTable**	FireDAC.Comp.Client
TQuery	**TFDQuery**	FireDAC.Comp.Client
TStoredProc	**TFDStoredProc**	FireDAC.Comp.Client
TUpdateSQL	**TFDUpdateSQL**	FireDAC.Comp.Client
TBatchMove	**TFDDataMove**	FireDAC.Comp.DataMove
–	**TFDPhysXXXXDriverLink**	FireDAC.Phys.XXXX
–	**TFDGUIxWaitCursor**	FireDAC.Comp.UI

※ BDEのコンポーネント名に**FireDAC**の「**FD**」を付加したネーミング
※ XXXX：ドライバーによって異なる

データベースクラスの継承関係





XXX ：FireDAC

XXX ：Delphi標準

XXX ：TDataSetから派生

FireDAC入門

FDMemTable

テーブルを生成する

```
 private
   CustomersTbl: TFDMemTable;

CustomersTbl := DataModule1.NewCustomers;
```

```
function TDataModule1.NewCustomers: TFDMemTable;
begin
  Result := TFDMemTable.Create(nil);
  with Result do
  begin
    // 項目定義                                          *1   *2
    FieldDefs.Add('CUSTOMERID', ftInteger, 0, False);
    FieldDefs.Add('COMPANYNAME', ftString, 40, False);
    FieldDefs.Add('ADDRESS', ftString, 120, False);
    FieldDefs.Add('PHONE', ftString, 15, False);
    FieldDefs.Add('FAX', ftString, 15, False);
    // インデックス定義
    IndexDefs.Add('Customers_IDX','CUSTOMERID', [ixPrimary]);
    // 空の内部データ記憶域を作成
    CreateDataSet;
    // ログ記録をoff
    LogChanges := False;
  end;
end;
```

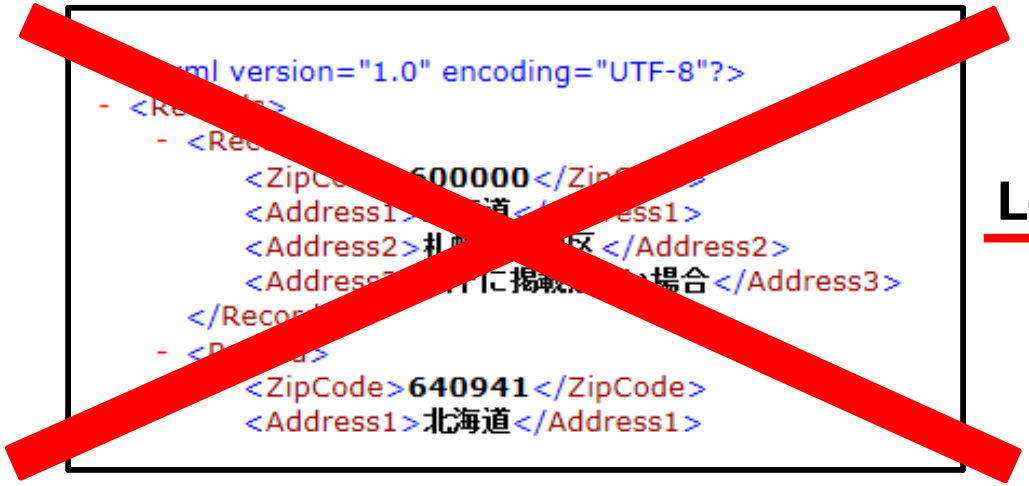
[次の様に書き換えることもできる]

```
FieldDefs.Add('CUSTOMERID', ftInteger);
FieldDefs.Add('COMPANYNAME', ftString, 40);
FieldDefs.Add('ADDRESS', ftString, 120);
FieldDefs.Add('PHONE', ftString, 15);
FieldDefs.Add('FAX', ftString, 15);
```

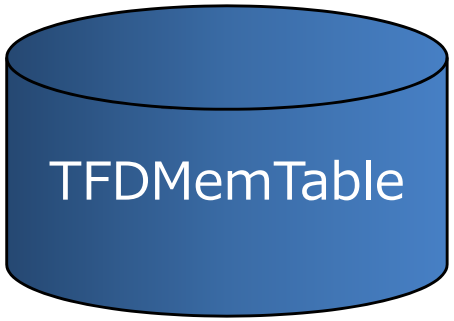
*1 Sizeプロパティは，**ftString/ftBCD/ftBytes/ftVarBytes/ftBlob/ftMemo/ftGraphic** のときのみ有効
*2 RequiredプロパティがFalseならNullを許可する（省略可能）

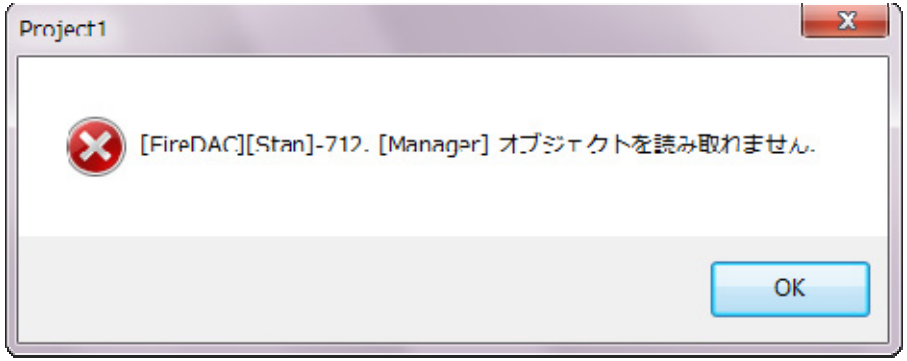
一般的な「Xml ファイル」は読込めない！

```
procedure TDataModule1.SaveToTable(AFileName: String);
begin
  FDMemTable1.LoadFromFile(AFileName, sfXML);
end;
```



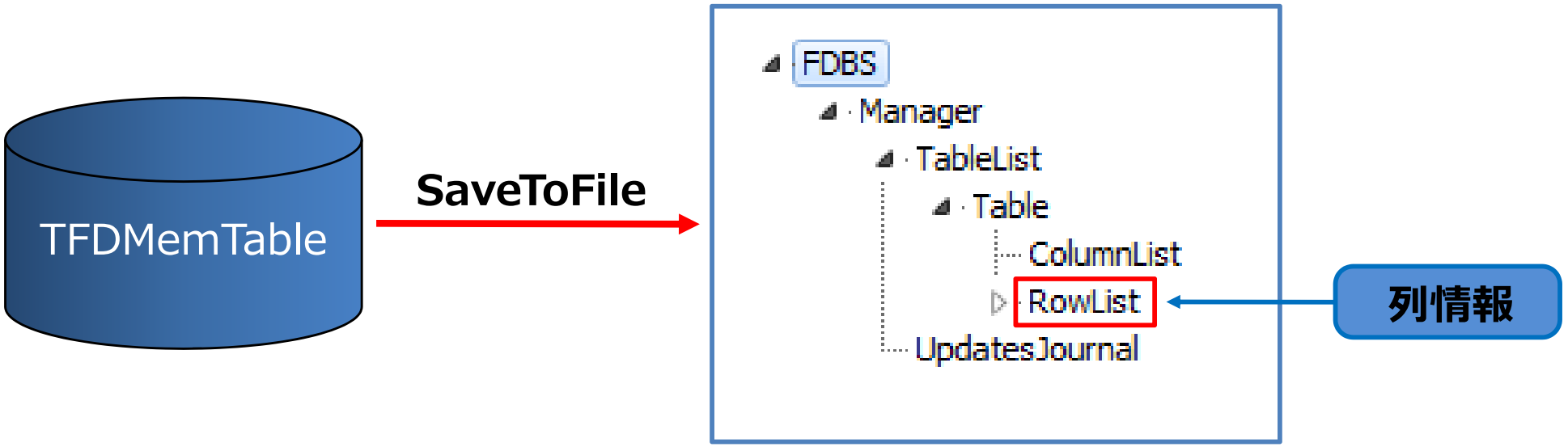
LoadFromFile





出力した「Xml ファイル」の構造は‥

```
procedure TDataModule1.SaveToTable(AFileName: String);
begin
  FDMemTable1.SaveToFile(AFileName, sfXML);
end;
```



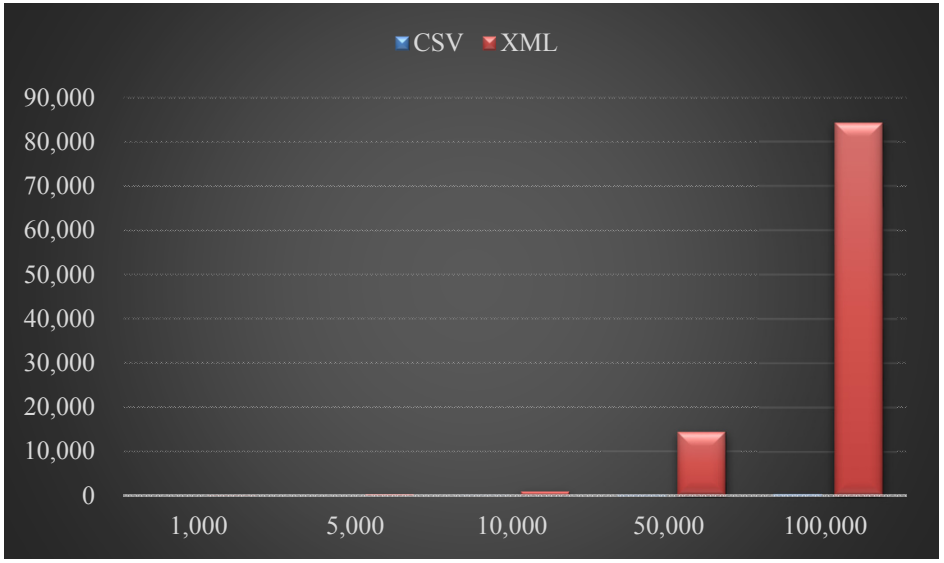
自力で「Xml ファイル」を 読み／書き してみると‥

```
<?xml version="1.0" encoding="UTF-8"?>
- <Records>
  - <Record>
      <ZipCode>600000</ZipCode>
      <Address1>北海道</Address1>
      <Address2>札幌市中央区</Address2>
      <Address3>以下に掲載がない場合</Address3>
    </Record>
  - <Record>
      <ZipCode>640941</ZipCode>
      <Address1>北海道</Address1>
```

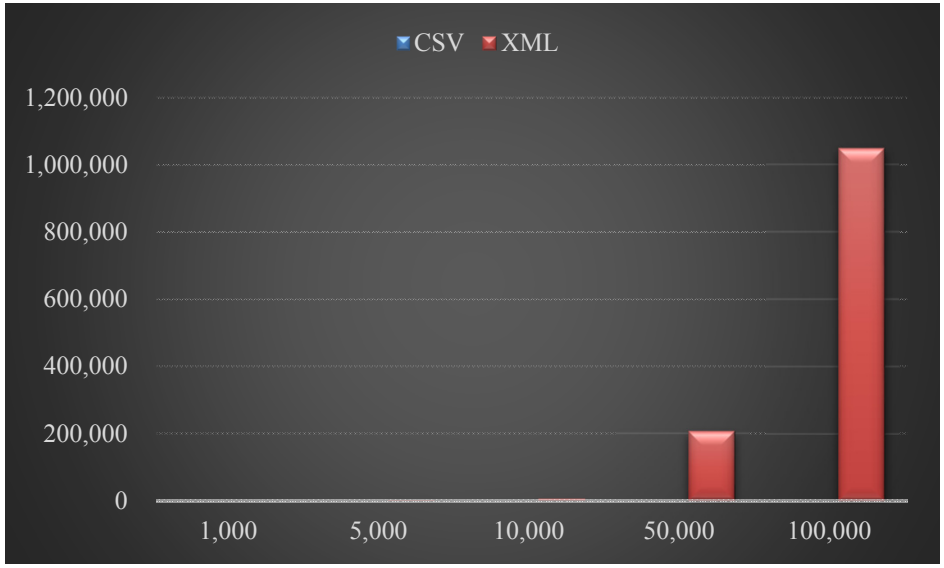
Load/Save

TFDMemTable

Load

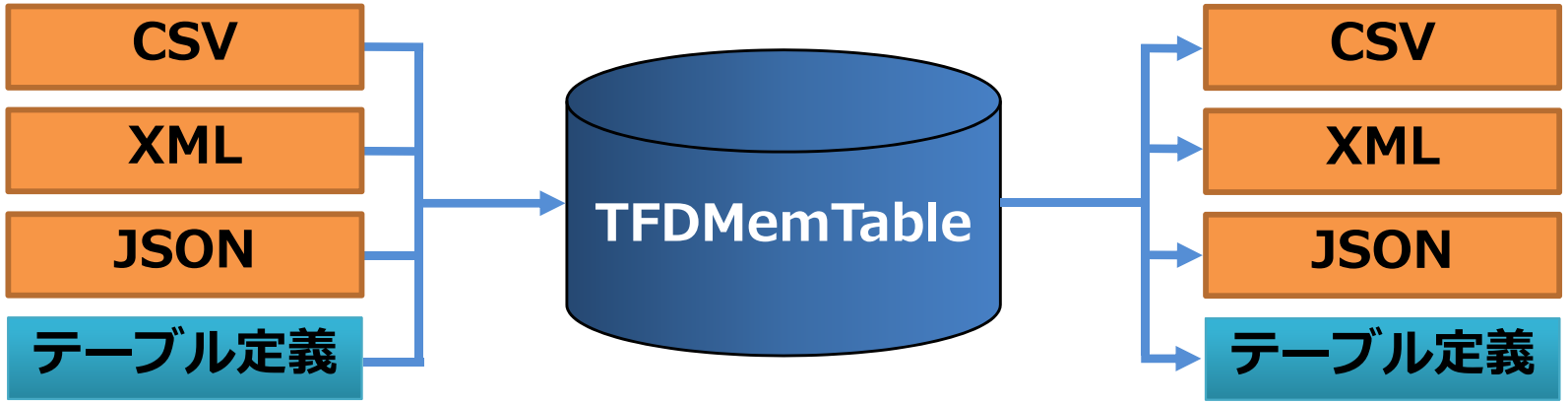


Save

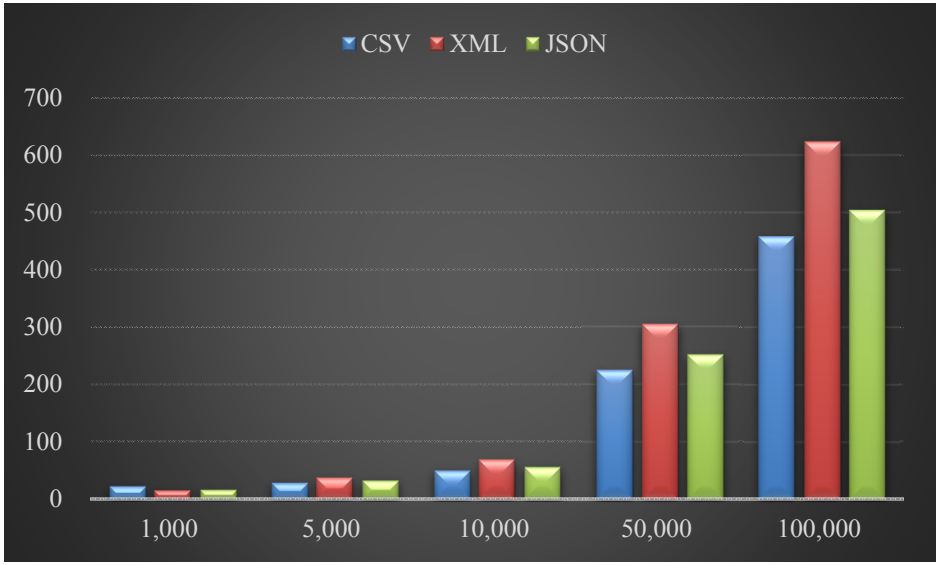


※ 日本郵便の「郵便番号データ」を使用し、5回の実測値の平均をグラフ化
※ グラフの横軸は件数／グラフの縦軸は処理時間（単位：ミリ秒(ms)）

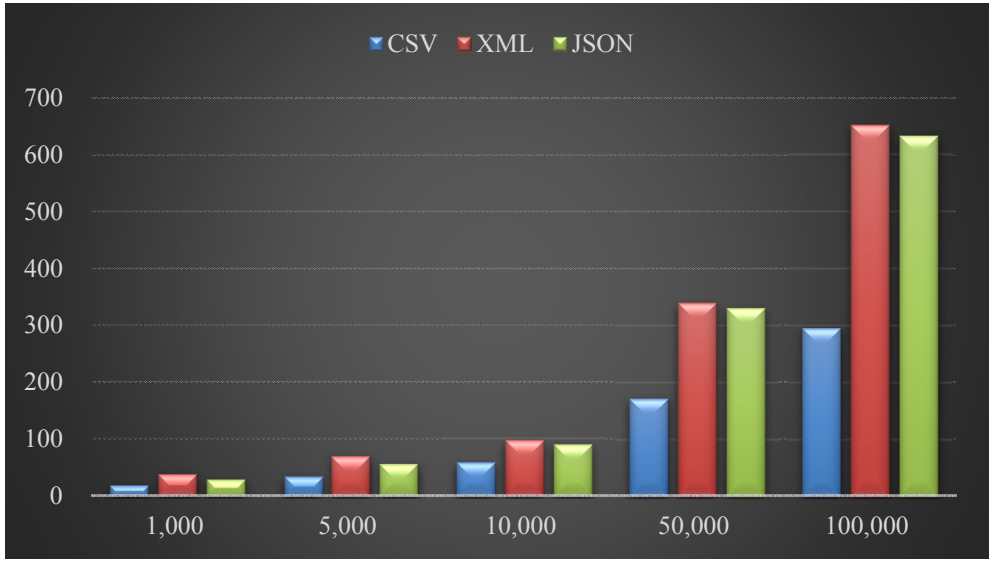
拙作クラス (TDataMigration) で読み／書きしてみると‥



Load



Save



※ 日本郵便の「郵便番号データ」を使用し、5回の実測値の平均をグラフ化
※ グラフの横軸は件数／グラフの縦軸は処理時間（単位：ミリ秒(ms)）

別のデータセットから、全てのレコードをコピーする

Dataプロパティを使用

```
procedure TDataModule1.CopyRecord;
begin
  FDQuery1.FetchOptions.Unidirectional:= False;
  FDQuery1.Open;
  FDQuery1.FetchAll;
  FDMemTable1.Data := FDQuery1.Data;
  FDMemTable1.First;
end;
```

CopyDataSetメソッドを使用

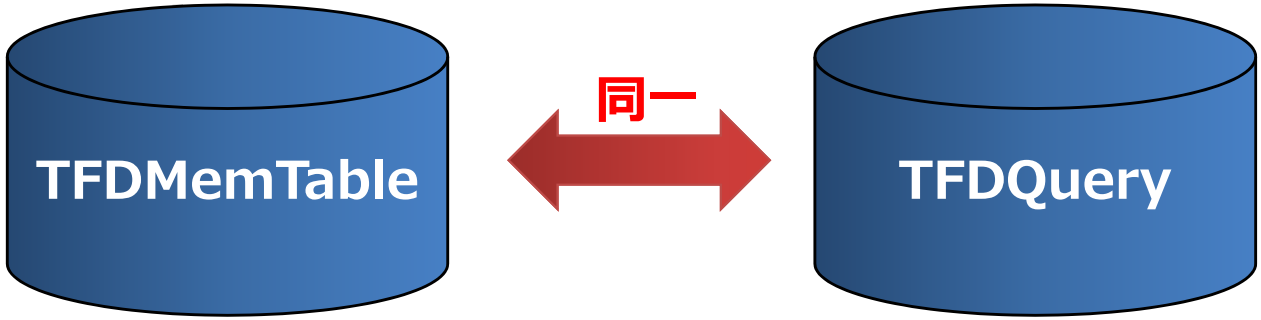
```
procedure TDataModule1.CopyTable;
begin
  FDMemTable1.CopyDataSet(FDQuery1, [coStructure, coRestart, coAppend]);
end;
```

比較項目	Dataプロパティ	CopyDataSetメソッド
FireDAC以外のデータセット	×	○
使用クラス	DatS(ローカルデータストレージ)	TDataSet
コピー対象	レコードすべてのバージョン(履歴含む)	現在のフィールド値
速度	高速	低速

別のデータセットのデータを共有する

CloneCursorメソッドを使ってクローンカーソルを作成する。
2つのデータセットの内部データ記憶域は、物理的に同一となる。

```
procedure TDataModule1.CopyTable;
begin
  FDMemTable1.CloneCursor(FDQuery1, False, False);
end;
```

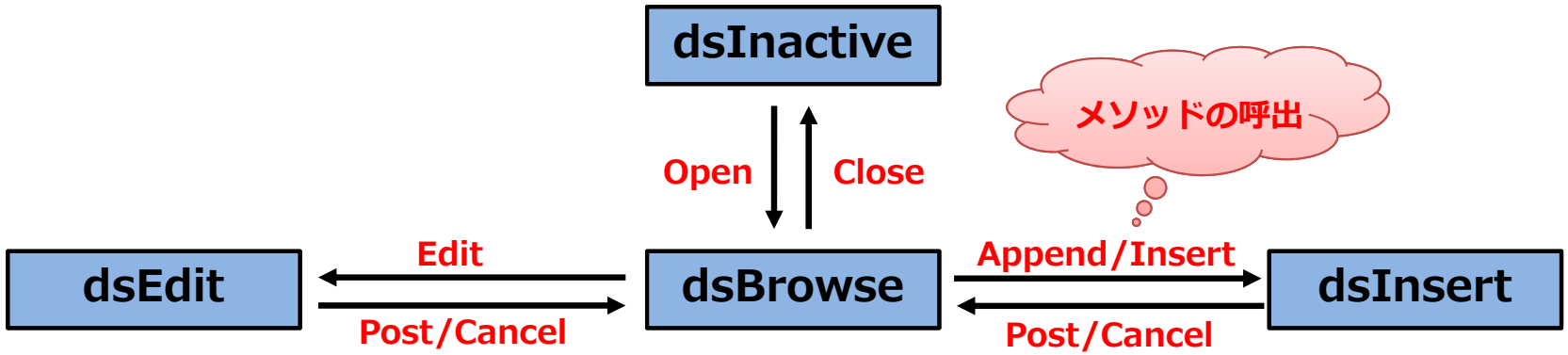


データセットの状態を調べる

TFDMemTableの「Stateプロパティ」を調べるとデータセットの状態を知ることができる

Stateプロパティ	状　態	説　明
dsInactive	非アクティブ状態	データセットが開かれていない
dsBrowse	参照状態	データセットは参照モードにある
dsEdit	編集状態	データセットは修正モードにある
dsInsert	挿入状態	データセットは挿入モードにある

Stateプロパティの変化



取得するデータを絞り込む

TFDMemTableの「Filterプロパティ」・「Filteredプロパティ」を使ってデータを絞り込むことができる

演算子	Filterプロパティの設定例
比較演算子	Address1 = '長野県'
論理演算子	Address1 = '長野県' OR Address1='新潟県'
LIKE演算子	Address1 LIKE '長%'
IN演算子	OrderID IN (10150, 10151, 10152)
NULL判定	Address1 is NULL

```
procedure TForm1.btnSearchClick(Sender: TObject);
begin
  with FDMemTable1 do
  begin
    Filtered := False;
    // フィルタのセット
    Filter   := 'Address1=' + QuotedStr('長野県');
    Filtered := True;
  end;
end;
```

TFDMemTable のまとめ

イン－メモリー、またはデータベースを切断しての使用に使われる。

主な機能

- 別のデータセットからコピー
  Dataプロパティ、CopyDataSetメソッドを使用
- ファイルとストリームによるデータの永続性
  ファイルの形式は XML(sfXML)、バイナリ(sfBinary)、JSON(sfJSON)
  メソッドは LoadFromFile、SaveToFile、SaveToStream、LoadFromStream
- キャッシュアップデート
- クローン化されたカーソル
  CloneCursorメソッド
- ネストされたデータセット

FireDAC入門

SQLite

接続の設定

FireDAC 接続エディタで設定

オブジェクトインスペクタの下にある**「接続エディタ ...」**をクリックする

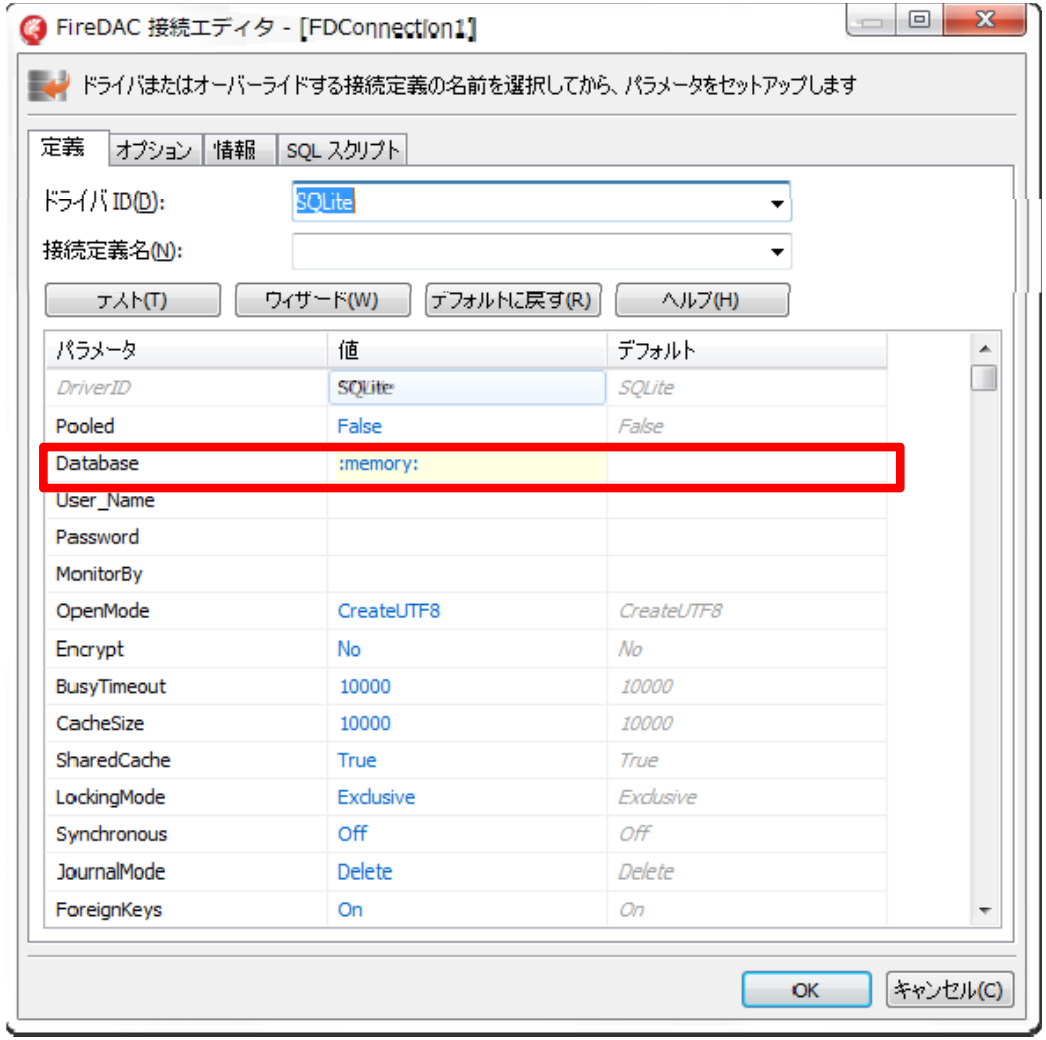
- **ドライバ ID:**
  リストから接続するサーバーのIDを指定
  選択すると既定値が、パラメータ欄に表示される

- **接続定義名:（必須ではない）**
  エリアス名をリストから選択
  エリアスは「FireDAC エクスプローラ」で設定

- **Database:**
  イン－メモリーデータベースを使用するので
  「**:memory:**」を指定

ここがポイント！

※他のパラメータは、適宜設定します
※入力後、**「テスト」**ボタンを押して接続を確認



接続の設定（続き）

コードで設定

実行時に接続に必要なパラメータをコードで指定する

```
try
  with FDConnection1 do
  begin
    // 接続パラメータのセット
    Params.Clear;
    Params.Add('DriverID=SQLite');
    Params.Add('Database=:memory:');
    LoginPrompt := False;
    // 接続
    Open;
  end;
except
  MessageDlg('データベースに接続できません！', mtError, [mbOK], 0);
end;
```

※インーメモリーデータベースの場合、Databaseパラメータに「**:memory:**」を指定する。
　従来型の「オンディスクデータベース」の場合は、データベースのフォルダを指定する。

テーブルを生成する

SQL文を発行するメソッドには２種類ある
結果セットを返す場合と、返さない場合とでは使用するメソッドが異なる

メソッド	機　能
Open	SELECT文を使ってデータセットに問い合わせを行い、その結果を受け取る
ExecSQL	DDL/DML文などのデータセットに対して操作を行い、 その結果を返さないSQL文を実行する

```
const
  strSQL = 'CREATE TABLE personal(id integer, name text(20))';

try
  with FDQuery1 do
  begin
    // SQL文をセット
    SQL.Clear;
    SQL.Add(strSQL);
    // SQLの実行
    ExecSQL;
  end;
except
  MessageDlg('テーブルを作成できません！', mtError, [mbOK], 0);
end;
```

高速にデータを挿入する

- FireDACの「DML配列機能」を使う
- 複数のDML(INSERT/UPDATE/DELETE)文をパラメータ付きで一度に実行できる
- 高速実行の実現
  通常のExecSQL → 5.5秒
  DML配列 → **0.03秒**

```
const
  strSQL = 'INSERT INTO personal (id, name) VALUES(:f1, :f2)';
var
  i: Integer;

with FDQuery1 do
begin
  // SQL文のセット
  SQL.Clear;
  SQL.Add(strSQL);
  // 配列の大きさをセット
  Params.ArraySize := 1000;
  // 配列に値をセット
  for i:=0 to Params.ArraySize do
  begin
    ParamByName('f1').Values[i-1] := i;
    ParamByName('f2').Values[i-1] := 'Str' + IntToStr(i);
  end;
  // SQLの実行
  Execute(FDQuery1.Params.ArraySize, 0);
end;
```

動的問い合わせ

実行時にパラメータを指定してSQLを実行する

SQL文にSQLパラメータ*1 を埋め込むことで、変数のように利用できる

プロパティ／メソッド	機　能
Params[n]	変数配列n番目に対して値をセットする
ParamByName(Value)	引数Valueに指定された変数名に対して値をセットする

```
procedure TForm2.btnExeSelectClick(Sender: TObject);
const
  strSQL = 'SELECT * FROM personal WHERE (id >= :startcode) AND (id <= :endcode)';
begin
  with FDQuery1 do
  begin
    // SQL文のセット
    SQL.Clear;
    SQL.Add(strSQL);
    // 変数に値をセットする
    ParamByName('startcode').AsInteger := 501;
    ParamByName('endcode').AsInteger   := 1000;
    // リソースの割り当てと最適化を実行する
    Prepare;
    // SQLの実行
    Open;
  end;
end;
```

[次の様に書き換えることもできる]

```
Params[0].AsInteger := 501;
Params[1].AsInteger := 1000;
```

*1 **SQLパラメータ**：バインド変数・ホスト変数とも呼ばれる。パラメータ名の前にコロン(:)を付けて記述します

データベースをバックアップする

SQLiteのバックアップは、次の2つのコンポーネントを使って行う。

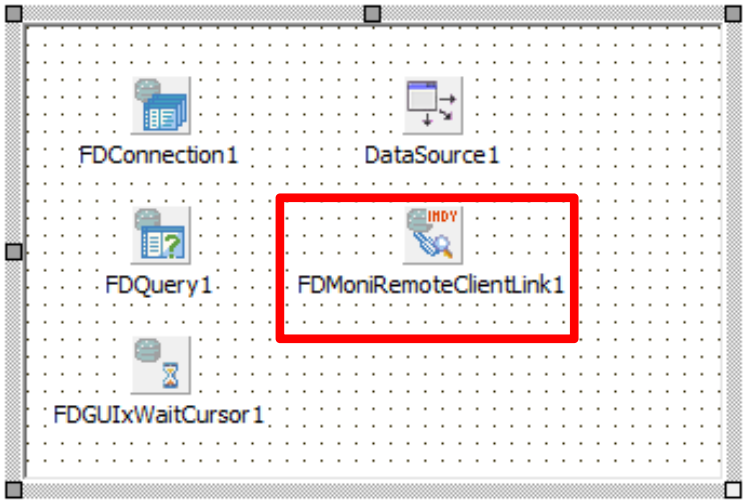
- TFDPhysSQLiteDriverLink *1
- TFDSQLiteBackup

```
procedure TForm1.btnBackupClick(Sender: TObject);
begin
  // sqlite3.dllのパスを指定
  FDPhysSQLiteDriverLink1.VendorLib := 'C:¥pg¥sqlite¥sqlite3.dll';
  FDSQLiteBackup1.DriverLink := FDPhysSQLiteDriverLink1;
  // データベースの指定
  FDSQLiteBackup1.DatabaseObj := FDConnection1.CliObj;
  // ファイル名の指定
  FDSQLiteBackup1.DestDatabase := 'C:¥Temp¥SQLite.backup';
  // バックアップ処理の実行
  FDSQLiteBackup1.Backup;
end;
```

*1 **TFDPhysSQLiteDriverLink は、**イン－メモリーデータベースのみの時は不要です。

SQLをトレースする

・プロジェクトにデータモジュールを追加し、コンポーネントを配置
・TFDMoniRemoteClientLinkコンポーネント(赤枠)を配置



TFDMoniRemoteClientLinkコンポーネントのプロパティを設定

TFDMoniRemoteClientLink	設定内容
HOST *1	「FireDAC Monitor」が起動しているTCP/IPのアドレスを指定 Defaultは127.0.0.1(localhost)
Port	「FireDAC Monitor」でリスニングしているポートを指定 標準ポートは8050

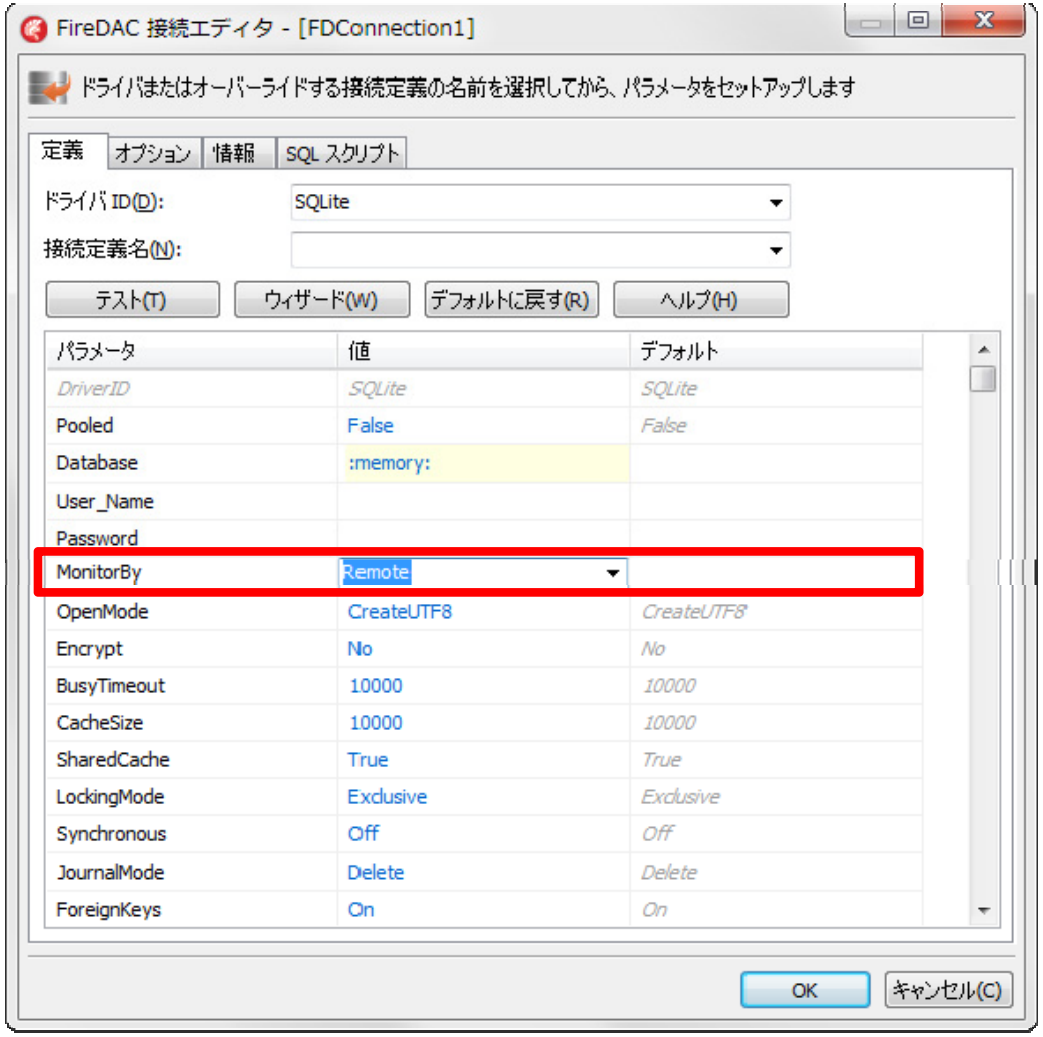
*1 別PCでモニタリングする場合は、そのIPアドレスを指定する

SQLをトレースする（続き）

FireDAC 接続エディタで設定
オブジェクトインスペクタの下にある**「接続エディタ ...」**をクリックする

ここがポイント！

- **MonitorBy**
  リストから「**Remote**」を選択

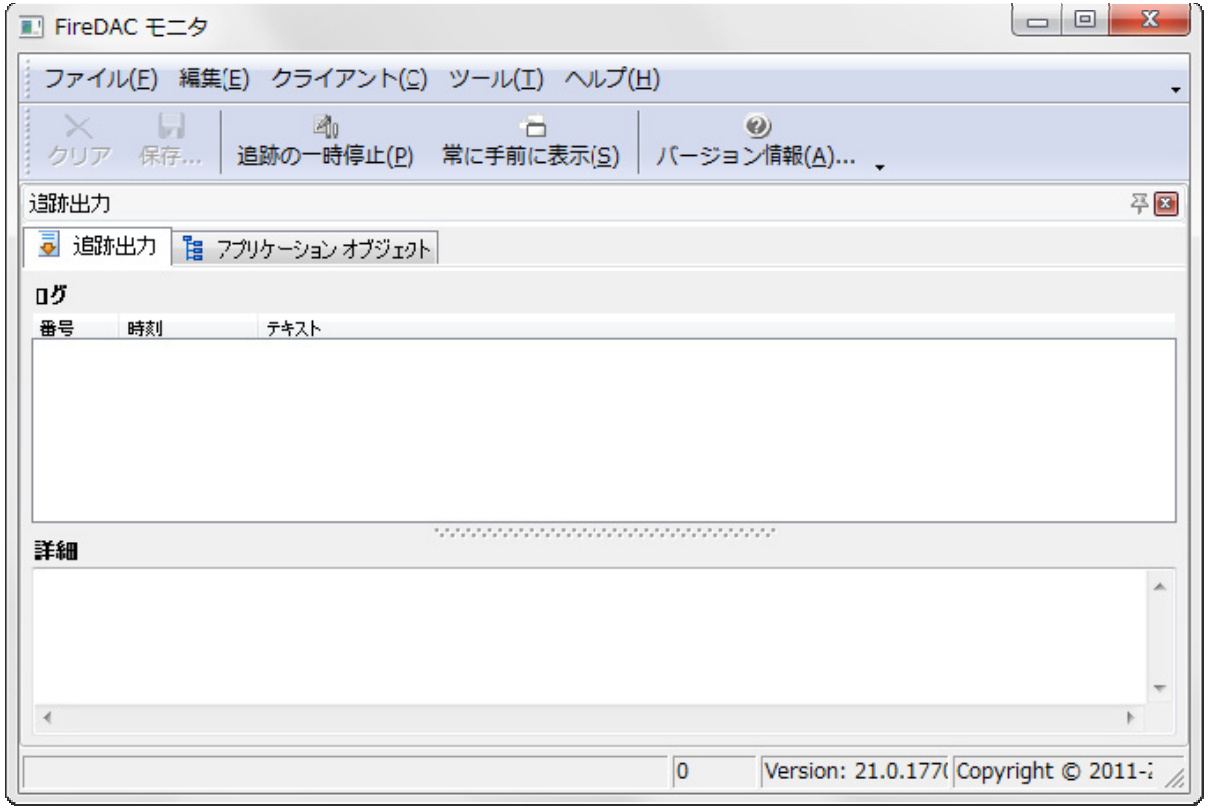


SQLをトレースする（続き）

- Tracingプロパティを設定します

```
FDMoniRemoteClientLink1.Tracing := True;*1
```

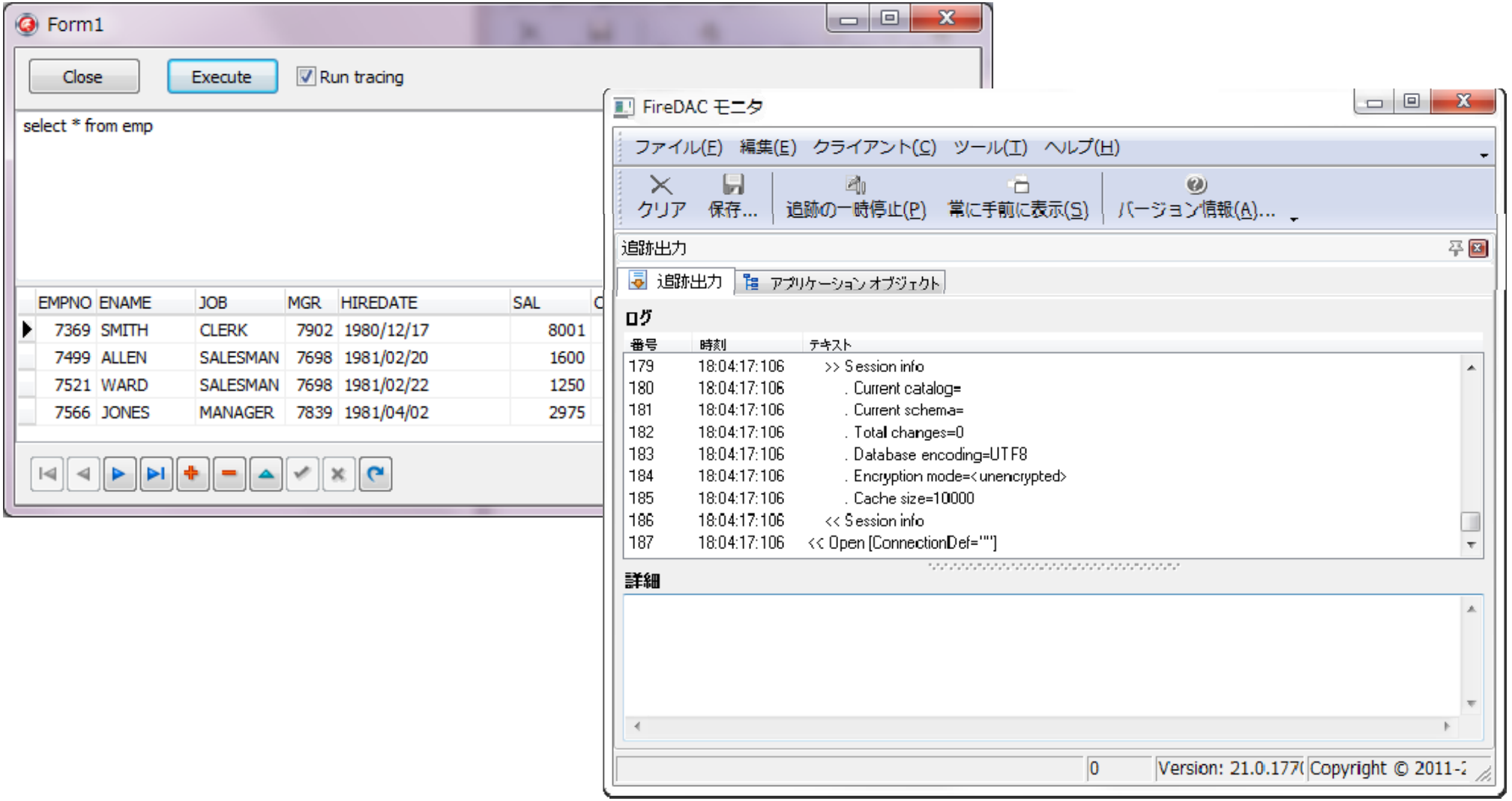
- FireDACモニタ を起動する



*1 True: トレース開始／False: トレース終了

SQLをトレースする（続き）

アプリケーションからSQLを実行し、モニタで結果を確認する

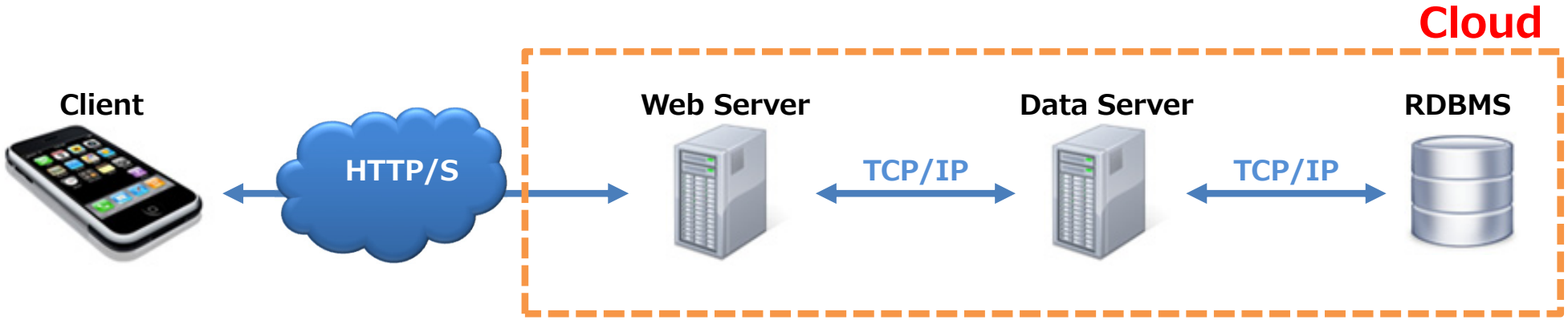


FireDAC入門

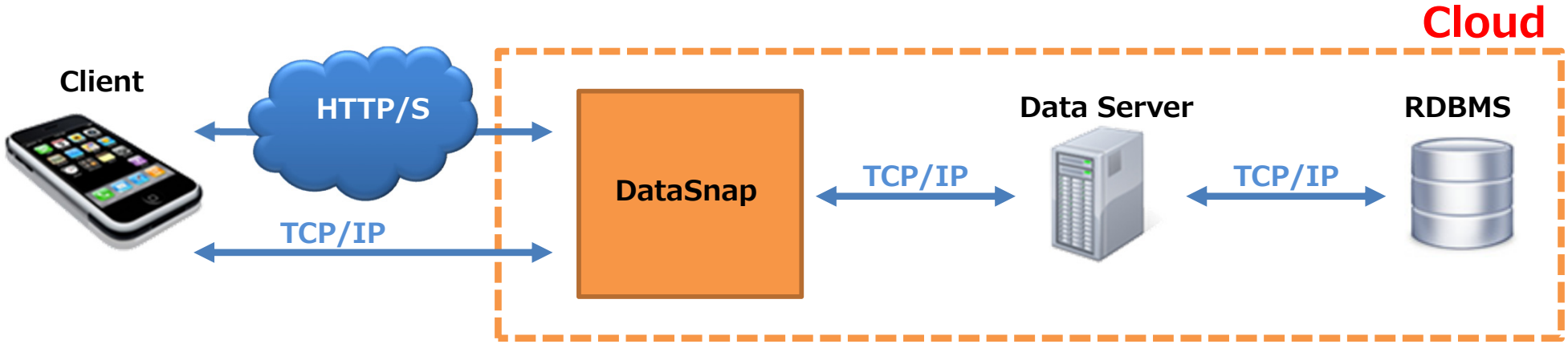
EMSサーバー／クライアント

Webアプリケーション／DataSnapの構成

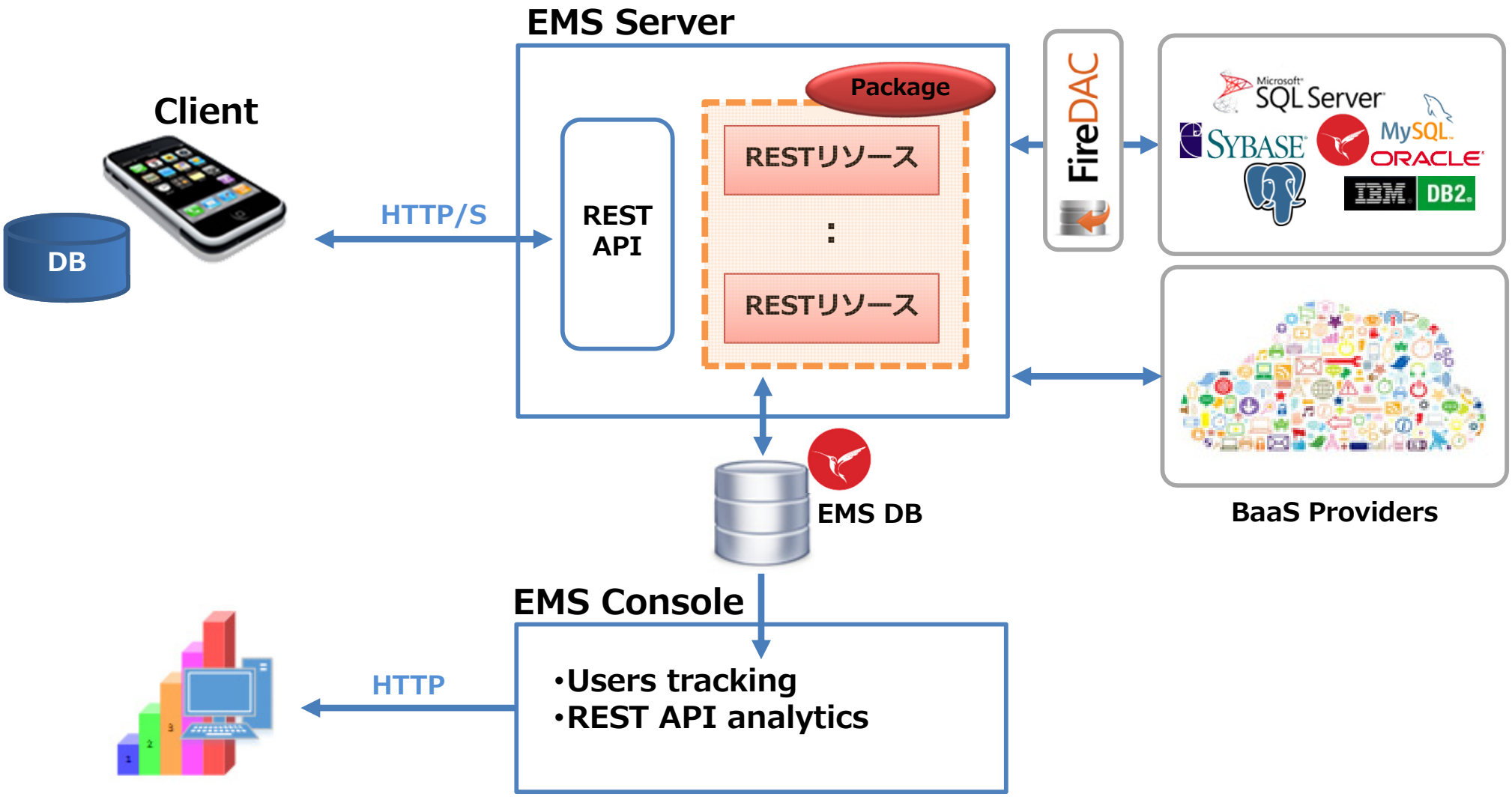
- 一般的なWebアプリケーション



- DataSnap(REST) アプリケーション

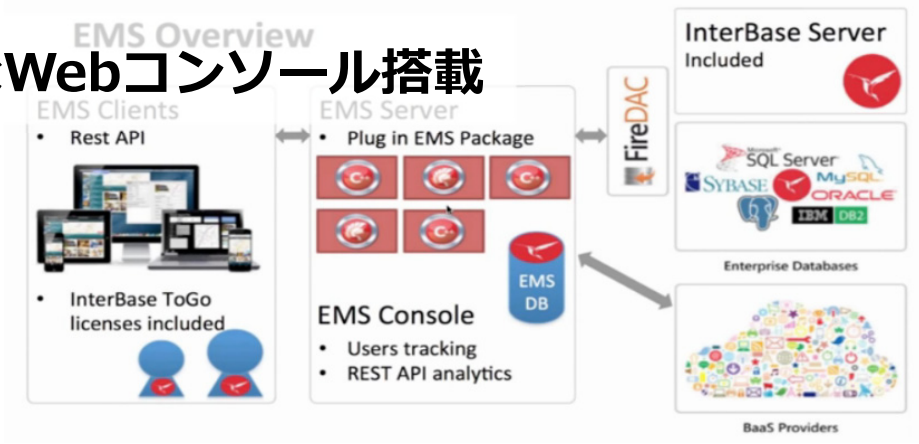


EMSサーバー／クライアントの構成



EMS って、なに‥?

- Enterprise Mobility Services の略
- データアクセス、カスタムAPIを搭載したミドルウェアサーバー
- ユーザー管理／認証機能を搭載
- REST上にロード可能なカスタムAPIを実装可能
- EMSを介して、企業のSQLデータベースへアクセス可能
- モバイル組込やサーバー側のSQLデータ･ストレージ
- ユーザー、デバイス、およびAPIの分析可能なWebコンソール搭載



REST って、なに‥?

- **REpresentational State Transfer の略**
- **HTTP／HTTPS を使ったシンプルな Webサービス**
- **URI でリソースを特定する**
  例えば (http://example.com/users/1234) という感じ
- **HTTPメソッド を使用して操作する**
  GET/POST/PUT/DELETE‥

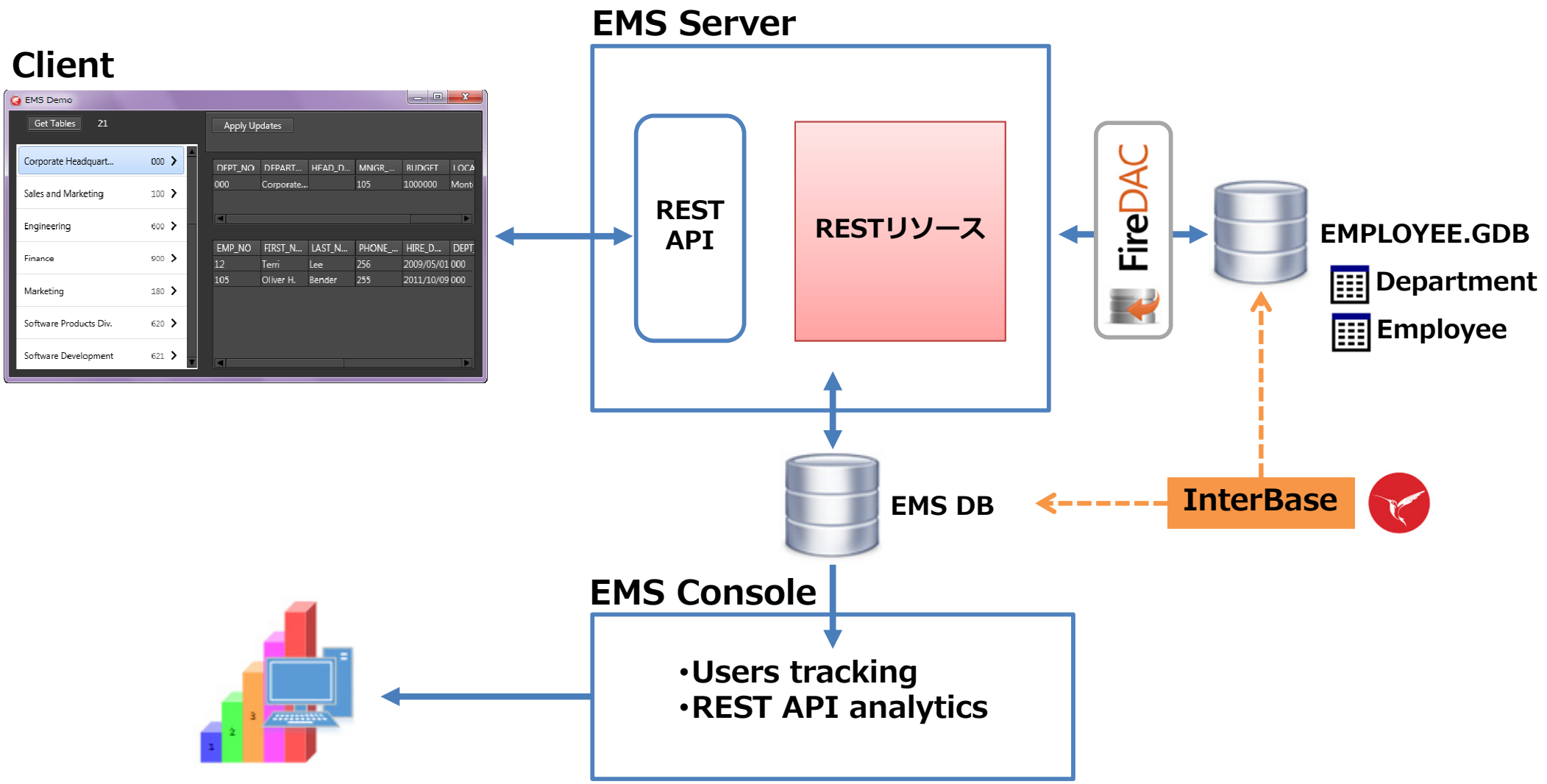
メソッド	役　割
GET	リソースの取得。GETでのアクセスはリソースの内容に影響を与えない
POST	リソースの新規作成
PUT	既存のリソースのアップデート（変更／更新）
DELETE	リソースの除去／削除

- **JSON／XML 等のデータを利用する**
- **多くのベンダーから開発者向けAPIがRESTで提供されている**
  Google/Amazon/Salesforce/JIRA‥

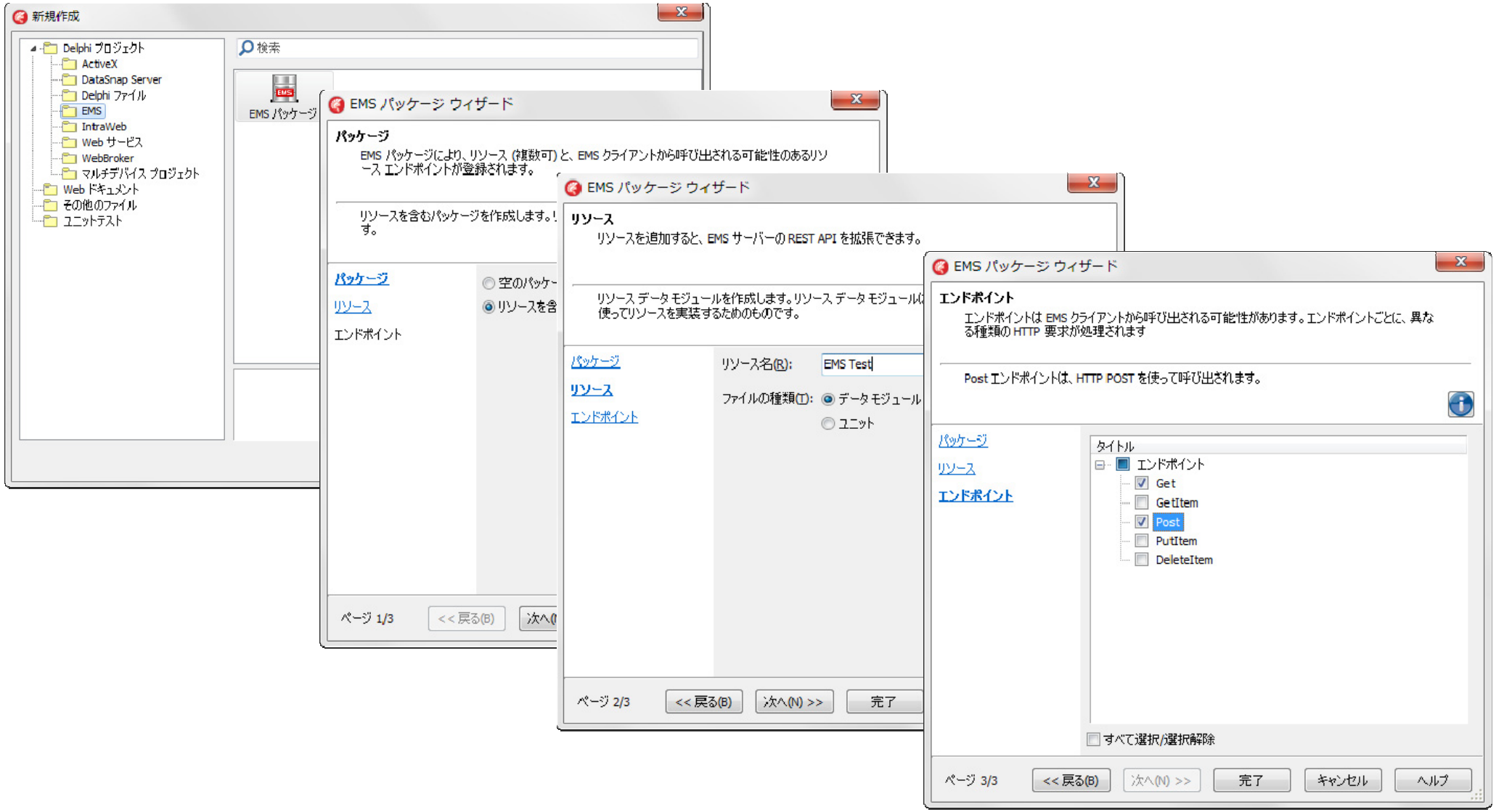
DataSetを渡す

DEMONSTRATION

デモンストレーションの説明



ウィザードを使ってRESTリソースを作成する



ウィザードが作成した RESTリソース の内容は・・

RESTリソースの中身は「パッケージ」です。

```
unit Unit1;

// EMS Resource Module

interface

uses
  System.SysUtils, System.Classes, System.JSON,
  EMS.Services, EMS.ResourceAPI, EMS.ResourceTypes;

type
  [ResourceName('EMS Test')]
  TFireDACEMSTestResource1 = class(TDataModule)*1
  published
    procedure Get(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
      const AResponse: TEndpointResponse);
    procedure Post(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
      const AResponse: TEndpointResponse);
  end;

procedure Register;

implementation

{%CLASSGROUP 'System.Classes.TPersistent'}

{$R *.dfm}
```

リソース名

メソッド(GET/POST)

*1 RESTリソースは、TDataModule を継承している

ウィザードが作成した RESTリソース の内容は・・（続き）

空のメソッドが作成される

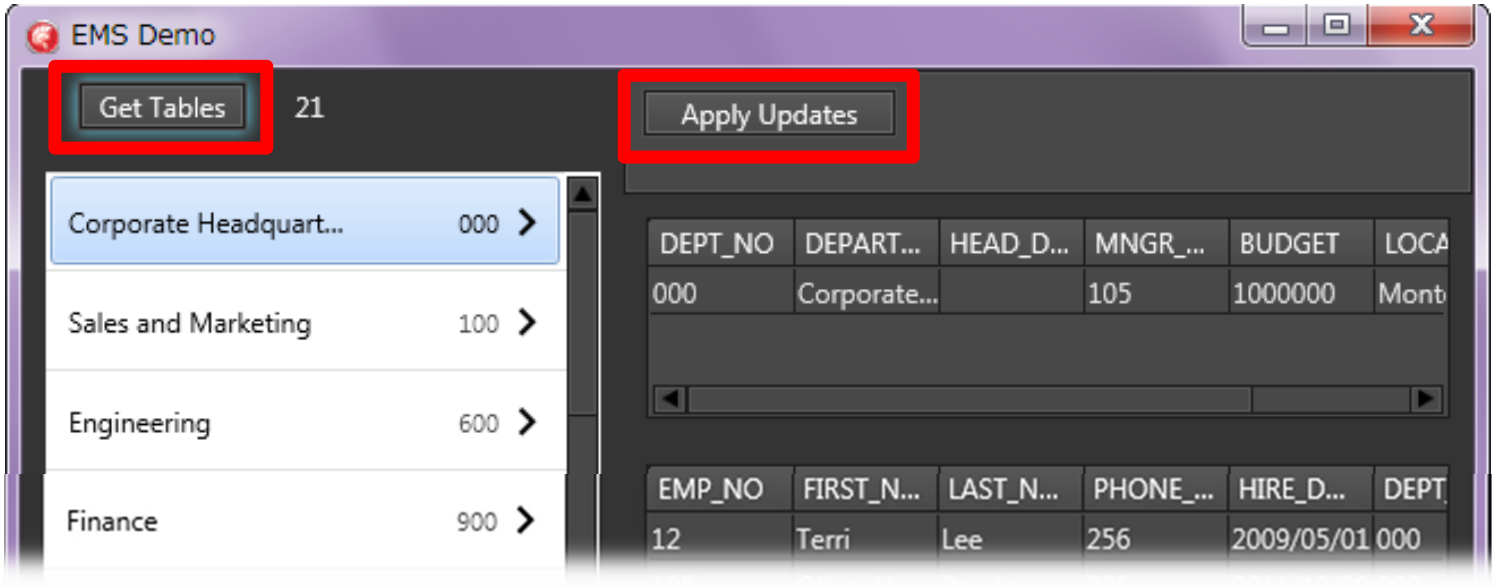
```
procedure TFireDACEMSTestResource1.Get(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
begin
end;

procedure TFireDACEMSTestResource1.Post(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
begin
end;

procedure Register;
begin
  RegisterResource(TypeInfo(TFireDACEMSTestResource1));
end;

end.
```

RESTメソッドの実装（Get／Post）

```
procedure TEMSTestResource1.Get(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
  const AResponse: TEndpointResponse);
var
  oStr: TMemoryStream;
begin
  oStr := TMemoryStream.Create;
  try
    FDQueryDepartment.Open;
    FDQueryEmployees.Open;
    FDSchemaAdapter1.SaveToStream(oStr, TFDStorageFormat.sfJSON);
    AResponse.Body.SetStream(oStr, 'application/json', True);
  except
    oStr.Free;
  end;
end;
```

```
procedure TEMSTestResource1.Post(const AContext: TEndpointContext; const ARequest: TEndpointRequest;
  const AResponse: TEndpointResponse);
var
  LStream: TStream;
begin
  if not SameText(ARequest.Body.ContentType, 'application/json') then
    AResponse.RaiseBadRequest('content type');
  if not ARequest.Body.TryGetStream(LStream) then
    AResponse.RaiseBadRequest('no stream');
  LStream.Position := 0;
  FDSchemaAdapter1.LoadFromStream(LStream, TFDStorageFormat.sfJSON);
  FDSchemaAdapter1.ApplyUpdates
end;
```

クライアントからRESTメソッドを呼び出す



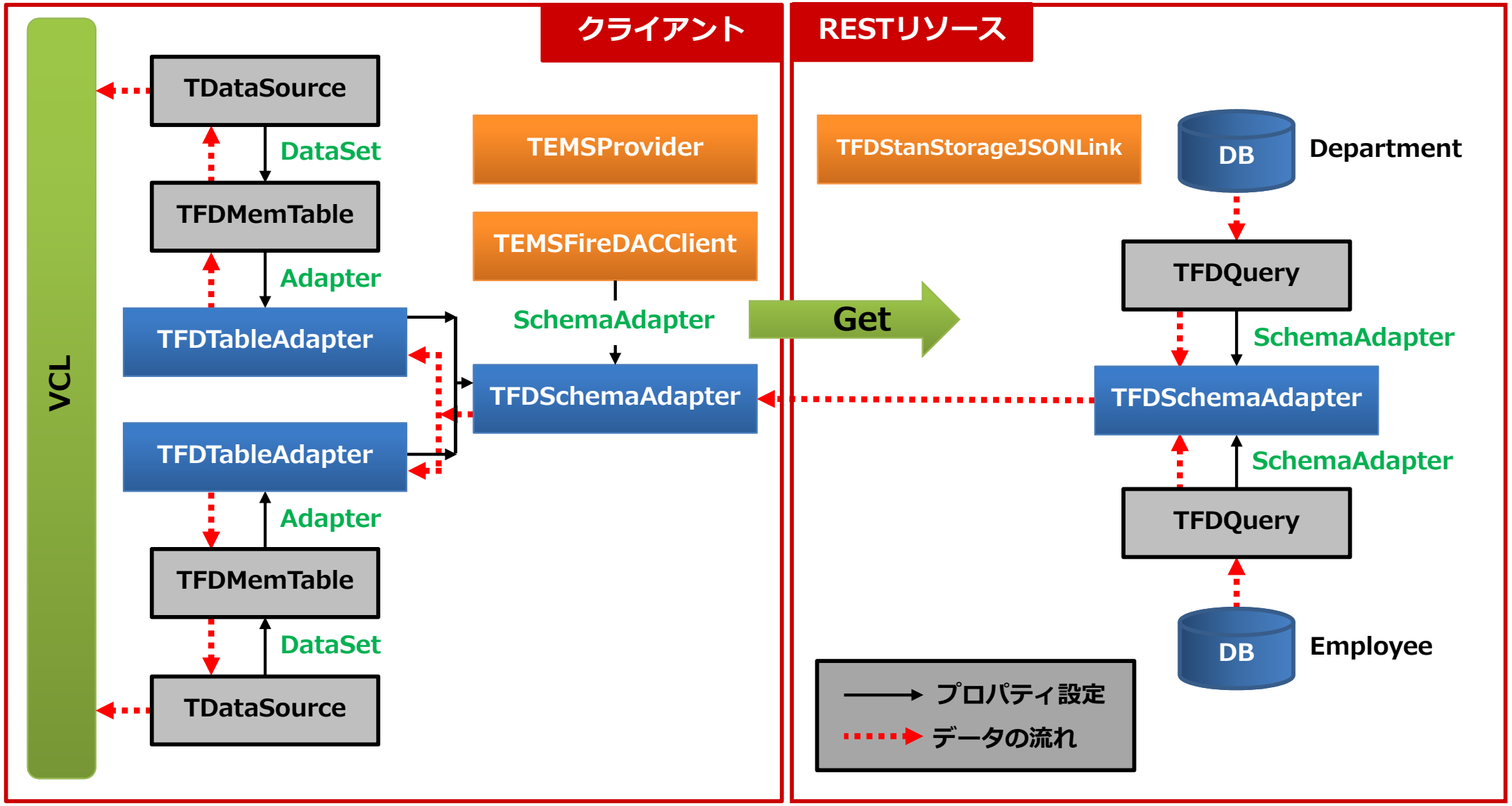
- データの取得（RESTリソースのGetメソッドが呼び出される）

```
procedure TForm1.btnGetTablesClick(Sender: TObject);
begin
  dm.EMSFireDACClient1.GetData;
end;
```

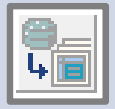
- データの更新（ RESTリソースのPostメソッドが呼び出される）

```
procedure TForm1.btnApplyUpdatesClick(Sender: TObject);
begin
  dm.EMSFireDACClient1.PostUpdates;
end;
```

コンポーネントの関係とデータの流れ



クライアント側の実装ポイントを整理

コンポーネント	設定ポイント
 **TEMSProvider**	接続するEMSサーバーのURL／ポートを指定する 指定するプロパティは **URLHost**／**URLPort**
 **TEMSFireDACClient**	REST APIでコールされるRESTリソース名を指定する 指定するプロパティは **Resource**
 **TFDSchemaAdapter**	データセットを一元的に管理する 一元キャッシュ更新をサポート
 **TFDTableAdapter**	RESTリソースのデータセットを指定する 指定するプロパティは **DatSTableName** **TFDSchemaAdapter**と接続される (**SchemaAdapter**) ※データセット単位に必要
 **TFDMemTable**	テーブルの更新情報をサーバー側に反映するために **CachedUpdates**プロパティを「True」に設定する **TFDTableAdapter**と接続される (**Adapter**)

EMSサーバーウィンドウ

サーバーを起動すると「EMSサーバーウィンドウ」が自動的に起動する

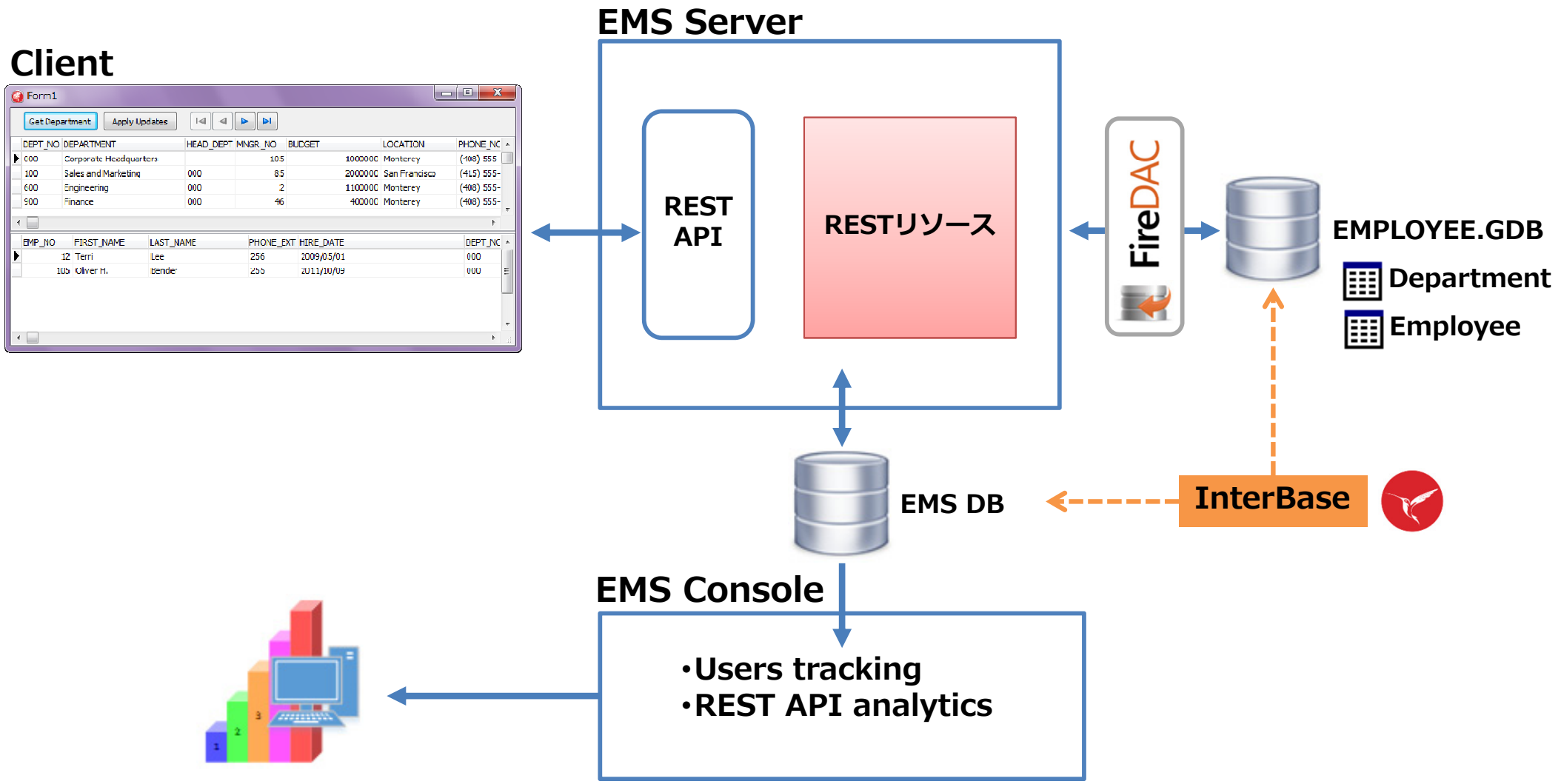
項　目	説　明
起動	EMSサーバーを起動
停止	EMSサーバーを停止
ブラウザを開く	デフォルト・ブラウザを開く
コンソールを開く	[EMS コンソール]サーバー ウィンドウを開く
ポート	EMSサーバーにアクセスするためのポート番号を入力
ログ	EMSサーバーメッセージを表示(リソース／サーバーへ要求)
ログ記録を有効にする	チェックするとログを記録
クリア	ログ表示をすべて削除

詳細は こちら を参照

パラメータを渡す

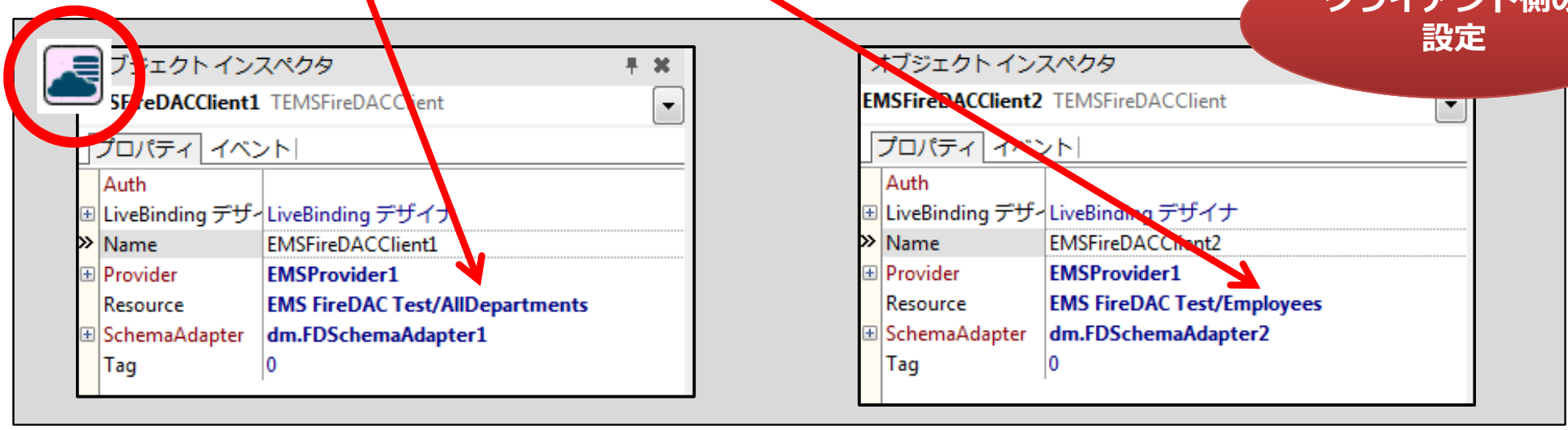
DEMONSTRATION

デモンストレーションの説明



RESTメソッドの定義

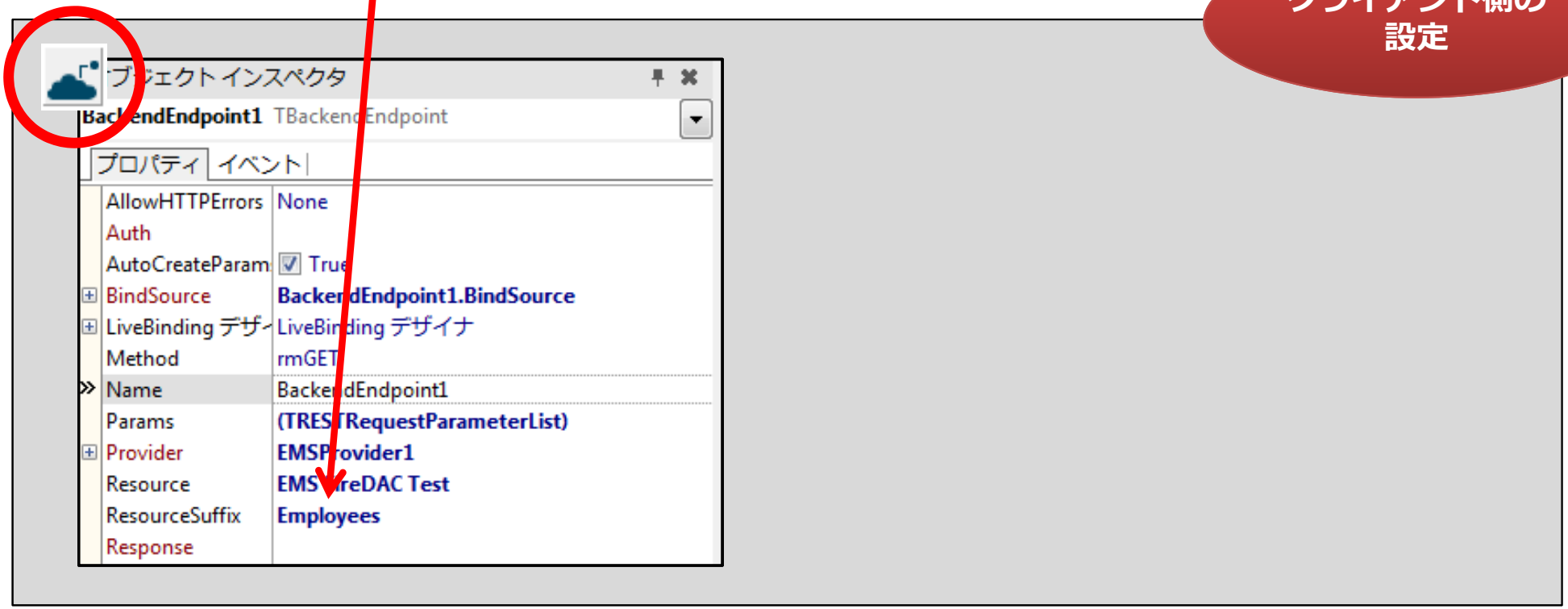
```
    ⋮
published
   [ResourceSuffix('AllDepartments')]
   procedure GetAllDepartments(const AContext: TEndpointContext;
     const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
   [ResourceSuffix('Employees')]
   procedure GetEmployees(const AContext: TEndpointContext;
     const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
   [ResourceSuffix('Employees')]
   procedure PostEmployees(const AContext: TEndpointContext;
     const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
 end;
```



・TEMSFireDACClient の Resource に ResourceSuffix で定義した文字列をセットする。

RESTメソッドの定義（続き）

```
 published
      :
    [ResourceSuffix('Employees')]
     procedure GetEmployees(const AContext: TEndpointContext;
       const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
      :
end;
```



・TBackendEndpoint の ResourceSuffix に ResourceSuffix で定義した文字列をセットする。

RESTメソッドの実装（Get）

```
procedure TEMSFireDACTestResource.GetAllDepartments(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  ms: TMemoryStream;
begin
  ms := TMemoryStream.Create;
  try
    FDQueryDepartment.SQL.Text := 'select * from department';
    FDQueryDepartment.Open;
    FDSchemaAdapter1.SaveToStream(ms, TFDStorageFormat.sfJSON);
    AResponse.Body.SetStream(ms, 'application/json', True);
  except
    ms.Free;
  end;
end;
```

・Getメソッド(GetAllDepartments)が呼ばれると、表(Department)のすべてのレコードがクライアントにJSON形式で戻される。

RESTメソッドの実装（Get）（続き）

```
procedure TEMSFireDACTestResource.GetEmployees(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  ms: TMemoryStream;
  vDeptNo: String;
begin
  ms := TMemoryStream.Create;
  try
    vDeptNo := ARequest.Params.Values['DeptNo'];
    if vDeptNo <> '' then
    begin
      FDQueryDepartmentEmployees.Close;
      FDQueryDepartmentEmployees.SQL.Text :=
        'select * from employee where dept_no = :DEPT';
      FDQueryDepartmentEmployees.ParamByName('DEPT').AsString := vDeptNo;
      FDQueryDepartmentEmployees.Prepare;
      FDQueryDepartmentEmployees.Open;
      FDSchemaAdapter2.SaveToStream(ms, TFDStorageFormat.sfJSON);
      AResponse.Body.SetStream(ms, 'application/json', True);
    end;
  except
    ms.Free;
  end;
end;
```

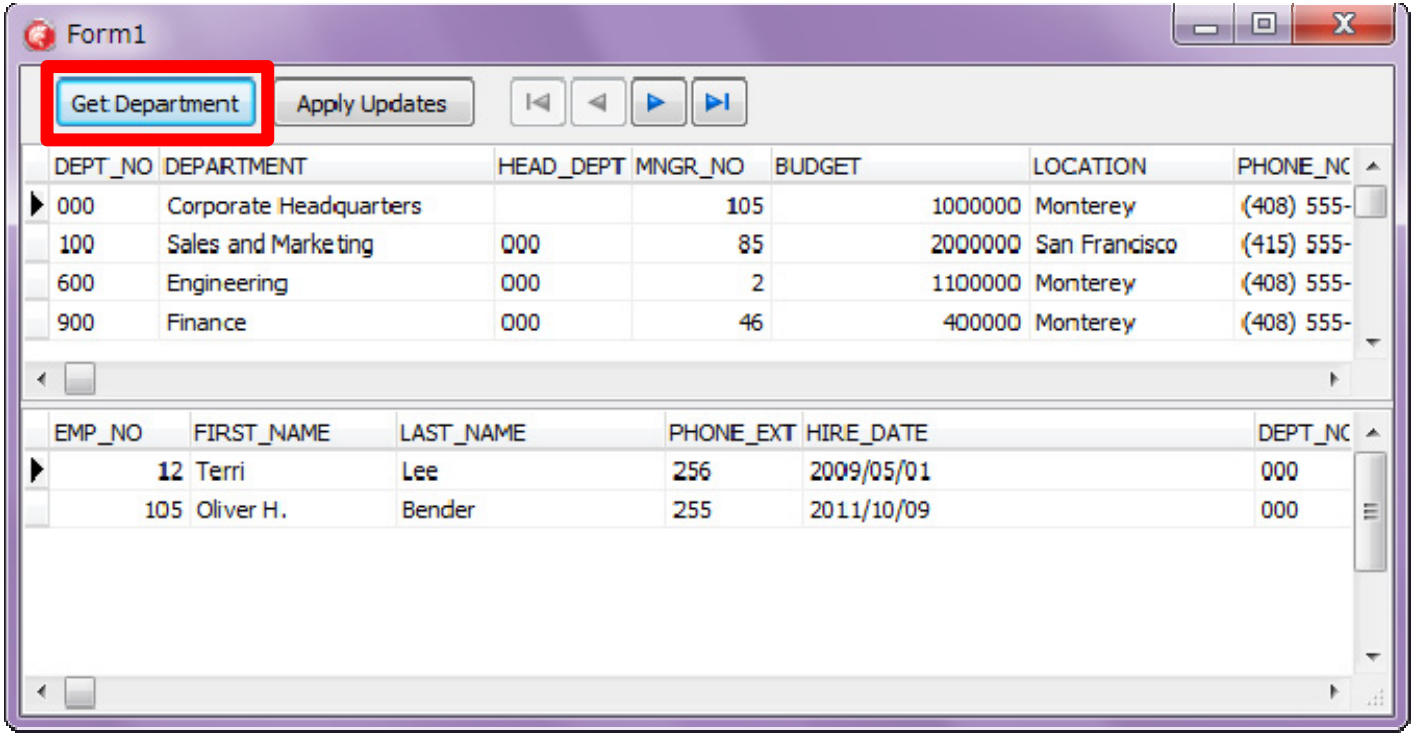
- Getメソッド(GetEmployees)が呼ばれると、表(Employee)から部門コードが等しいデータを抽出し、クライアントにレスポンス(JSON形式)で返す。

RESTメソッドの実装（Post）

```
procedure TEMSFireDACTestResource.PostEmployees(const AContext: TEndpointContext;
  const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  s: TStream;
begin
  if not ARequest.Body.TryGetStream(s) then AResponse.RaiseBadRequest('no stream');
  s.Position := 0;
  FDSchemaAdapter2.LoadFromStream(s, TFDStorageFormat.sfJSON);
  FDSchemaAdapter2.ApplyUpdates;
end;
```

・クライアントから送られた変更データ(JSON形式)で表(Employee)を更新する。

クライアントからRESTメソッド(Get)を呼び出す




```
procedure TForm1.btnGetTablesClick(Sender: TObject);
begin
  EMSFireDACClient1.GetData;
end;
```

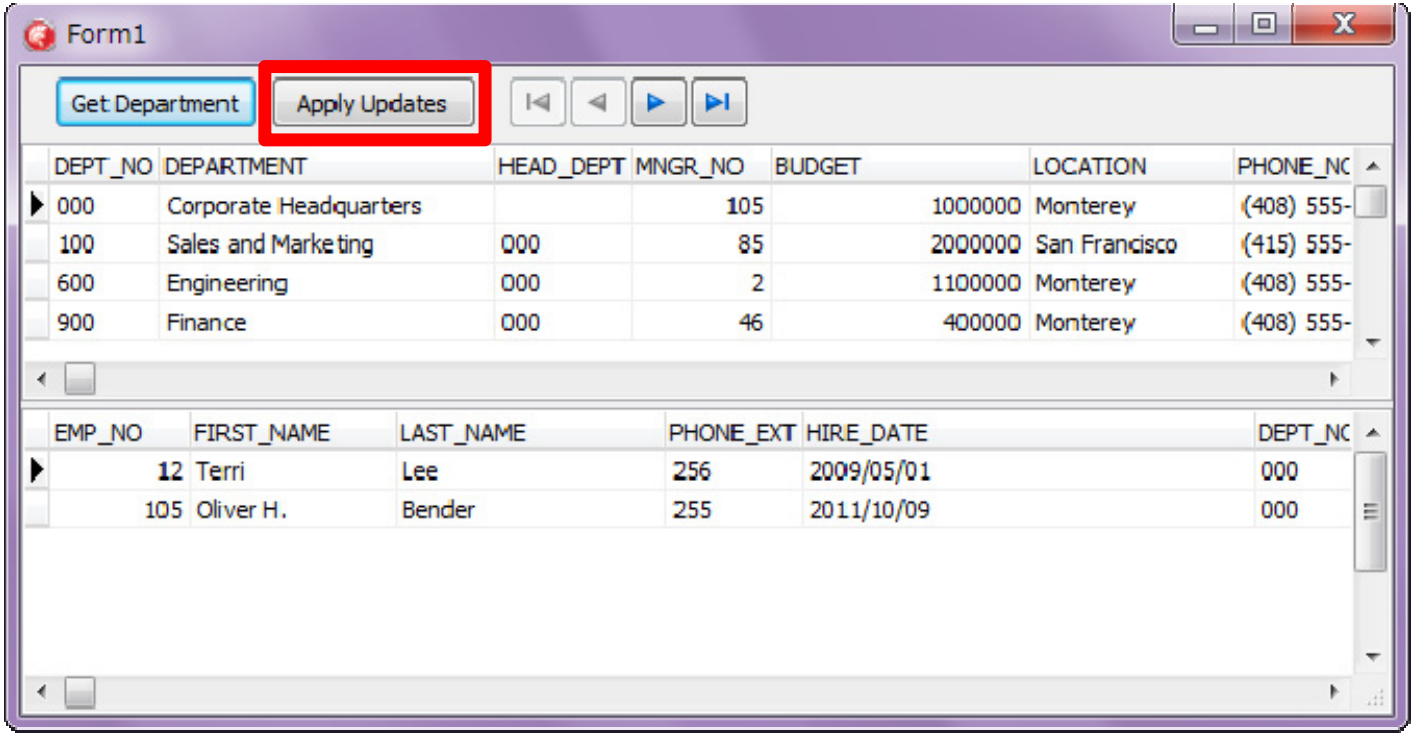
・TEMSFireDACClient の GetData メソッドでサーバ側のRESTメソッド(GetAllDepartments)を呼び出す。

クライアントからRESTメソッド(Get)を呼び出す（続き）

```
procedure TDataModule1.FDMemTable1AfterScroll(DataSet: TDataSet);
var
  ts: TStringStream;
  vDeptNo: String;
begin
  // 部門コードの取得
  vDeptNo := FDMemTable1.FieldByName('DEPT_NO').AsString;
  // サーバーへ部門コードを送出
  Form1.BackendEndpoint1.Params.Items[0].Value := vDeptNo;
  Form1.BackendEndpoint1.Execute;
  // サーバーから送られたJsonを取り込む
  ts := TStringStream.Create(Form1.BackendEndpoint1.Response.JSONText);
  try
    // FDMemTable2にDataSetをセットする
    FDMemTable2.LoadFromStream(ts, sfJson);
  finally
    ts.Free;
  end;
end;
```

・カーソル位置の部門コードを取得し、TBackendEndpoint の Params にセットする。
・TBackendEndpoint の Execute メソッドでサーバ側のRESTメソッド(GetEmployees)を呼び出す。
・サーバーからのレスポンス(JSON形式)をFDMemTableにセットする。

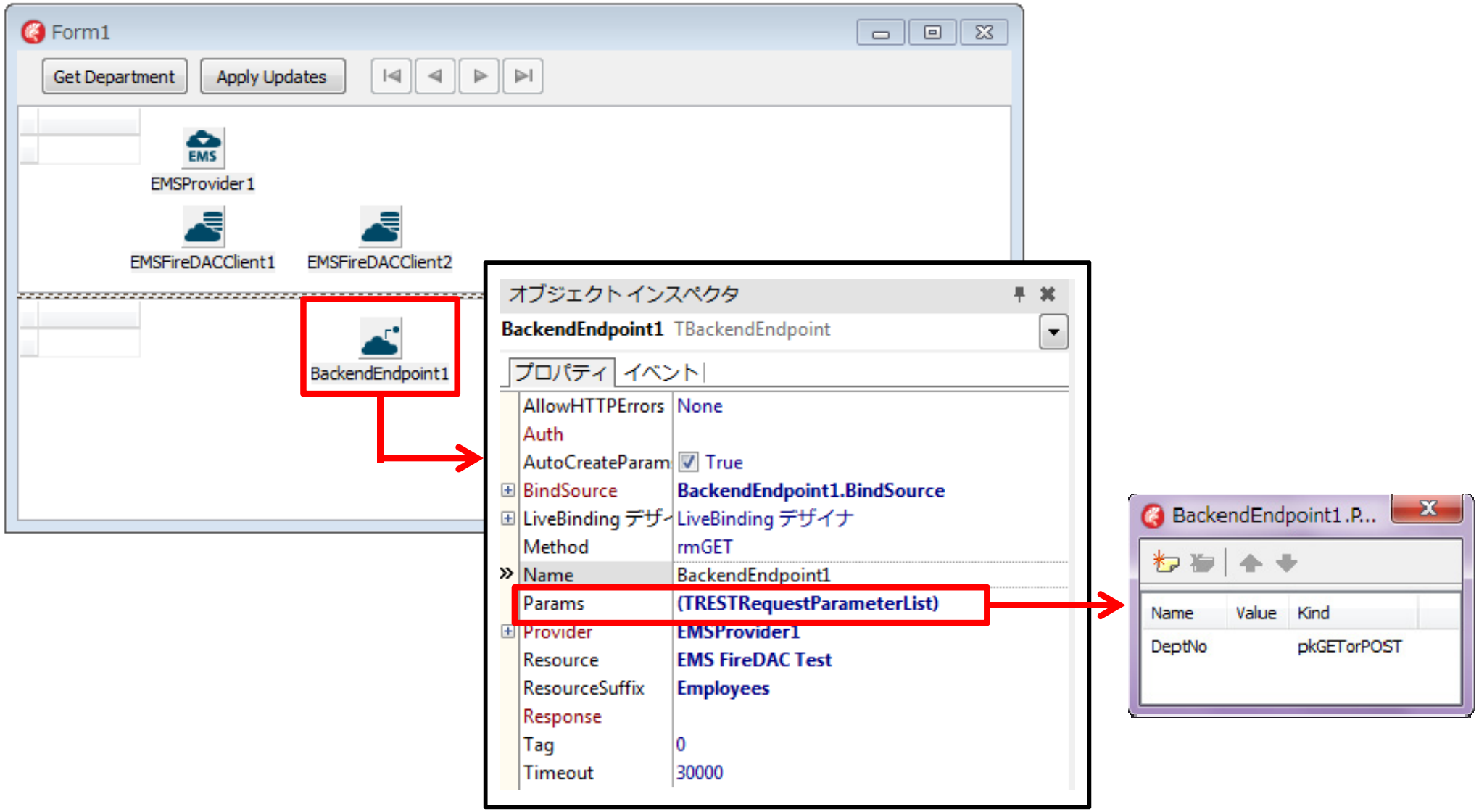
クライアントからRESTメソッド(Post)を呼び出す



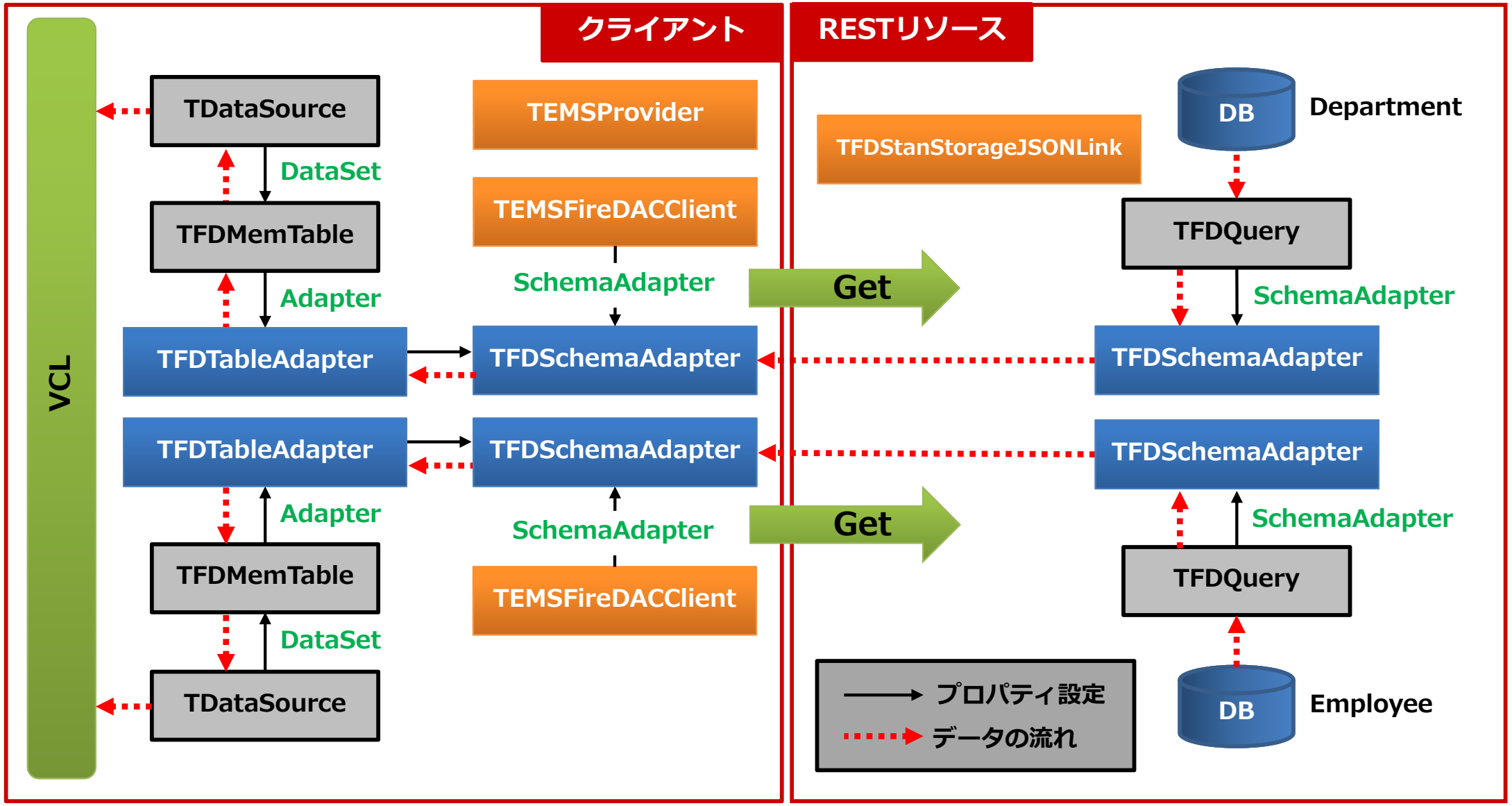

```
procedure TForm1.btnApplyUpdatesClick(Sender: TObject);
begin
  EMSFireDACClient2.PostUpdates;
end;
```

・TEMSFireDACClient の PostUpdate メソッドでサーバ側のRESTメソッド(PostEmployees)を呼び出す。

受け渡すパラメータの設定

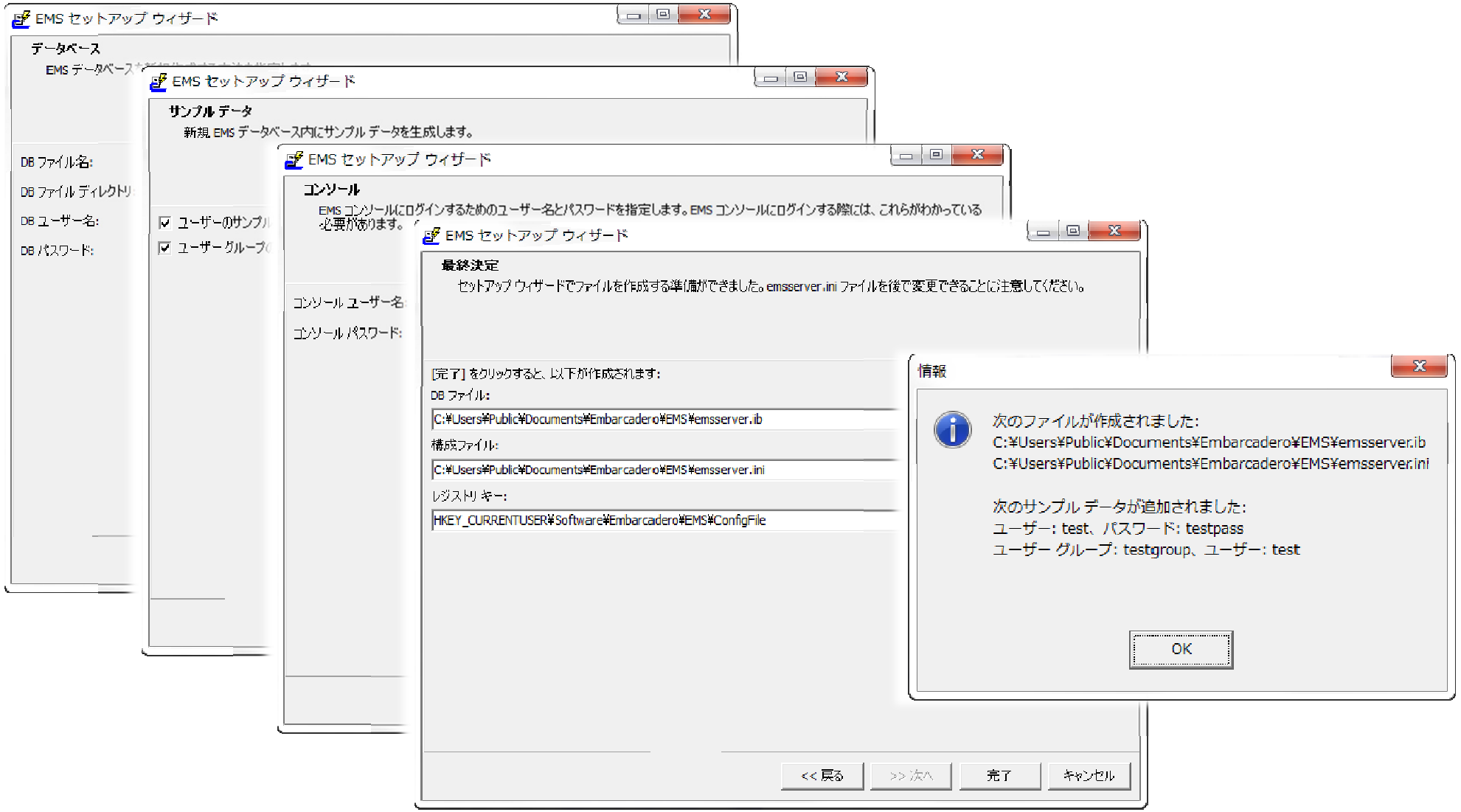


コンポーネントの関係とデータの流れ



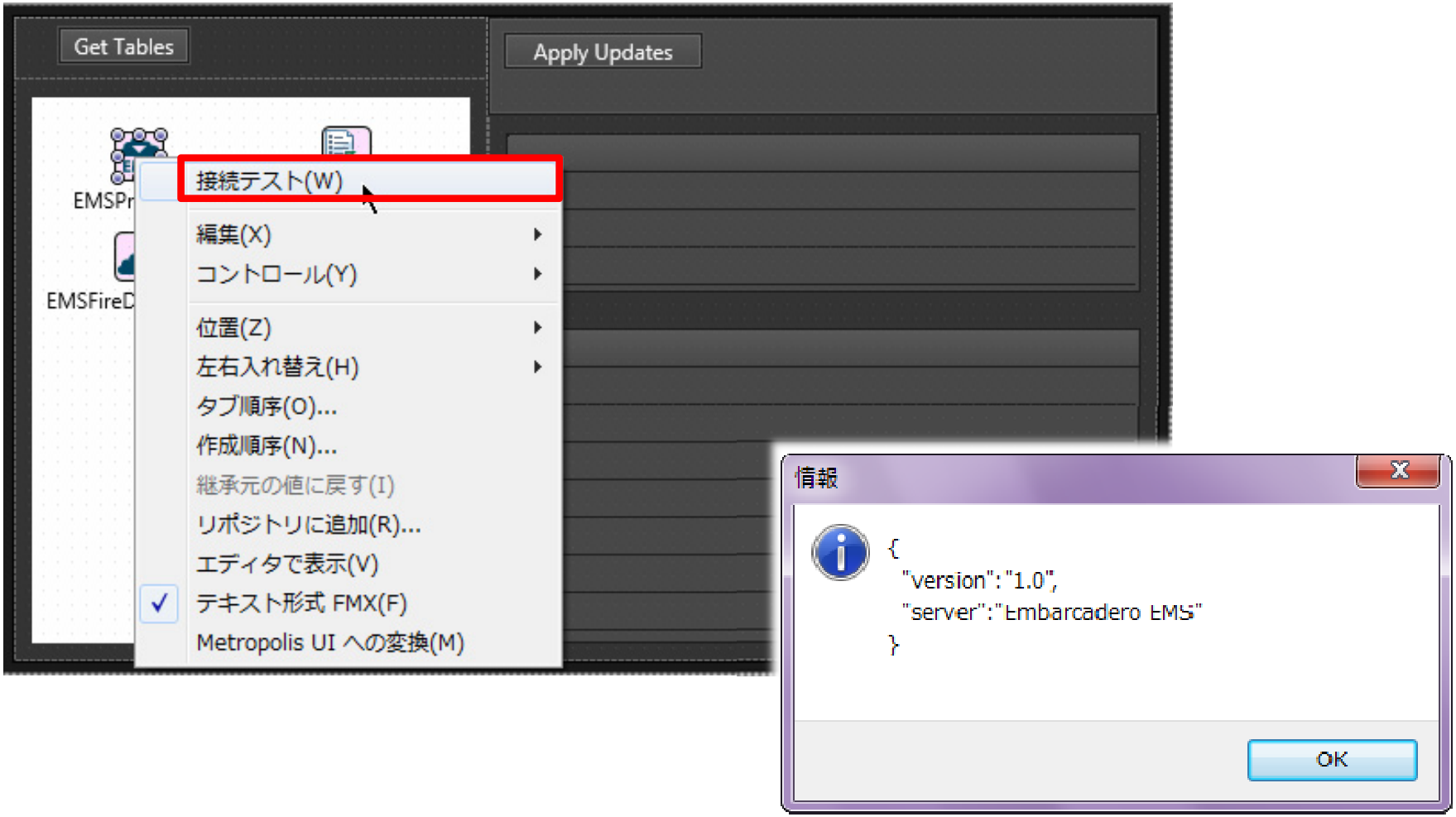
EMSサーバーを構築する

- はじめてEMSサーバーを起動すると、構築設定の画面が表示される。



EMSサーバーとの「接続」を確認する

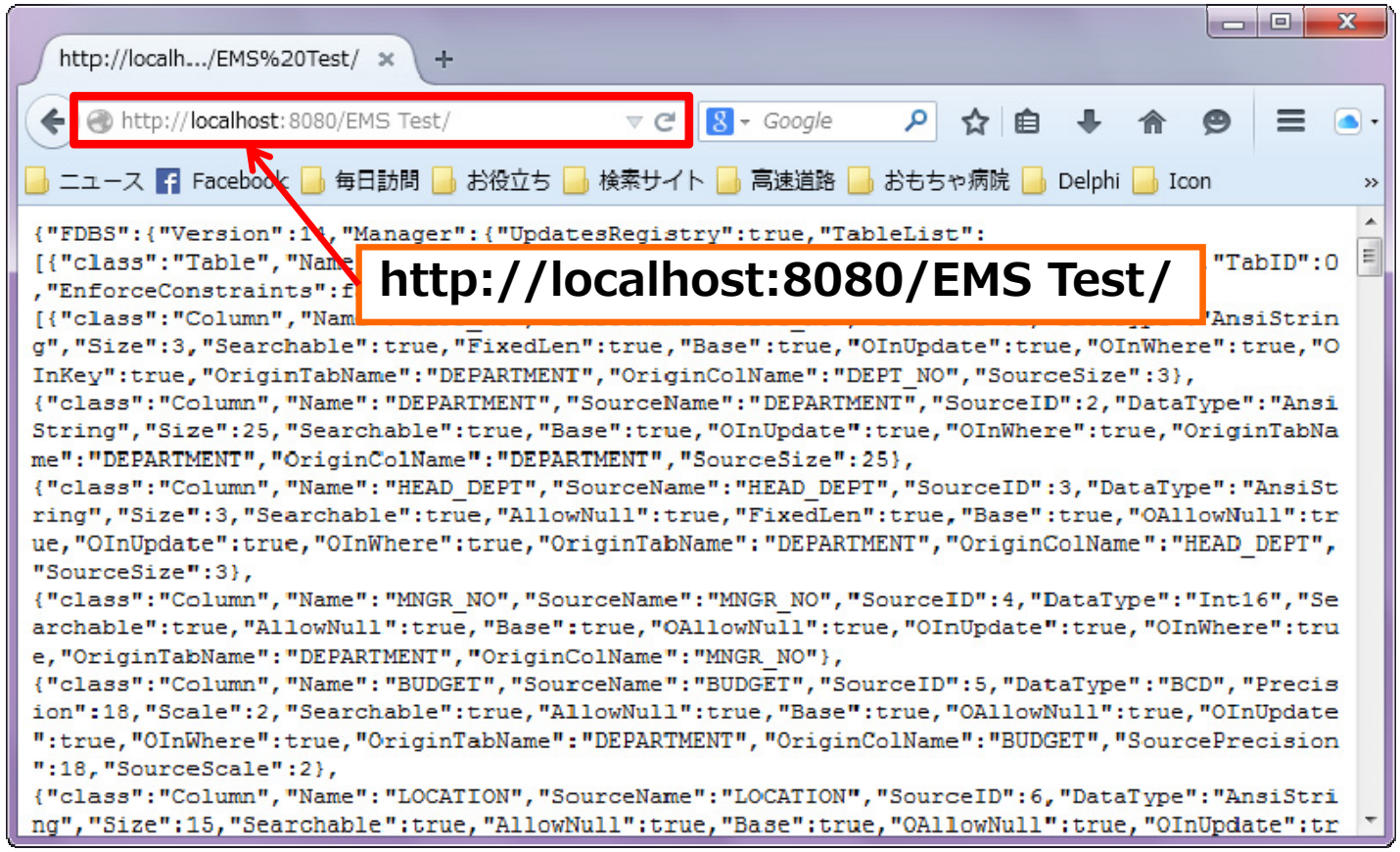
- TEMSProviderで右クリックし、メニューから［**接続テスト**］をクリック
- 接続が正しければEMSサーバーの現在のバージョンが表示される



ブラウザからEMSサーバーへアクセスする

EMSサーバーには、通常のWebブラウザを使ってアクセスできる

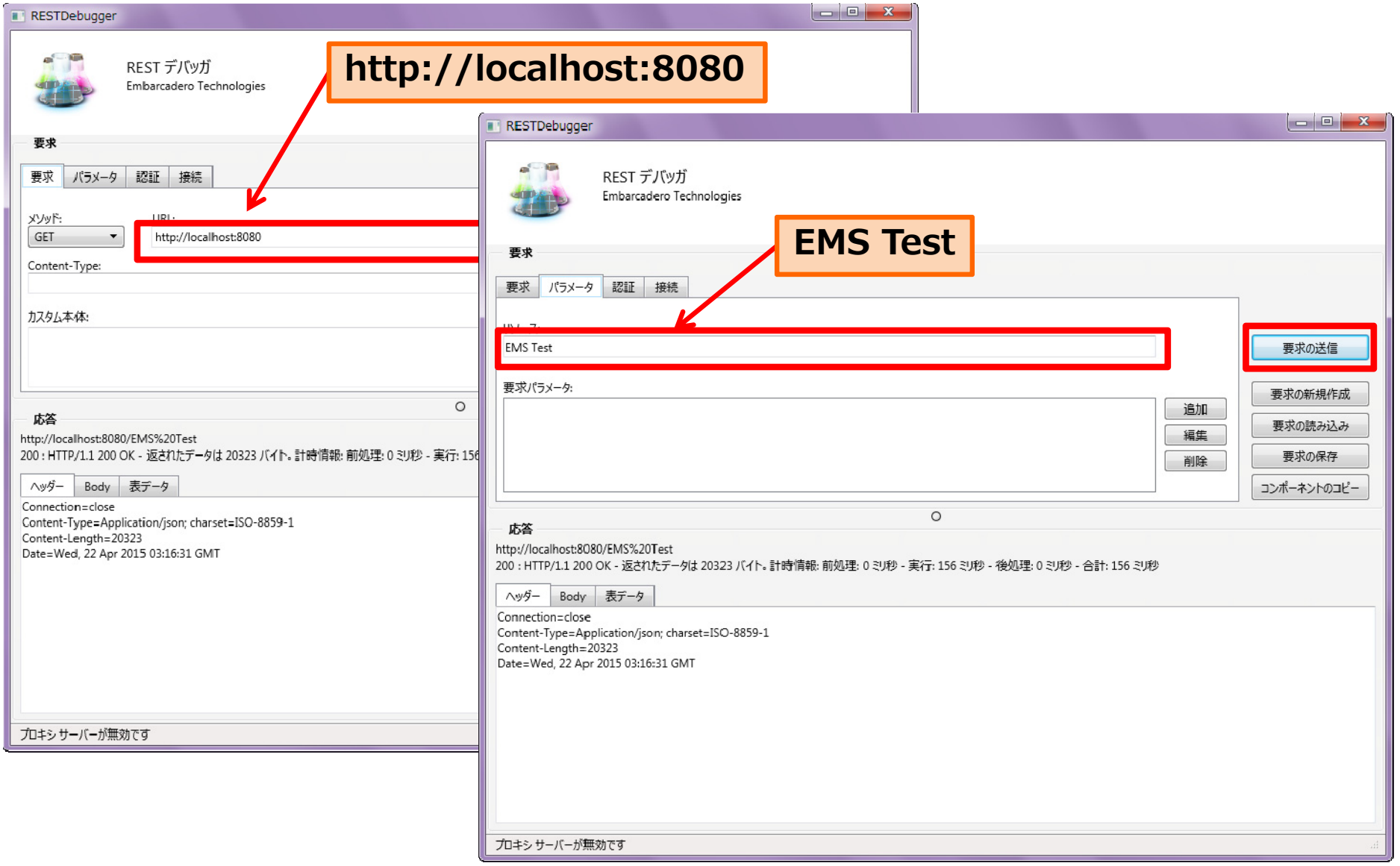
http://<IP アドレス>:<ポート番号>/<RESTリソース名>



取得したJSONの内容

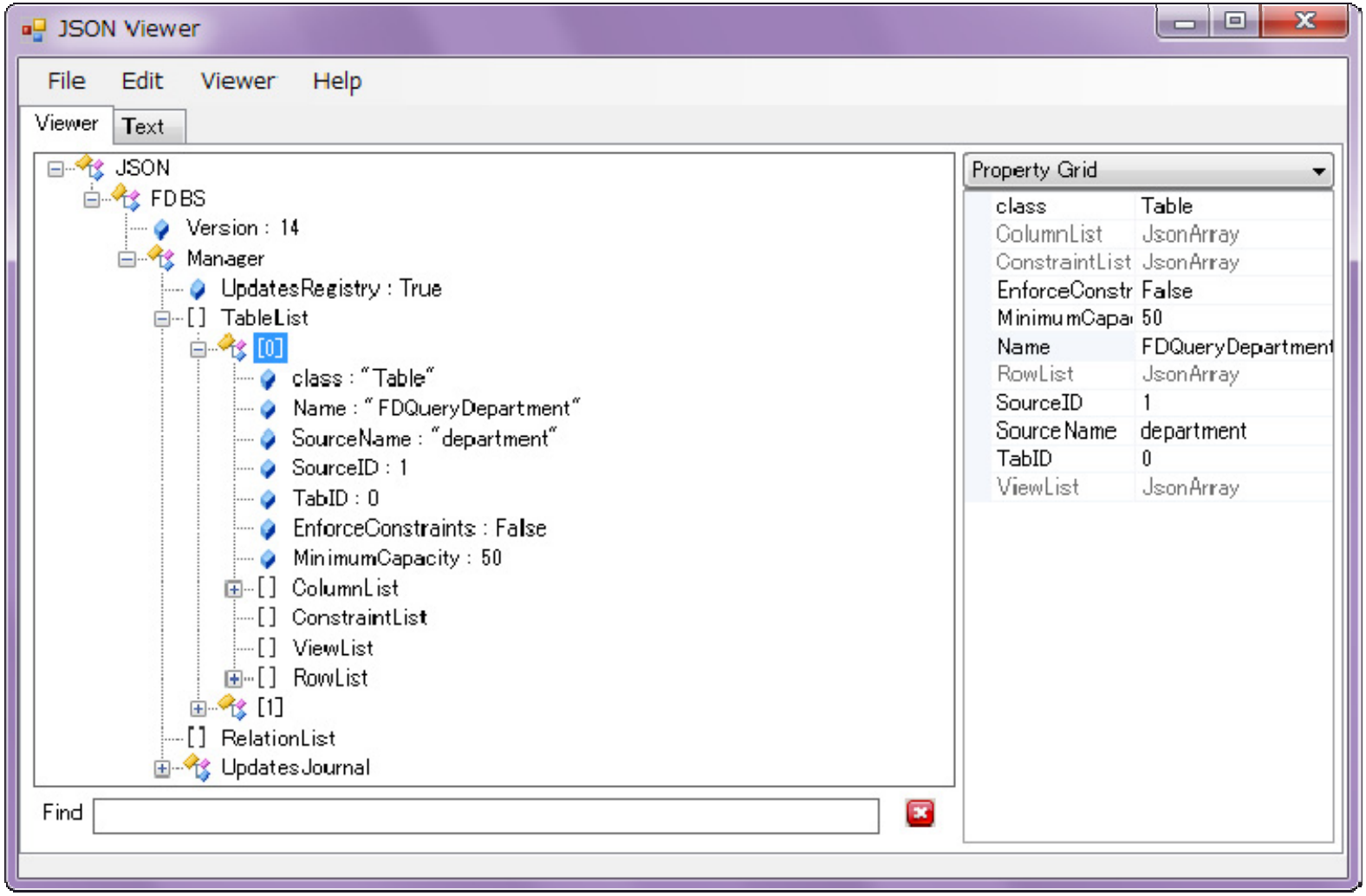
運用環境の場合：
http://<IP アドレス>/<サイト名>/emsserver.dll

RESTデバッガを使ってみる



JSONテキストをインターセプトする

JSON Viewer を使ってJSONテキストを表示する



*1 JSON Viewerは、次のサイトからダウンロードできます。
https://jsonviewer.codeplex.com/

XE7とXE8でEMSサーバーを共有させる

- ✓ XE8で「EMS DB」の構造が変更されている。
- ✓ XE8インストール時、 XE7 の「EMS DB」があれば、「EMSSERVER1.IB」となる。
- ✓ EMSの初期設定ファイル「emsserver.ini 」が、上記DBに書き換わる。
- ✓ 「Data」セクションの「Database」パラメータをコメントアウトする。
- ✓ 新たに「Database」パラメータを追加する。（赤枠の部分）

```
[Data]
;# Interbase connection parameters
;Database=C:¥Users¥Public¥Documents¥Embarcadero¥EMS¥emsserver1.ib
Database=C:¥Users¥Public¥Documents¥Embarcadero¥EMS¥emsserver.ib
UserName=sysdba
Password=masterkey
SEPassword=
InstanceName=
;# SEPassword connects to an encrypted database

[Server.Keys]
      :
```

- ✓ 環境に応じて次のようにEMSサーバーの指定を切り替える
  XE8 → emsserver1.ib
  XE7 → emsserver.ib

XE8の新機能

機能	Enterprise 以上	Professional	Starter
Introduced in XE7! API ホスティング、データアクセス、SQL データベースアクセスを含む REST ベースのミドルウェアスタック EMS（Enterprise Mobility Services）[6]	C D	C[2] D[2]	
Introduced in XE7! EMS サーバーにロード可能なパッケージを用いて、ビジネスロジックを実装、カスタム URI にマップし、カスタム REST API を作成可能[6]	C D	C[2] D[2]	
Introduced in XE7! Oracle、DB2、Microsoft SQL Server、Informix、SQL Server など多様なデータベースに接続できる FireDAC ハイパフォーマンスエンタープライズデータアクセスを統合[6]	C D	C[2] D[2]	
New in XE8! FireDAC / EMS 統合の改善 – 特に更新管理にフォーカス[6]	C D	C[2] D[2]	
New in XE8! iOS および Android 向け EMS プッシュ通知サーバーサポート[6]	C D	C[2] D[2]	
New in XE8! EMS 外部認証のサポート[6]	C D	C[2] D[2]	
New in XE8! 拡張された EMS 管理 API - 新たに EMS インストール、EMS プッシュリソースをサポート[6]	C D	C[2] D[2]	
New in XE8! データベースコネクションプーリングおよび他の EMS 機能の最適化[6]	C D	C[2] D[2]	
New in XE8! EMS クライアント側の配置を簡単にする新しい EMSClientAPI コンポーネント[6]	C D	C[2] D[2]	
Enhanced in XE8! EMS コンソールの Web ベースのインターフェイスにより、ユーザー／グループ／セッション／API コールの分析／レポートが可能[6]	C D	C[2] D[2]	
New in XE8! EMS コンソールから.CSV ファイルにデータをエクスポート[6]	C D	C[2] D[2]	
New in XE8! ユーザーとグループの分析機能の改善[6]	C D	C[2] D[2]	
New in XE8! ユーザーアカウントを管理できる EMS クライアントアプリケーション[6]	C D	C[2] D[2]	
*6 RAD Studio には、EMS サーバーパッケージおよび EMS サーバーにアクセスするクライアントを開発するためのツール、開発とテスト用に使用できる 5 ユーザーライセンスが含まれています。EMS を用いて開発したアプリケーションを配布するには、別途ユーザーライセンスが必要です。			

D: Delphi / C:C++Builder
C2 D2: Professionalでは、FireDAC Client/Server Pack が必要

動作環境／ライセンス について

動作環境

運用環境は、EMSサーバー/コンソールサーバー共に、Webサーバーにセットアップされることが推奨されている。（詳細はこちら）

対応しているWebサーバーは、IIS(Microsoft社)のみ。

しかし、ロードマップを見るとLinuxへの対応が明言されているので、将来はWindows/IIS以外の環境・Webサーバソフトで稼働できることも推測される。（ロードマップはこちら）

ライセンスについて

次のエディションのRAD Studioが必要

- Enterprise 版以上
- Professional 版の場合は、別途 FireDAC Client/Server Add-On Pack（有償）が必要
- Appmethod Windows

配布ライセンス

EMSの配布については、サービス機能を使用するユーザー数に応じてライセンスの購入が必要

※ 価格等詳細については、エンバカデロ宛てお問合せ下さい。

参考情報

- ✓ **Developer Skill Sprints**
  http://www.embarcadero.com/jp/landing-pages/skill-sprints

- ✓ **DataSnapユースケース研究**
  - 多層技術の概要と最適化、実践テクニック -
  http://edn.embarcadero.com/jp/article/images/43547/b4.pdf

- ✓ **多層分散型基幹業務システム構築の課題と解決**
  http://edn.embarcadero.com/jp/article/images/43816/c5.pdf

- ✓ **Mobilizing your Business with Enterprise Mobility Services Middleware**
  http://img.en25.com/Web/Embarcadero/%7Bfc7a9f78-251e-4540-9c6c-7172feca344e%7D_Mobilizing_Your_Business_with_EMS_Middleware.pdf

- ✓ **はじめてのFireDAC**
  http://edn.embarcadero.com/jp/article/images/43368/a2.pdf

関連情報 (Docwiki)

- ✓ エンタープライズ モビリティ サービス（EMS）
- ✓ EMS サーバー
- ✓ EMS コンソール サーバー
- ✓ EMS パッケージのインストール
- ✓ DataSnap の概要とアーキテクチャ

ご清聴
ありがとうございました

人と技術をつなぐ

Yoshiki.tanaka-avsoft@nifty.com