EMBARCADERO TECHNOLOGIES®

White Paper

Delphi 2010 DataSnap
活用ガイド

Bob Swart – Bob Swart Training & Consultancy (eBob42)

2009 年 12 月

目　次

1. DataSnap の歴史

DataSnap は、元来、Delphi 3 で「MIDAS」、Delphi 4 では MIDAS II、Delphi 5 では MIDAS III として、TCP/IP、HTTP、(D)COM の接続機能における COM ベースのリモートデータモジュールを構築するとても強力な手段として提供されてきました。Delphi 6 からは、「DataSnap」という名称に変更され、Delphi 2007 までこのフレームワークはほぼそのままで提供されていました。

Delphi 2009 で、DataSnap は再構築され、COM 依存性の排除、リモートサーバーオブジェクトの軽量化が行われ、TCP/IP のみに限られていたクライアント接続が、Delphi Prism 2009 では、.NET クライアントの構築もサポートしました。

Delphi 2010 では、DataSnap 2009 アーキテクチャー上に、このフレームワーク向けの 2 つのウィザードで新しいターゲット（VCL フォーム、Windows サービス、コンソール、さらに Web ターゲットとして ISAPI、CGI または Web App Debugger)をサポートし、HTTP(S)トランスポートプロトコル、HTTP 認証、クライアントコールバック機能、REST と JSON のサポート、圧縮（組み込み済み）と暗号化をサポートするフィルターを含む、新しい拡張された機能が提供されました。

1.1 DataSnap サンプルデータ

このホワイトペーパーでは、デモと使用例をあわせて演習していただくよう強くお勧めします。Delphi では、DBX4、dbGo for ADO やその他のデータアクセステクノロジーなどにより、多くの異なったデータベースシステムをサポートしていますが、デモと使用例の演習を簡単にするために、以下のディレクトリ上の employee.jds データベースと、BlackfishSQL を使用した DBX4 を DBMS として使用します。

Windows XP では、employee.jds データベースは、

C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL

フォルダ内に、Windows Vista、または Windows 7 では、

C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL

フォルダ内にあります。

画面ショットからおわかりのように、使用例の作成には OS として Windows 7 Professional を、そして DataSnap ISAPI サーバーの配布には Windows Server 2008 Web Edition を使用しています。

2. DataSnap Windows ターゲット

DataSnap 2010 は、VCL フォーム、Windows サービス、そしてコンソールアプリケーションの 3 つの異なった Windows ターゲットをサポートしています。この章では、それぞれのメリットと違いについて、そして、それぞれのターゲットタイプを適用するベストケースについて説明します。

DataSnap サーバーとクライアントの使用例を構築することによって、TDSServer、TDSServerClass、TDSTCPServerTransport、TDSHTTPService、TDSHTTPWebDispatcher そして TDSHTTPService-AuthenticationManager コンポーネントと、同様にカスタムサーバーメソッドと TDSServerModule クラスをカバーします。

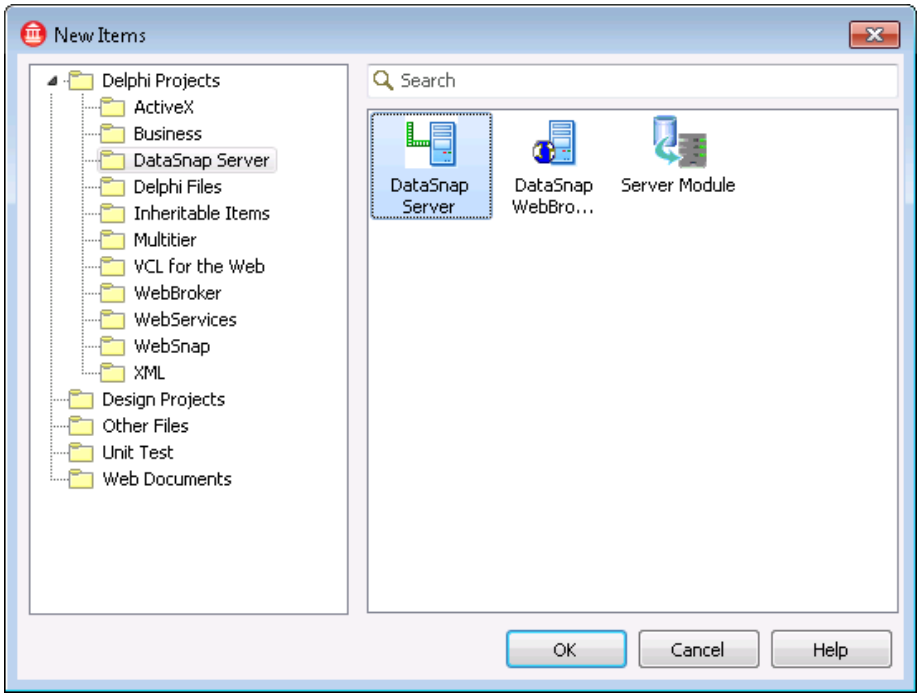
異なったトランスポートプロトコル（TCP、HTTP）についてはそれらの効果と潜在的なメリットを説明していきます。DataSnap サーバーオブジェクトのライフタイムについての異なったオプション（サーバー、セッション、そして起動）について説明し、そしてこれらの効果と現実的な提案をします。最後に、配布における課題についてもいくつか言及します。

2.1. DATASNAP サーバーサンプル

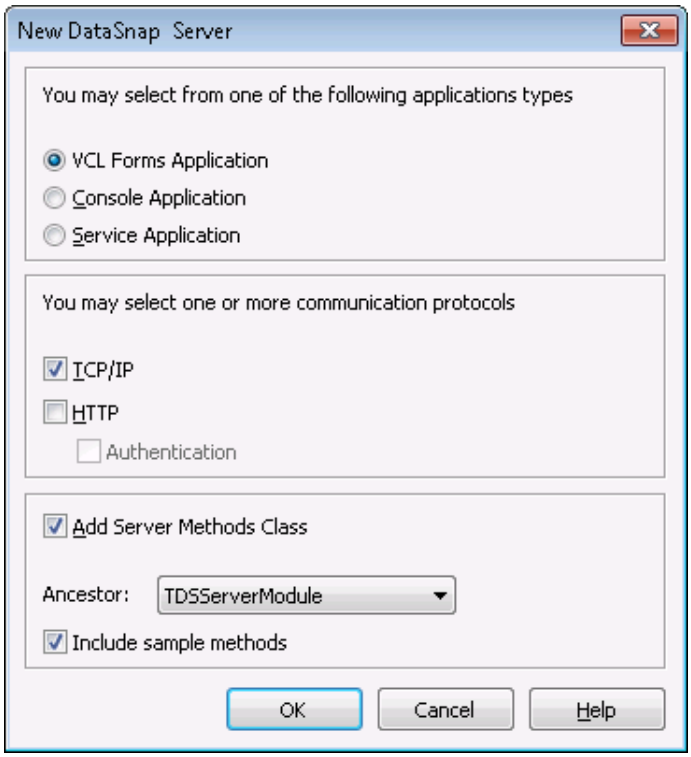
オブジェクトリポジトリには、2 つの異なる DataSnap サーバーウィザードがあります。1 つは DataSnap サーバープロジェクトをベースにしたウィンドウを作成するもの、もう 1 つは IIS（Internet Information Services）のような Web サーバーによってホストされなければならない DataSnap サーバープロジェクトをベースにした WebBroker を作成するものです。まず前者からスタートします。

Delphi 2010 を起動し、［ファイル｜新規｜その他］メニューを選択すると、オブジェクトリポジトリに「DataSnap サーバーウィザード」があります。オブジェクトリポジトリ の DataSnap サーバーカテゴリーは次の 3 つのアイコンを表示します。

- DataSnap Server
- DataSnap WebBroker Server
- Server Module



最初の DataSnap Server アイコンをダブルクリックします（他の 2 つについてはこのホワイトペーパー内で後ほど説明します）。すると次のようなダイアログが表示されます。



このダイアログの最初のセクションでは、プロジェクトのターゲット設定を行います。デフォルトは、メインフォームと一緒にビジュアルな VCL フォームアプリケーションを作成します。2 番目はコンソールアプリケーションで、コンソールウィンドウで動作します。これはリクエストとその応答状況をトレースするには理想的です（サーバーアプリケーションの中で「何が行われているか」を示すには、シンプルな Writeln 文を使用することができます）。この両方のアプリケーションタイプは、デモと初期開発には理想的ですが、DataSnap サーバーアプリケーションを最終的に配布したい場合には、それほど理想的ではありません。新しい DataSnap のアーキテクチャーは COM ベースではないため、受信したクライアント接続は DataSnap サーバーアプリケーションを「起動」することができないのです。そのため、受信したクライアントリクエストを処理するためには、DataSnap サーバーは「起動し、稼動している」状態でなければなりません。そして、もし 24 時間 365 日、受信リクエストを処理するのであれば、DataSnap サーバーアプリケーションは、その間ずっと稼動し続けていなければなりません。VCL Forms またはコンソールアプリケーションのために、使用者は Windows にログオンした状態でサーバーアプリケーションを稼動させていなければならないことを意味しますが、これは決して理想的な状態ではありません。実際に、この場合は 3 番目の選択の方がより望ましいのです。つまり、Windows サービスアプリケーションの使用です。これはマシンが起動された時に自動的にインストールと設定がなされ、実行されます（誰かがマシンにログインしている必要はありません）。サービスアプリケーションのマイナス面は、デフォルトではデスクトップ上にアプリケーション自身が表示されず、デバッグが多少難しくなることがあげられます。この 3 つのアプリケーション形態からベストのものを得るために、VCL フォーム DataSnap サーバーアプリケーション、コンソール DataSnap サーバーアプリケーション、そして Windows サービス DataSnap サーバーアプリケーションのそれぞれのプロジェクトグループを作成する手順を紹介します。これらの 3 つはすべて同じカスタムサーバーメソッドを共有しており、その結果 DataSnap サーバーアプリケーションを、必要なときに 3 つの異なるターゲットにコンパイル（そして配布）することが可能になります。

新しい DataSnap サーバーダイアログの 2 番目のセクションでは、使用可能ないろいろな通信プロトコルを選択できます。DataSnap 2009 と比較すると、HTTP 通信のほかに HTTP 認証も利用できるようになりました。柔軟性を持たせるために、ここではすべてのオプションをチェックしておきます。これにより、TCP/IP と HTTP、そして HTTP と HTTP 認証の組み合わせについても同様に利用することができます。

新しい DataSnap サーバーダイアログの最後のセクションは、すでに設定されています。サーバーメソッドクラスを生成するメソッドがあるとすれば、次のような上位クラスを選択することができます。

TPersistent

TDataModule

TDSServerModule

ベストなのは最後の選択で、起動の直後からメソッドのための RTTI を可能にします（TDataModule が標準だと感じることがあるかもしれませんが、何のデータベースも非ビジュアルコントロールも使用していない場合にはおそらく TPersistent の使用で十分です）。

以下の DSServer.pas ユニットのコードからの抜粋は、TDSServerModule と TProviderDataModule の関連を示しており、TDataModule から順に派生したものです。

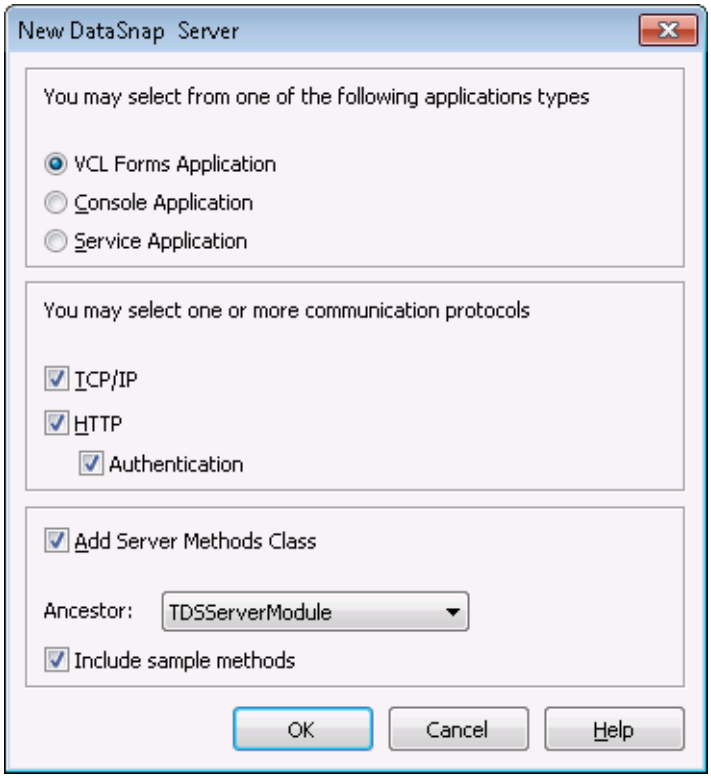
```
TDSServerModuleBase = class(TProviderDataModule)
public
  procedure BeforeDestruction; override;
  destructor Destroy; override;
end;

{$MethodInfo ON}
TDSServerModule = class(TDSServerModuleBase)
end;
{$MethodInfo OFF}
```

何をすべきかがわからないときには、TDSServerModule を上位クラスとして選択してください。

2.1.1. マルチターゲットプロジェクトグループ – VCL フォーム

お約束したとおり、マルチターゲット DataSnap サーバープロジェクトグループを作成しましょう。初めにすべての通信プロトコルを選択した DataSnap サーバーとして、VCL Forms Application から始めましょう。



この結果、新しいプロジェクトが作成されます。デフォルトでは Project1.dproj という名称で、3 つのユニットがあり、これらはデフォルトでは ServerContainerUnit1.pas、ServerMethodsUnit1.pas、Unit1.pas という名称です。最初に［ファイル｜保存］を実行しますが、必要に応じて適当なファイル名を指定して保存します。Unit1.pas ファイルは、MainForm.pas という名称で、ServerContainerUnit1.pas は ServerContainerUnitDemo.pas という名称、ServerMethodsUnit1.pas は ServerMethodsUnitDemo.pas、そして Project1.dproj は、DataSnapServer.dproj という名称で保存します。

コンソールアプリケーションとサービスアプリケーションをプロジェクトグループに加えます。そして、この時点での状態を確認し、プロジェクトをコンパイルしてみましょう。DataSnapServer プロジェクトをコンパイルすると、エラーメッセージが表示されるはずです（これは私のミスで、生成した ServerMethodsUnit1.pas ユニットのファイル名を変更したからです）。すなわち、このエラーメッセージは、実行セクションの節を利用した ServerMethodsUnit ユニットを含む ServerContainerUnitDemo.pas ユニットによって生じています（Line 30）。同様に、ServerMethodsUnit1.pas を ServerMethodsUnitDemo として保存します。この問題を解決するために、Uses 節を変更して、このユニットのファイル名の変更を反映し、再度コンパイルします。今度は ServerMethodsUnit1 が条件付けとして使われている TServerMethods1 クラスの line 37 にエラーが発生します。ここでは、ServerMethodsUnit1 を ServerMethodsUnitDemo へ変更します。これで、その他の問題なく DataSnap Server プロジェクトをコンパイルすることができます。

ServerContainerUnitDemo の実装部は、次のようになります。

```
implementation
uses
  Windows, ServerMethodsUnitDemo;
```

```
{$R *.dfm}

procedure TServerContainer1.DSServerClass1GetClass(
  DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
begin
  PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
end;

end.
```

2.1.1.1. ServerContainerUnitDemo

ServerContainerUnitDemo ユニットのデザインタブを確認すると、次の 5 つのコンポーネントがあります。

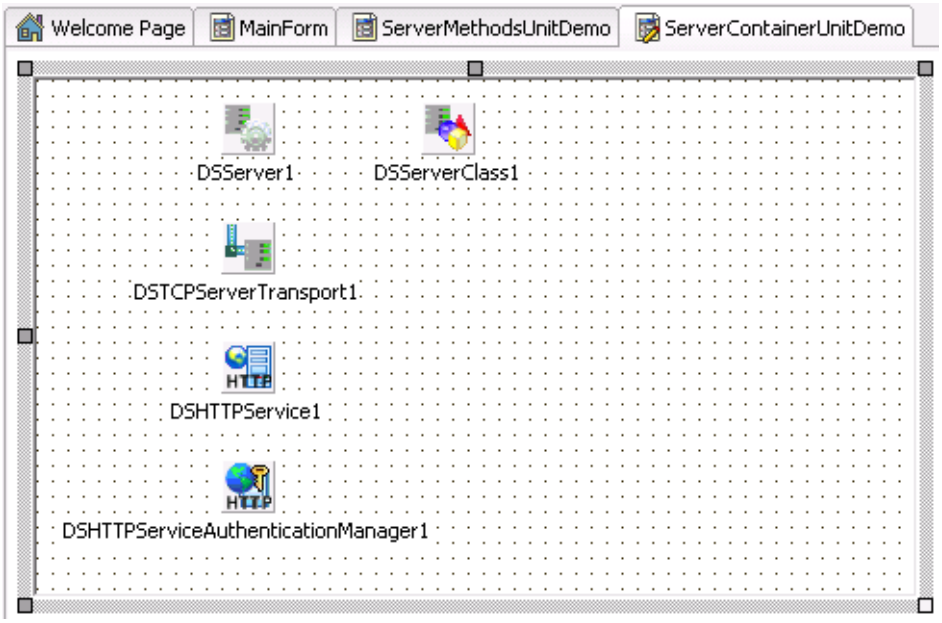
TDSServer

TDSServerClass

TDSTCPServerTransport (TCP/IP コミュニケーション用)

TDSHTTPService (HTTP コミュニケーション用)

TDSHTTPServiceAuthenticationManager コンポーネント(HTTP 認証用)



最初の 2 つは、いつも含まれており、その他の 3 つについては選択したコミュニケーションプロトコルオプションによります。

2.1.1.1.1. TDSServer

TDSServer コンポーネントには、次の 4 つのプロパティしかありません。AutoStart、HideDSAdmin、Name、そして Tag です。AutoStart プロパティのデフォルトは True にセットされており、これは DataSnap サーバーが、フォームが生成されるのと同時に起動することを意味しています。AutoStart を True に設定しない場合には、手動で Start メソッドを呼び出し、いつでも Stop を呼び出すことが

できます。DataSnap サーバーがすでに起動しているかどうかを調べるには、Started 関数を利用します。

HideDSAdmin プロパティのデフォルトは False に設定されています。True に設定すると、この DataSnap Server に接続するクライアントは、TDSAdmin クラスから組み込みのサーバーメソッドを呼び出すことができません。TDSAdmin クラスは実際にはクラスではなく、DSNames ユニット内で文書化されていて TDSAdmin メソッドと呼べるものです:

```
  TDSAdminMethods = class
  public
    const CreateServerClasses = 'DSAdmin.CreateServerClasses';
    const CreateServerMethods = 'DSAdmin.CreateServerMethods';
    const FindClasses = 'DSAdmin.FindClasses';
    const FindMethods = 'DSAdmin.FindMethods';
    const FindPackages = 'DSAdmin.FindPackages';
    const GetPlatformName = 'DSAdmin.GetPlatformName';
    const GetServerClasses = 'DSAdmin.GetServerClasses';
    const GetServerMethods = 'DSAdmin.GetServerMethods';
    const GetServerMethodParameters = 'DSAdmin.GetServerMethodParameters';
    const DropServerClasses = 'DSAdmin.DropServerClasses';
    const DropServerMethods = 'DSAdmin.DropServerMethods';
    const                    GetDatabaseConnectionProperties                    =
'DSAdmin.GetDatabaseConnectionProperties';
  end;
```

TDSServer コンポーネントには次の 5 つのイベント、OnConnect、OnDisconnect、OnError、OnPrepare、OnTrace があります。異なった状況に反応するこれらの 5 つのイベントについてイベントハンドラを記述できます。例えばログファイルの一連のテキストです。

OnConnect、OnDisconnect、OnError、OnPrepare イベントは TDSEventObject から派生する引数を持っており、それは DxContext、Transport、Server そして DbxConnection コンポーネントのためのプロパティが含まれます。OnConnect と OnDisconnect に使われる TDSConnectEventObject 型にも ConnectionProperties と ChannelInfo が含まれます。TDSErrorEventObject にはエラーを発生させる Exception が含まれており、TDSPrepareEventObject には、MethodAlias と使用したい ServerClass のためのプロパティが含まれます。

OnTrace イベントハンドラは TDBXTraceInfo 型の引数を持っています。生成された OnTrace イベントハンドラもまた TDBXTraceInfo 型と CBRType がコンパイラにとって未知であるため、Code Insight でいくつかのエラーが発生することに注意してください。この問題を解決するためには DBXCommon ユニット（DBXTraceInfo 型のため）と DBCommonTypes ユニット（CBRType のため）を加える必要があります。

OnConnect イベントハンドラでは、ChannelInfo の接続をテストできます。以下はその例です（カスタム関数 LogInfo を使って、ログファイルにこの情報を書き込みます）。

```
procedure TServerContainer1.DSServer1Connect(
  DSConnectEventObject: TDSConnectEventObject);
begin
  LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
end;
```

そして OnTrace イベントハンドラの内部では、サーバーが何を行っているのかを知るよい方法として TraceInfo.Message のログを記録することができます。

```
function TServerContainer1.DSServer1Trace(TraceInfo: TDBXTraceInfo): CBRType;
begin
  LogInfo('Trace ' + TraceInfo.CustomCategory);
  LogInfo('  ' + TraceInfo.Message);
  Result := cbrUSEDEF; // take default action
```

```
end;
```

クライアントサイドでも、DataSnap クライアントと TSQLConnection コンポーネント（この DataSnap サーバーのためにクライアントを構築するときにデモンストレーションする予定のコンポーネントです）に接続した TSQLMonitor コンポーネントにより、使用されているサーバーとの間の通信をトレースすることができることを覚えておいてください。

トレース出力の例は次のようになります:

```
17:05:55.492 Trace
17:05:55.496   read 136 bytes:{"method":"reader_close","params":[1,0]}
               {"method":"prepare","params":[-1,false,"DataSnap.ServerMethod",
                "TServerMethods1.AS_GetRecords"]}
17:05:55.499 Prepare
```

ご覧のとおり、TraceInfo.Message はバイト数についての情報と同様にメソッドと呼ばれる情報も含んでいます。

2.1.1.1.2. TDSServerClass

TDSServerClass コンポーネントには、リモートクライアント（ダイナミックメソッドの起動を使用している）に公開されたメソッドを公開するために使われるサーバーサイドのクラスを定義する役割があります。

TDSServerClass コンポーネントは、TDSServer コンポーネントを指す Server プロパティを持っています。Name と Tag プロパティのほかに、もう 1 つは重要なプロパティは、LifeCycle プロパティです。これはデフォルトでは Session に設定されていますが、Server または Invocation に設定することもできます。長いものから短いものまで、サーバーの意味するところは、Session と Invocation はサーバーの全体のライフタイム、DataSnap セッションのライフタイム、または、メソッドの単一の起動のためのクラスのインスタンスに使われることを意味しています。Session の場合、サーバークラスでは、コネクションごとにそれぞれインスタンスが作られます。Invocation に変更すると、ステートレスのサーバークラスを利用することになります。例えば CGIWeb サーバープリケーション（これもステートレスで、リクエストごとに読み込んだり、読み込まなかったりします）を配布したい場合には有効です。LifeCycle を Server に変更すると、これは単一のサーバークラスインスタンスがすべての受信した接続とリクエストにシェアされることになります。これは、例えばリクエストの数を「カウント」したいときには有用といえますが、（複数のリクエストを受け、そして一斉に処理することを期待しているときには）スレッドに関する問題が起こらないように確認しておかなければなりません。

TDSServerClass コンポーネントには、４つのイベント OnCreateInstance、OnDestroyInstance、OnGetClass、OnPrepare があります（これらはインスタンスが作成されるか、明示的に破棄されたときに利用されなくなります）。OnPrepare イベントハンドラは、サーバーメソッドの準備をするために利用できます。

Delphi 2009 または Delphi 2010 を使用するときには、手動で TDSServerClass コンポーネントをコンテナに設置しなければなりません。どのクラスをサーバーからクライアントへ「リモート」するのかを指定する必要があるため、OnGetClass イベントを実行しなければなりません。しかし、Delphi 2010 の DataSnap ウィザードでは、OnGetClass イベントハンドラの実行を、以下のとおり自動的に行うようになりました。

```
procedure TServerContainer1.DSServerClass1GetClass(
  DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
```

```
begin
  PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
end;
```

これは自動生成されたコードであって、生成された ServerMethodsUnit1 ユニットのファイル名を変更し、ServerMethodsUnitDemo.pas として保存したときには若干の修正が必要であることに注意してください。

2.1.1.1.3. TDSTCPServerTransport

TDSTCPServerTransport コンポーネントは、DataSnap サーバーと DataSnap クライアント間の通信を担当し、TCP/IP を通信プロトコルとして使用しています。

TDSTCPserverTransport コンポーネントには、5 つの重要なプロパティ BufferKBSize、Filters（Delphi 2010 で新しく追加されました）、MaxThreads、PoolSize、Port、Server があります。

BufferKBSize プロパティは、通信のためのバッファのサイズを指定し、デフォルトでは 32（KB）に設定されています。Filters プロパティには、トランスポートフィルターのコレクションが含まれます。これについては、第 4 章で触れることにします。MaxThreads プロパティは、スレッドの最大数を定義するために使用できます（デフォルトでは、最大数も限界も制限しない 0 に設定されています）。PoolSize は、接続をプールしておけるようにします、そして Port プロパティは、サーバーがクライアントと接続するために用いている TCP/IP ポートをコントロールするために使用できます（もしここでこれを変更したら、後で DataSnap クライアントでも変更する必要があります）。

Server プロパティは TDSServer コンポーネントを指しています。DSTCPServerTransport コンポーネントにイベントはありません。

2.1.1.1.4. TDSHTTPService

TDSHTTPService コンポーネントは HTTP プロトコルを使用した DataSnap サーバーと DataSnap クライアント間の通信を担当します。

TDSHTTPService コンポーネントには、（Name と Tag を除き）10 のプロパティがあります。Active、AuthenticationManager、DSHostname、DSPort、Filters HttpPort、Name、RESTContext、Server、そしてリードオンリーの ServerSoftware プロパティです。

Active プロパティは、DSHTTPService が受信リクエストを受けた場合に指定します。設計時には True に設定することができますが、これは DataSnap Server アプリケーションが実行時にスタートすることを妨げることになります（1 つのアクティブ DSHTTPService コンポーネントしか同じポートを受けることができず、そして設計時にはすでに 1 つがアクティブになっているため、他のコンポーネントを実行時にスタートさせることができないのです）。TDSHTTPService をスタートさせる最善の方法は TServerContainer の OnCreate イベントハンドラを Active から True に設定することです:

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
  DSHTTPService1.Active := True;
end;
```

AuthenticationManager プロパティは HTTP 認証を処理するマネージャーコンポーネントを定義するのに使用され、この例ではすでに TDSHTTPServiceAuthenticationManager コンポーネントを指しています。このコンポーネントについての詳細は 2 章の 2.1.1.1.5.で説明します。

DSHostname と DSPort プロパティはサーバーの接続を定義するために使用されますが、Server プロパティが設定されていない場合のみ使用されます。たいていの場合には、単に Server プロパティを TDSServer コンポーネントへ代わりに接続するだけです。

Filters プロパティには、トランスポートフィルターのコレクションを指定できます。これについては第 4 章で説明します。

HttpPort プロパティは、受信した接続をどの DSHTTPService コンポーネントが受けるのかを定義するために使用されます。デフォルトでは、ポートの値は 80 に設定されています（Internet Information Services IIS のような）。Web サーバーがインストールされているマシンを使用して開発をする場合には、Web サーバーがポート 80 を使用していて、TDSHTTPService コンポーネントがポート 80 でリストを開始することができず、変更しなければならないことに注意してください。アプリケーションを実行しようとしたときにはエラーが表示されますが、このメッセージはすぐには意味がわからないかもしれません。

RESTContext プロパティは REST サービスとして DataSnap Server を呼び出すことのできる REST コンテキストの URL を定義します。デフォルトでは RESTContext は“rest”に設定されており、第 6 章で REST、JSON とクライアントコールバックメソッドのデモンストレーションを行いますが、これはサーバーを http://localhost/datasnap/rest/... として呼び出すことができることを意味しています。

最後に、Server プロパティは同じコンテナ上の TDSServer コンポーネントを指していなければなりません。もし Server プロパティに接続したくなければ、TCP を使っている DataSnap Server に DSHostname と DSPort プロパティを使って接続することができます。Server プロパティが設定されているとき、DSHostname と DSPort プロパティは無視されます。

TDSHTTPService コンポーネントには、次の 5 つのイベントがあります。REST 関連のものが 4 つと Trace イベントが 1 つです。REST イベントについては第 6 章で詳細に説明します。

OnTrace イベントは、DSHTTPService コンポーネントへの呼び出しをトレースするために使用することができます、例えば以下のようになります:

```
procedure TServerContainer1.DSHTTPService1Trace(Sender: TObject;
  AContext: TDSHTTPContext; ARequest: TDSHTTPRequest;
  AResponse: TDSHTTPResponse);
begin
  LogInfo('HTTP Trace ' + AContext.ToString);
  LogInfo('  ' + ARequest.Document);
  LogInfo('  ' + AResponse.ResponseText);
end;
```

HTTP Trace インフォメーションは、クライアントが実際にサーバーに接続するために HTTP を使用しているときにのみ表示されることに注意してください（デフォルト TCP/IP プロトコルの代わりではありません）。

トレース出力の例は次のようになります:

```
17:05:55.398 HTTP Trace TDSHTTPContextIndy
17:05:55.400   /datasnap/tunnel
17:05:55.403   OK
```

これは AContext が TDSHTTPContextIndy に設定されていることを意味しており、ARequest は /datasnap/tunnel に設定され、そして AResponse は OK に設定されます。

2.1.1.1.5. TDSHTTPServiceAuthenticationManager

TDSHTTPServiceAuthenticationManager コンポーネントは、HTTP 通信プロトコルのための Authentication チェックボックスがチェックされていたときに Server Container に配置されることになります。もちろん手動でサーバーコンテナに配置することもできます。

TDSHTTPServiceAuthenticationManager コンポーネントは TDSHTTPService コンポーネントの AuthenticationManager プロパティに接続されなければなりません。

TDSHTTPServiceAuthenticationManager コンポーネントには、OnHTTPAuthenticate というイベントが 1 つあります。これは、DataSnap クライアントからサーバーへ供給される HTTP Authentication 情報をチェックするために使用できます。

```
procedure TServerContainer1.DSHTTPServiceAuthenticationManager1HTTPAuthenticate(
  Sender: TObject; const Protocol, Context, User, Password: string;
  var valid: Boolean);
begin
  if (User = 'Bob') and (Password = 'Swart') then
    valid := True
  else
    valid := False
end;
```

例えばハッシュ値を持ったバージョンのパスワードを使ったデータベース内の検索で、この検証ルーチンを拡張しなければならないことは明らかです。HTTPS を利用するにはクライアントからサーバーへユーザー名と（ハッシュ値を持った）パスワード情報が送られる場所である HTTP Authentication が使われるべきであり、そのため個人的には Embarcadero が現在の HTTP と TCP/IP プロトコルのリストに HTTPS プロトコルを加えることを期待しています。

HTTPS は接続が安全で、データパケットが暗号化されていることを確実にするので、Sniffer データを使用する人々は、あなたのユーザー名と（ハッシュ値を持つ）パスワード情報には手を触れることができません。お使いの ISP または Web マスターに、あなたのドメインが HTTPS を使用できるかどうか確認してください。これは強く推奨します（サンプルでは、https://www.bobswart.nl を使用しています）。

現実に DataSnap サーバーアプリケーションに用いられるその他のよい方法は、HTTP Authentication（アテンプト）をログファイルに書き込むことです。ログファイルには、プロトコルとコンテキスト情報も書き込みます。これによって誰がログインしているのが、誰がログインしようとしているのか（例えばあなたの DataSnap サーバーにアクセスしようと試みる人々の偽のログインの試み）がわかります。

2.1.1.2. ServerMethodsUnitDemo

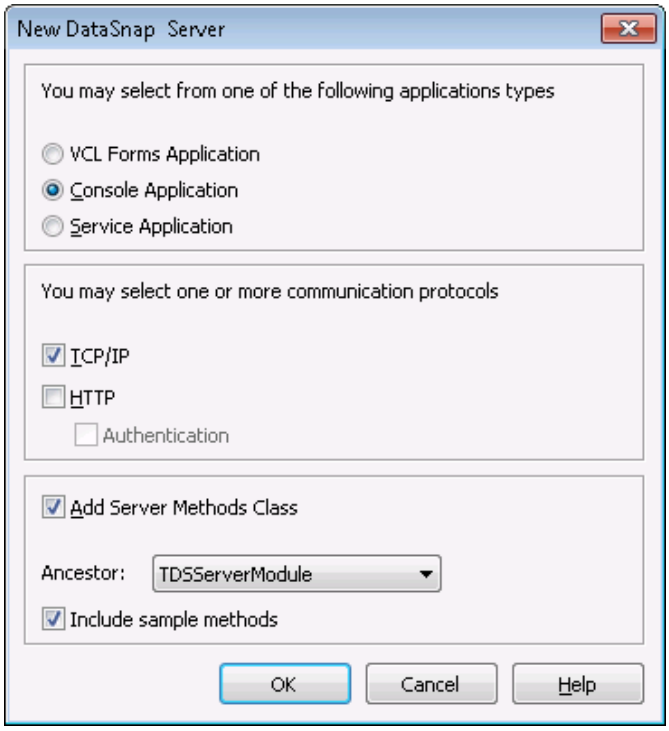
これまで ServerContainerUnitDemo.pas ユニットについて分析をしてきましたが、新しい DataSnap サーバーアプリケーションの他の重要なユニット:ServerMethodsUnitDemo.pas ユニットについて切り替えましょう。新規 DataSnap サーバーダイアログでは、上位クラスとして使用するために TDSServerModule クラスを指定しましたが、TServerMethods1 型は TDSServerModule を継承しています （これは TDSServerModuleBase から派生しており、また同じく TProviderDataModule から派生していて、Destroy デストラクタと BeforeDestruction プロシージャを加えています）。TProviderDataModule は通常の TDataModule を継承していて、プロバイダーと作業をする能力も追加しています（詳細については後ほど説明します）。

TServerMethods1 の上位クラスのひとつが TDataModule であるために、デザインタブはデータモジュール用のデザイナーエリアを表示します。つまり、データアクセスコントロールのようなビジュアル化されていないコントロールはどれでもここに配置することができます。データアクセスコンポーネントは第 3 章で配置しますが、ここではデザインエリアを空にしておいて、TServerMethods1 クラスにメソッドを追加するだけにしておきます。

もしプロジェクトグループに、さらにターゲット（DataSnap コンソールアプリケーションや DataSnap Windows サービスアプリケーション）を追加したくない場合には、Server Methods を実行する第 2 章の 2.1.4 まで進んでください。

2.1.2. マルチターゲットプロジェクトグループ – コンソール

プロジェクトグループに戻って、2 つめのターゲットを追加します。今回は、コンソールアプリケーションです。プロジェクトグループノードを右クリックし、［新規プロジェクトの追加］を選択してください。オブジェクトリポジトリで、再び DataSnap Server カテゴリーに行き、DataSnap Server アイコンをダブルクリックしください。今回は、新規 DataSnap サーバーダイアログでコンソールアプリケーションターゲットを選択します。



その他のオプションについては、何を選択しても問題がないことに注意してください。DataSnap サーバープロジェクトから、とにかく ServerContainerUnitDemo.pas と ServerMethodsUnitDemo.pas を再利用します。

［OK］をクリックして、新しいプロジェクトを生成します。再び Project1.dproj が生成され、加えて ServerContainerUnit2.pas と ServerMethodsUnit2.pas も生成されます。ServerMethodsUnit2.pas ユニットを右クリックし、新しいプロジェクトから削除します。ServerContainerUnit2.pas は、メソッ

ドをコピーするためにとっておきます。プロジェクトを DataSnapConsoleServer.dproj として保存します。

ServerContainerUnitDemo.pas と新しく生成したユニット、ServerContainerUnit2.pas（コンソールアプリケーション）の内容を比較すると、後者には RunDSServer と呼ばれるグローバルプロシージャーが含まれていることに気づくでしょう。このグローバルプロシージャーはコンソールアプリケーションにのみ有用なので、主要なブロックを開始する前に、これを DataSnapConsoleServer.dproj プロジェクトのソースコード内部にコピーアンドペーストしておかなければなりません。

ErrorInsight はいくつかの問題について警告しますが、DataSnapConsoleServer.dpr の uses 節に Windows ユニットを加えることで解決できます。DataSnapConsoleServer は次のようになります。

```
program DataSnapConsoleServer;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Windows,
  ServerContainerUnit2 in 'ServerContainerUnit2.pas'
    {ServerContainer2: TDataModule};

procedure RunDSServer;
var
  LModule: TServerContainer2;
  LInputRecord: TInputRecord;
  LEvent: DWord;
  LHandle: THandle;
begin
  Writeln(Format('Starting %s', [TServerContainer2.ClassName]));
  LModule := TServerContainer2.Create(nil);
  try
    LModule.DSServer1.Start;
    try
      Writeln('Press ESC to stop the server');
      LHandle := GetStdHandle(STD_INPUT_HANDLE);
      while True do
      begin
        Win32Check(ReadConsoleInput(LHandle, LInputRecord, 1, LEvent));
        if (LInputRecord.EventType = KEY_EVENT) and
        LInputRecord.Event.KeyEvent.bKeyDown and
        (LInputRecord.Event.KeyEvent.wVirtualKeyCode = VK_ESCAPE) then
          break;
      end;
    finally
      LModule.DSServer1.Stop;
    end;
  finally
    LModule.Free;
  end;
end;


begin
  try
    RunDSServer;
  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end
end.
```

これから行う 3 つの操作によって、残念ながら、ErrorInsight には再び問題が表示されます。DataSnapConsoleServer プロジェクトは、まだ ServerContainerUnit2.pas を使用しているので、そのユニットを削除します（右クリックで［プロジェクトから削除］を選択し、次に DataSnapConsoleServer.exe ノードを右クリックし［追加］を選択。ServerContainerUnitDemo.pas と ServerMethodsUnitDemo.pas ユニットを選択して、プロジェクトに追加します）。

その結果、すべての TServerContainer2 のリファレンスは構文エラーとして警告されます。ServerMethodsUnitDemo.pas は、TServerContainer1 型を定義しなければならないので、問題を修正します。ソースコード内の TServerContainer2 を DataSnapConsoleServer から TServerContainer1 へ名前を変更します（全部で 3 箇所あります）。

これで、新しい DataSnapConsoleServer.dproj プロジェクトと、元の DataSnapServer.dproj プロジェクトの両方をコンパイルすることができるようになります。2 つのプロジェクトターゲットの間で、ServerContainerUnitDemo.pas と同様に ServerMethodsUnitDemo.pas ユニットを共有しています。

2.1.3. マルチターゲットプロジェクトグループ – WINDOWS サービス

プロジェクトグループの DataSnap VCL フォームサーバーと DataSnap コンソールサーバープロジェクトに加えなければいけないターゲットが、あと 1 つ残されています。DataSnap Windows サービスアプリケーションです。このターゲットを加えるには、プロジェクトグループノードを右クリックして、［新規プロジェクトの追加］を選択し、オブジェクトリポジトリから［新規 DataSnap Server アイコン］を再びダブルクリックします。今回はサービスアプリケーションを選択し、すべての通信プロトコルオプションを選択します（TCP/IP、HTTP、HTTP Authentication）。すぐにわかるように、サービスアプリケーションのサーバーコンテナは VCL フォームやコンソールアプリケーション内部のサーバーコンテナとは若干違っています、そのためすべての通信プロトコルオプションを確実に選択してください。

再び新規 DataSnap サーバーダイアログから、新しいプロジェクトを作成し、加えて ServerContainerUnit と ServerMethodsUnit も作成します。ServerMethodsUnit は以前に見たものと同じなのでプロジェクトから削除して、VCL フォームとコンソール DataSnap アプリケーションとで共有する ServerMethodsUnitDemo.pas 内に置き換えることができます。

しかし、新しい ServerCotainerUnit1.pas ユニットは異なります。TDSServer に配置する TDataModule を使用するかわりに、TDSServerClass とトランスポートコンポーネント、コンポーネントを含む TService から派生したクラスがあります。TService からの派生したメソッドとは別に、4 つの特別なメソッドが追加されています。Stop、Pause、Continue というサービスを操作するものと、イベントを実行する Interrogate です。

```
type
  TServerContainer3 = class(TService)
    DSServer1: TDSServer;
    DSTCPServerTransport1: TDSTCPServerTransport;
    DSHTTPService1: TDSHTTPService;
    DSHTTPServiceAuthenticationManager1: TDSHTTPServiceAuthenticationManager;
    DSServerClass1: TDSServerClass;
    procedure DSServerClass1GetClass(DSServerClass: TDSServerClass;
      var PersistentClass: TPersistentClass);
    procedure ServiceStart(Sender: TService; var Started: Boolean);
  private
    { Private declarations }
  protected
    function DoStop: Boolean; override;
    function DoPause: Boolean; override;
```

```
    function DoContinue: Boolean; override;
    procedure DoInterrogate; override;
  public
    function GetServiceController: TServiceController; override;
  end;
```

言い換えれば、Windows サービスプロジェクトの元の ServerContainerUnitDemo.pas ユニットを共有することはできないので、ServerContainerUnit1.pas を ServerContainerUnitServiceDemo.pas へ名前を変更しなくてはいけないということです。そしてこのときには、プロジェクト自身も DataSnapServiceServer.dproj という名称で保存しましょう。

ServerContainerUnitServiceDemo 内の古い ServerMethodsUnit2.pas ユニットの参照を修正する必要があるので、ServerMethodsUnitDemo.pas へ変更します。これで、少なくとも 3 つすべてのターゲットで、同じ Server Methods ユニットを共有していることになります。

2.1.4. SERVER METHODS

1 つまたはそれ以上の DataSnap サーバープロジェクトがあるとき、すべてのプロジェクトは同じ ServerMethodsUnitDemo ユニットを共有しています。次は Server Methods をいくつか詳細に検討してみましょう。

前に述べたように、TServerMethod1 クラスは、DataSnap クライアントを宣言している（RTTI を使用している）メソッドを宣言する DataSnap サーバーオブジェクトです。「Include サンプルメソッド」オプションをチェックしていると、TServerMethods1 クラスにはすでにサンプルメソッドが 1 つ存在しているでしょう。それは EchoString 関数です。DataSnap サーバーマシンから現在の時間を返すために書き換えます。これには、2 つのパブリックメソッドを含む TServerMethods1 の定義を以下のように修正します。

```
type
  TServerMethods1 = class(TDSServerModule)
  private
    { Private declarations }
  public
    { Public declarations }
    function EchoString(Value: string): string;
    function ServerTime: TDateTime;
  end;
```

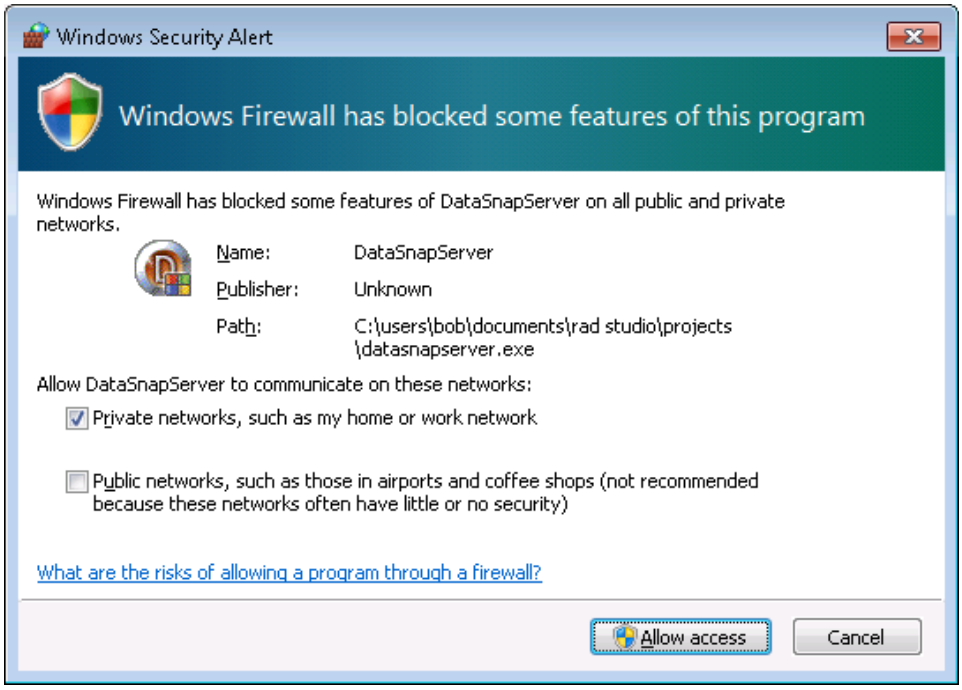
ServerTime メソッドの実行は非常に簡単で、ほとんど EchoString メソッドの例と同じくらい短いものになります。

```
function TServerMethods1.EchoString(Value: string): string;
begin
  Result := Value;
end;

function TServerMethods1.ServerTime: TDateTime;
begin
  Result := Now
end;
```

サーバーアプリケーションをコンパイルし、実行できるようになりました。もし複数のプロジェクトターゲットを作成していたら、現時点でテストする一番簡単な方法は DataSnapServer 実行ファイルです。お使いの Windows のバージョンとセキュリティ設定のレベルによりますが、Windows ファイ

ヤーウォールが DataSnapServer アプリケーションのいくつかの機能をブロックしたことを知らせる Windows セキュリティ警告が表示されます。

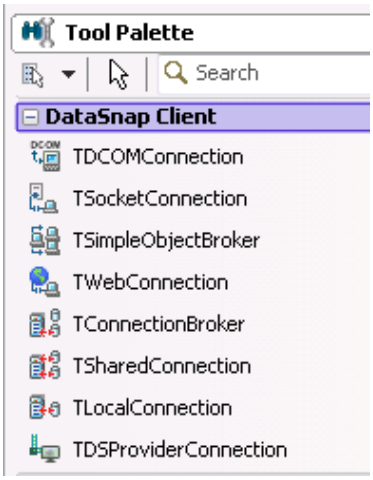


これは DataSnapServer アプリケーションがこの時点で TCP/IP と HTTP を通じて受信リクエストを積極的に受けているという事実によるものです。トロイの木馬のように、受信リクエストを受けたい唯一のものなので、［アクセスを許可する］をクリックし、DataSnapServer の起動を継続してください。

2.2. DATASNAP クライアント

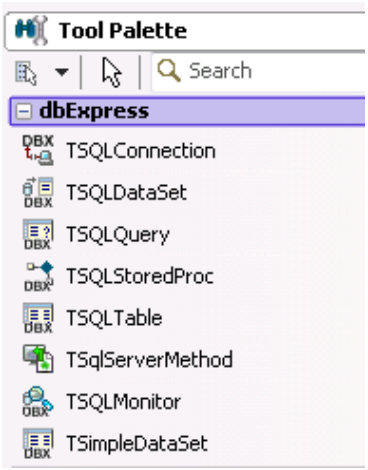
最初の DataSnap サーバーのデモが実行されリクエストを受けつけられるように、クライアントを構築します。この章では、クライアントからサーバーへ接続する方法、生成したサーバークラスによってメソッドをどのようにインポートするのかを説明します。

まず、DataSnap サーバーが起動されていて、稼働中であることを確認してください。DataSnap クライアントアプリケーションを作成するためには、サーバーが稼動してなければなりません。DataSnap サーバープロジェクトと DataSnap クライアントプロジェクトを設計時に簡単に切り替えられるように、DataSnap クライアントプロジェクトを同じプロジェクトグループに加えます。どんなプロジェクトも DataSnap クライアントにできますが、このデモでは、VCL フォームアプリケーションを使用して、ClientForm.pas のフォームと一緒に DataSnapClient.dpr として保存します。ツールパレットの DataSnap Server カテゴリーの 6 つの DataSnap（Server）コンポーネントがあるのに対し、DataSnap Client カテゴリーには、今すぐに使う必要のあるコンポーネントはありません。



DataSnap Client カテゴリーのコンポーネントは、まだ使用することができる「古い」DataSnap コンポーネントですが、DataSnap を使用する方法としては、推奨されていませんが、1 つ例外があります。新しい TDSProviderConnection コンポーネントは、「古い」DataSnap サーバーと新しい DataSnap クライアント（これは第 3 章の 3.2 で言及する予定です）を接続するために使用できます。

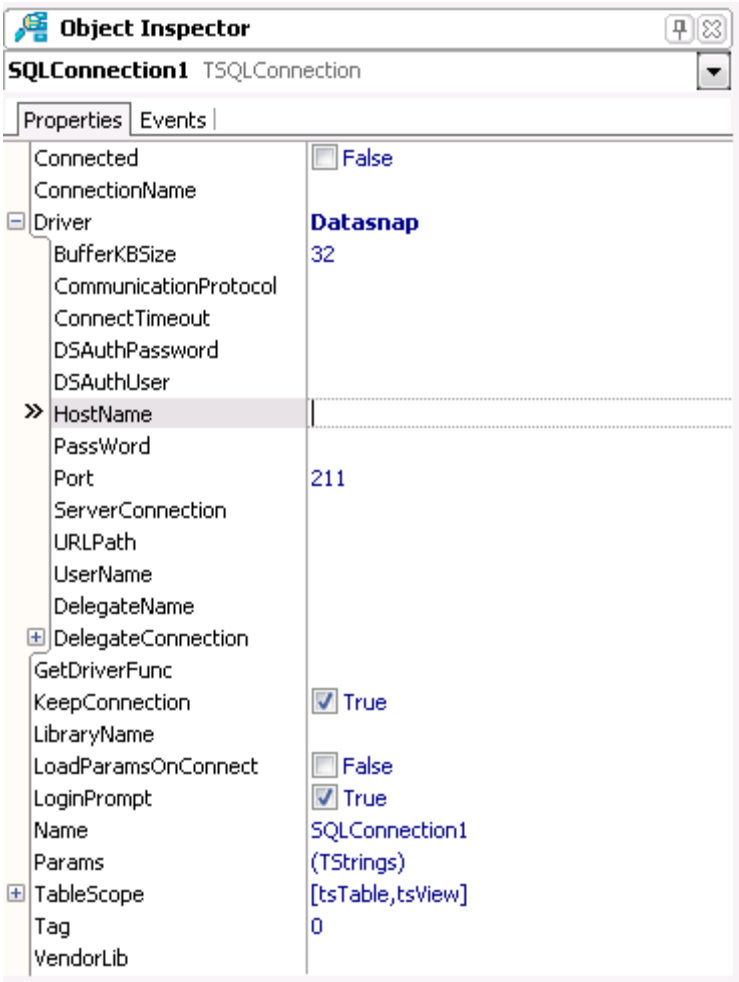
DataSnap Client カテゴリーのかわりに dbExpress カテゴリーを見なくてはいけません。ここには、TSqlServerMethod と呼ばれる新しいコンポーネントがあります（次の画面ショットではこの新しいコンポーネントを簡単に識別できますが、これは、従来の dbExpress コンポーネントに付加されている接頭語「TSQL」のかわりに「TSql」が付加されているからだということに注意してください）。



TSqlServerMethod コンポーネントは DataSnap サーバーからリモートメソッドを呼び出すために使用できます。その場合、最初に DataSnap Server に接続しなければなりません。

TSQLConnection コンポーネントを使用すること、古い方の TxxxConnection コンポーネントを使用しないことで、接続が可能になります。これを容易に進めるために、TSQLConnection コンポーネントの Driver ドロップダウンリストには、DataSnap と呼ばれる新しいドライバ名があります。

では、SQLConnection コンポーネントを ClientForm 上に配置し、その Driver プロパティを DataSnap に設定します。その結果、Driver プロパティには＋印が表示され（プロパティ名の左側）、すべてのサブプロパティを表示するようにプロパティを展開することができます。



デフォルトでは、CommunicationProtocol プロパティを空のままにしていると、TSQLConnection は TCP/IP をプロトコルとして使用することに注意してください（特定の 211 ポート経由です）。

BufferKBSize は 32（KB）に設定されており、ポートはデフォルトでは 211 に設定されています（サーバーサイド）。実際のところ、私はポートを 211 からそれ以外のものへと、いつも変更しています（サーバーサイドとクライアントサイドの両方について）。それは、211 ポートは DataSnap サーバーへ接続するポートとしては、少し有名すぎるからです。

HostName、UserName と Password は DataSnap サーバーへの接続に使われます。ローカルでのテストのために、HostName に localhost を設定することができますが、一般的にはホスト名、DNS 名または、IP アドレスを直接、このプロパティのための値として登録します。

TSQLConnection コンポーネントの LoginPrompt プロパティを、ログインダイアログを表示しないようにするため、False に設定することを忘れないでください。

一度 TSQLConnection Driver プロパティを指定すると、DataSnap サーバーに接続（を試行）するために Connected プロパティを True に設定することができます。接続を行うには、DataSnap サーバーがサーバーサイドで稼働中でなければならないことに注意してください。

2.2.1. DATASNAP クライアントクラス

接続が検証できたら、TSQLConnection コンポーネントを右クリックして、DataSnap クライアントクラスの生成のクラスオプションを選択します。これは、デフォルトでは TServerMethods1Client というクラスと Unit1 と呼ばれる新しいユニットを作成します（実際の DataSnap サーバーメソッドクラスの名前は、サーバーサイドのものに「Client」という部分が追加されたものです）。このユニットを ServerMethodsClient.pas として保存します。

生成された TServerMethods1Client クラスの定義は、次のようになります。

```
type
  TServerMethods1Client = class
  private
    FDBXConnection: TDBXConnection;
    FInstanceOwner: Boolean;
    FEchoStringCommand: TDBXCommand;
    FServerTimeCommand: TDBXCommand;
  public
    constructor Create(ADBXConnection: TDBXConnection); overload;
    constructor Create(ADBXConnection: TDBXConnection;
      AInstanceOwner: Boolean); overload;
    destructor Destroy; override;
    function EchoString(Value: string): string;
    function ServerTime: TDateTime;
  end;
```

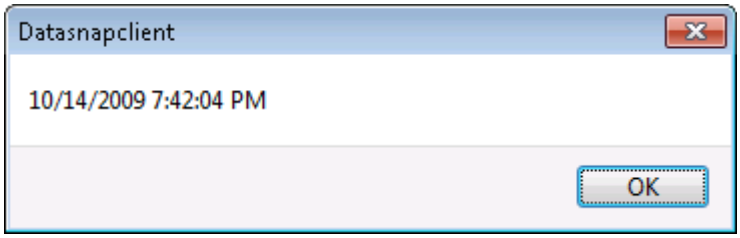
このように、TServerMethods1Client クラスには 2 つのコンストラクタがあり、1 つのデストラクタと、DataSnap Server サイドで定義された 2 つのサーバーメソッドが存在します。

これらのメソッドを使用するために、ServerMethodsClient ユニットを ClientForm の uses 節に追加し、TButton コンポーネントをクライアントフォームに追加して、OnClick イベントハンドラに以下のコードを記述します。

```
procedure TForm2.Button1Click(Sender: TObject);
var
  Server: TServerMethods1Client;
begin
  Server := TServerMethods1Client.Create(SQLConnection1.DBXConnection);
  try
    ShowMessage(DateTimeToStr(Server.ServerTime))
  finally
    Server.Free
  end;
end;
```

このコードは TServerMethods1Client のインスタンスを作成して、ServerTime サーバーメソッドを呼び出し、最終的に DataSnap Server のプロキシを再び破棄します。

このボタンをクリックすると、期待した通り、サーバーの現在時刻の値を示すメッセージボックスが表示されます。

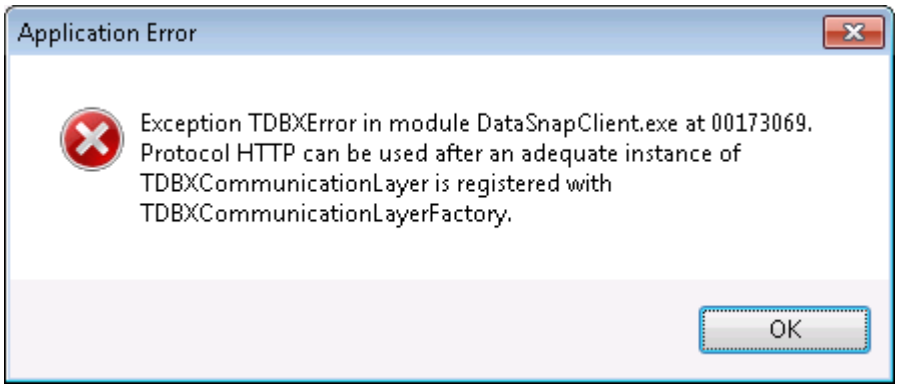


演習として、ご自身で EchoString メソッドのテストを同様の方法で行えるよう、このままにしておきます。

2.2.1.1. HTTP 通信プロトコル

TSQLConnection コンポーネントのデフォルトの CommunicationProtocol である TCP/IP について触れたことに注意してください。いいかえれば、（第 2 章の 2.1.1.1.4 で定義した）いずれの HTTP トレースメッセージも見られないということです。しかしながら、通信プロトコルを変更することは難しくありません。つまり、TSQLConnection の Driver プロパティの CommunicationProtocol サブプロパティに「HTTP」を値として入力するのです。211 が TCP/IP に使用されているため、Port プロパティを修正する必要があるということを意味していることにも注意してください。ServerContainer の TDSHTTPService コンポーネントのために、指定したポートと同じ値を指定してください。

これらの変更を行い DataSnap クライアントを再度実行すると、次のようなアプリケーションエラーが表示されるはずです。

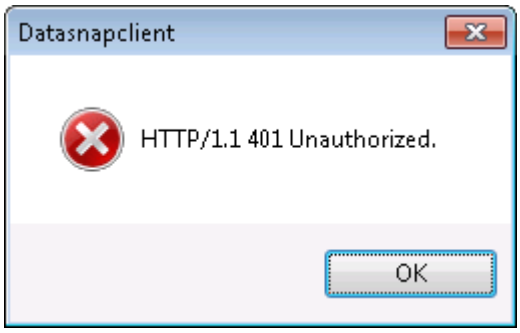


このエラーの解決方法は、DSHTTPLayer ユニットを DataSnap クライアントの uses 節（例えば ClientForm に）に加えることです。

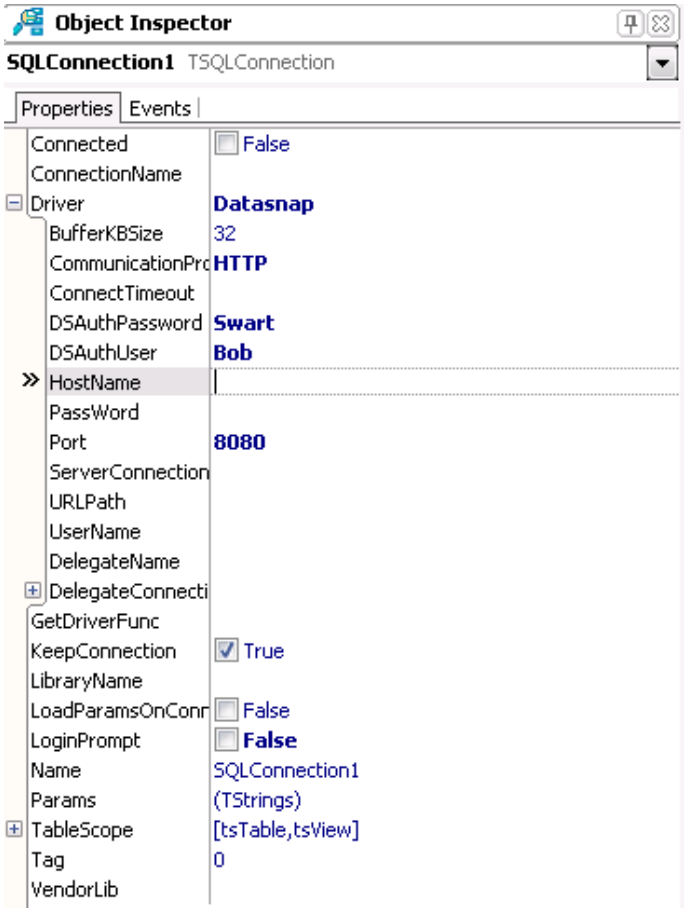
2.2.1.2. HTTP 認証

HTTP 通信プロトコルを使用する利点の 1 つに、HTTP 認証を含めることができる点が挙げられます。これは 2 章の 2.1.1.1.5.で説明したように、TDSHTTPServiceAuthenticationManager コンポーネントによって DataSnap サーバー側でサポートされています。

OnHTTPAuthenticate イベントハンドラが実行されると同時に、HTTP 認証のチェックがここで行われるので、TDSHTTPServiceAuthenticationManager からアクセスが許可されるように、正しい情報を送信していることを確認する必要があります。さもなければ、次のような HTTP/1.1 401 Unauthorized エラーが表示されます。



HTTP 認証のユーザーネームとパスワード情報を DataSnap クライアントから DataSnap サーバーへ送信するためには、TDSHTTPServiceAuthentication コンポーネントをより厳密に、（クライアントのフォーム上の）DSAuthUser と TSQLConnection コンポーネントの DSAuthPassword プロパティに記入する必要があります。



DataSnap サーバーを同じローカルマシン上で稼動させていない場合は、HostName の値を指定しなければならないことに注意してください。

2.3. DataSnap サーバーの配布

この例は、サーバーとクライアントの両方が同じローカルマシンで実行されているときには正しく動作しますが、現実には、DataSnap サーバーはサーバーマシンで実行され、ネットを通じて 1 つまたはそれ以上のクライアントが接続しなければならないことになります。DataSnap サーバーアプリケーションは Delphi がインストールされていないマシンに配布されることが一般的です。この場合、DataSnap サーバーを、ランタイムパッケージを使用せずにコンパイルすることを考慮しても良いですが、そうすると大きな実行ファイルになります。

現在はどんなデータアクセスコンポーネントも使用していないため、今回はデータベースドライバの追加や外部 DLL については必要がありません。

2.3.1. DataSnap クライアントの配布

DataSnap サーバーアプリケーションとは異なるマシンで DataSnap クライアントアプリケーションが実行されると仮定した場合、クライアントがサーバーに接続できることを確認しなくてはいけません。このことを実際に確かめるために、クライアントフォームの TSQLConnection コンポーネントは CommunicationProtocol と Driver プロパティの Port サブプロパティの値を指定するだけでなく、HostName の値も指定する必要があります。IP アドレスをここで割り当てることを避けるために、論理的な DNS 名をどこでも可能な場所で使用します。例えば私の DataSnap サーバーでは、www.bobswart.nl という HostName の値を使用できます(実際に使用されるプロトコルは CommunicationProtocol で指定されているはずですので、「http://」プレフィクスを指定する必要がないことに注意してください)。

3. DataSnap とデータベース

Delphi 2010 DataSnap フレームワークでは、簡単なサーバーメソッドを構築するために使用するだけでなく、データベースアクセスをサーバーに追加し、多層データベースアプリケーションを構築することができます。DataSnap サーバーがデータベースに接続するのに対し、DataSnap クライアントはシンクライアントまたはスマートクライアントなので、（クライアントサイドで）データベースドライバを必要としません。

ここまでに構築してきた DataSnap のサンプルでは、SQLConnection コンポーネントと生成されたクライアントクラスを直接使用してきましたが、2 つの新しい DataSnap コンポーネント、TSqlServerMethod と TDSProviderConnection も使用できます。

はじめに、サーバーが実際に TDataSet を見られることを確認しておいたほうがいいでしょう。そのため、ServerMethodsUnitDemo に戻り、TSQLConnection コンポーネントをデータモジュールに配置します。TSQLConnection コンポーネントを好きな DBMS とテーブルに接続します）この例では BlackfishSQL の Employees データベースから Employees テーブルに接続します）。

この動作を行うために、TSQLConnection コンポーネントを ServerMethods1 デザイナーエリアに設置し（ServerMethodsUnitDemo.pas ユニットの中です）、TSQLConnection コンポーネントの Driver プロパティを BlackfishSQL に設定します。もしくは Driver プロパティを開き、Database プロパティに employee.jds へのパスとして次のどちらかを設定します。

Windows XP：

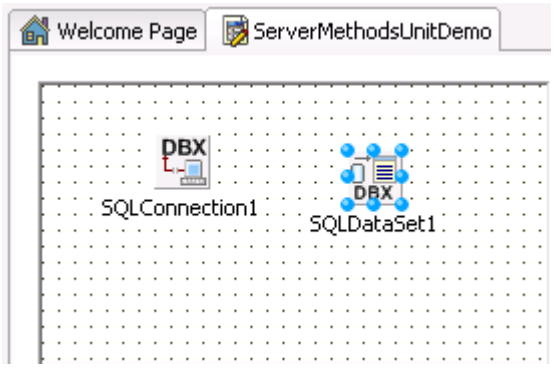
C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL

Windows Vista または Windows 7：

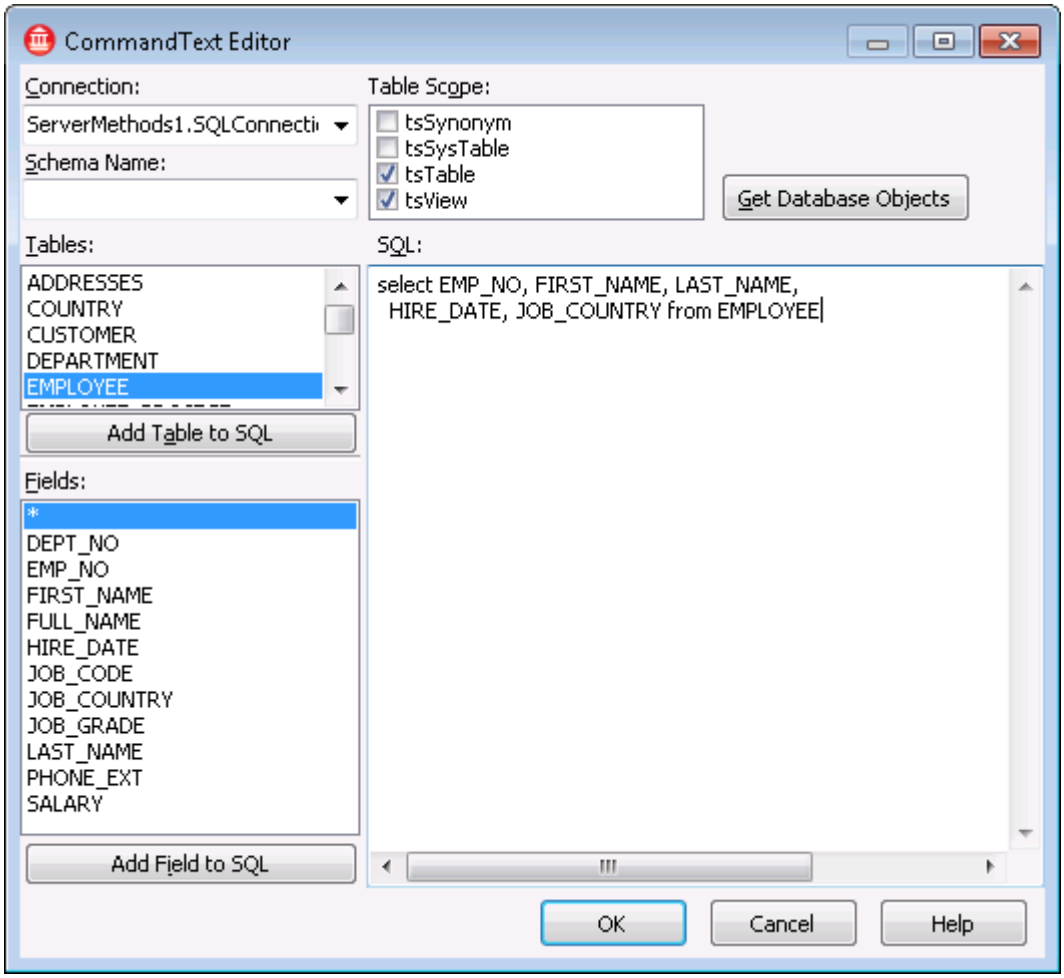
C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL

TSQLConnection コンポーネントの LoginPrompt プロパティが False に設定されていることを確認したら、Employee.jds データベースへの接続をオープンできるかを確認するために Connected プロパティを True に設定してみてください。

次に、TSQLDataSet コンポーネントを TSQLConnection コンポーネントの隣に配置し、そのSQLConnection プロパティで TSQLConnection コンポーネントを指定します。



CommandType は ctQuery に設定したままにします。そして、CommandText プロパティをダブルクリックして、SQL クエリーを入力します。




```
SELECT EMP_NO, FIRST_NAME, LAST_NAME, HIRE_DATE, JOB_COUNTRY FROM EMPLOYEE
```

TSQLConnection コンポーネントでは、設計時に自身の LoginPrompt と Connected プロパティを False に設定しているので、TSQLDataSet の Active プロパティも False に設定されていることを確認してください。

ServerMethodsUnitDemo ユニットの TServerMethods1 クラスに、TSQLDataSet コンポーネントのコンテンツを返す public 関数を追加する必要があります。

```
type
  TServerMethods1 = class(TDSServerModule)
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
  private
    { Private declarations }
  public
    { Public declarations }
    function EchoString(Value: string): string;
    function ServerTime: TDateTime;
    function GetEmployees: TDataSet;
  end;
```

TServerMethod1 で定義されているように、新しい関数は GetEmployees という名称で、その実装は次のようになります。

```
function TServerMethods1.GetEmployees: TDataSet;
begin
  SQLDataSet1.Open; // make sure data can be retrieved
  Result := SQLDataSet1
end;
```

サーバーを再コンパイルし、動作しているを再度確認します。もし DataSnap サーバーを閉じているのに、再コンパイルすることができない場合、まだ DataSnap サーバープロジェクトが動作中であることが考えられます。

3.1. TSQLSERVERMETHOD

クライアントアプリケーションに戻ります。TSQLConnection コンポーネントは DataSnap サーバーに接続していてはいけません（もしまだ接続していたら、そのプロセスが使用中ということになるので DataSnap サーバーを再コンパイルできない理由になっていたかもしれません）。Connected を True に再度設定し、サーバーからの情報をリフレッシュすれば、その後クライアントクラスを生成することができます（再度マウスの右ボタンを使います）。

以前の ServerMethodsClient ユニットに上書きするためには、プロジェクトから ServerMethodsClient の以前のバージョンを削除して［DataSnap クライアントクラスの生成］を再度選択することをおすすめします。もし生成した新しいユニットを ServerMethodsClient.pas ファイルに再び保存したら、以前のクライアントコードを変更する必要はありません。［DataSnap クライアントクラスの生成］メニューを再度実行したら、生成された TServerMethods1Client クラスが GetEmployees メソッドが追加されて拡張されます（まだ利用可能な ServerTime と EchoString 関数も同様にリストされています）。

```
type
  TServerMethods1Client = class
  private
    FDBXConnection: TDBXConnection;
    FInstanceOwner: Boolean;
    FEchoStringCommand: TDBXCommand;
    FServerTimeCommand: TDBXCommand;
    FGetEmployeesCommand: TDBXCommand;
  public
    constructor Create(ADBXConnection: TDBXConnection); overload;
    constructor Create(ADBXConnection: TDBXConnection;
      AInstanceOwner: Boolean); overload;
    destructor Destroy; override;
    function EchoString(Value: string): string;
    function ServerTime: TDateTime;
    function GetEmployees: TDataSet;
  end;
```

TDataSet を取り出すために GetEmployees メソッドを使用するには、ツールパレットの dbExpress カテゴリーから TSqlServerMethod コンポーネントを選択します。クライアントフォーム上に TSqlServerMethod を設置して、SQLConnection プロパティを SQLConnection1 に接続します。そして呼び出し可能なすべてのメソッドを表示するために ServerMethodName ドロップダウンリストを開きます。たくさんの DSAdmin メソッドがあり（これは TDSServer コンポーネントの HideDSAdmin プロパティが True に設定されていることによって無効になっている可能性があります）、3 つの DSMetadata メソッドと 7 つの TServerMethods.AS_xxx メソッドが後に続き（オリジ

ナルの IAppServer インターフェイスを提供しています）、最後に TServerMethods1 EchoString、ServerTime と GetEmployees メソッドとなります。

この例では、ServerMethodName プロパティ値として TServerMethods1.GetEmployees を選択します。ServerMethodName の値による SqlServerMethod の結果は、内部の Employees レコードと一緒のデータセットとなります（少なくとも SQL 文からの結果と一緒です）。

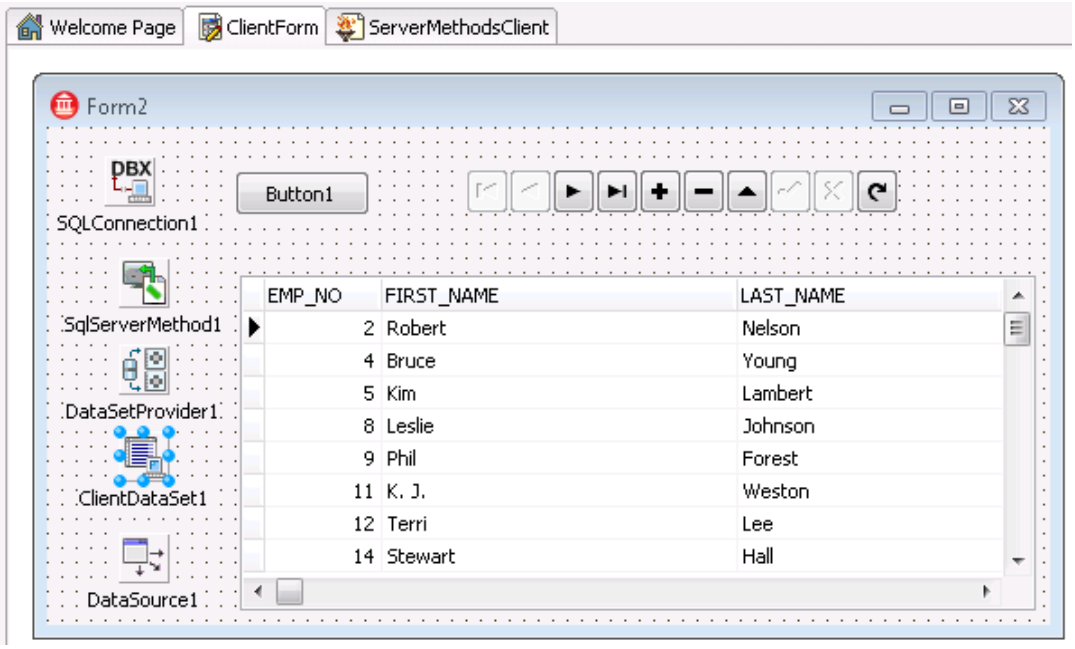
ここでは、通常の TDataSetProvider、TClientDataSet と TDataSource のリンクを、例えば TDBGrid にデータを表示できるようにするために使用します。

TDataSetProvider をクライアントフォームに設置して、DataSet プロパティを SqlServerMethod コンポーネントに接続します。次に、TClientDataSet コンポーネントをクライアントフォームに設置して、ProviderName プロパティを DataSetProvider コンポーネントに接続します。RemoteServer プロパティはブランクのままにしておきます。これは「古い」DataSnap アプローチでのみ使用されるもので、新しい DataSnap アーキテクチャーで使用することはありません。

最後に TDataSource をクライアントフォームに設置して、DataSet プロパティを TClientDataSet コンポーネントと接続したら、TDBGrid と TDBNavigator コンポーネントを使用し、これらの DataSource プロパティと TDataSource コンポーネントを接続すれば、クライアントフォームでデータを見ることができるようになります。

設計時に接続を検証するには、ClientDataSet の Active プロパティを True に設定します。これは SqlServerMethod の Active プロパティを切り替えることになります（データを取り出すための短い間だけで、その後、Active プロパティは再び False に設定されます）。TSQLConnection コンポーネントの Connected プロパティを True に設定して、DataSnap クライアントとサーバー間の接続ができるようにします。

接続がアクティブのままですが、結果として TClientDataSet と TSQLConnection の両方がアクティブで、TDBGrid 内部のデータを表示しています:



これは読み取り専用でデータを見るための簡単な方法を提供しています。TSqlServerMethod は、TDataSetProvider-TClientDataSet の組み合わせに対して、DataSnap クライアントから DataSnap サ

ーバーにこの方法での更新を許可していないために、特別に「読み取り専用」という言葉を使用したことに注意してください。

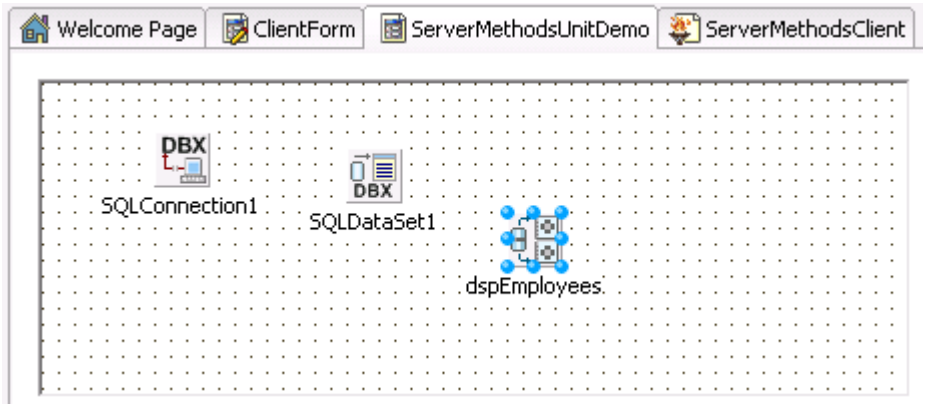
TSqlServerMethod は軽量で、読み取り専用データに接続するには便利な方法です。これは DataSnap サーバーから、だれにも修正できないデータを出力したいとしたら、現在のアーキテクチャーでは、TServerMethods1 クラスから TSQLDataSet の結果を出力するのがよい方法だということを意味しています。

しかし、時には更新を許可する必要があるので、そのような場合にはデータをエクスポーズするために違ったアプローチを使う必要があります。

3.2. TDSPROVIDERCONNECTION

更新を適用したいのであれば、例えばサーバーサイドで DataSetProvider を参照することができる TDSProviderConnection コンポーネントが必要です。そうするとデータを読み取るだけでなく、更新もできるようになります。

まず、サーバーサイドでサーバーデータモジュールを変更する必要があります。現在は TDataSet を返すための機能しか追加していないため、実際の TDataSetProvider を追加しなければならず、そして DataSnap サーバーから DataSnap クライアントに公開されることを確認してください。そして ServerMethodsUnitDemo ユニットへ戻り、TDataSetProvider コンポーネントを TSQLDataSet の隣に配置して、TSQLDataSet を TDataSetProvider の DataSet プロパティに設定します。TDataSetProvider コンポーネントの名前を変更しなければなりませんが、dspEmployees のように意味がある名前にするとよいでしょう。



こうして DataSnap サーバーが再コンパイルおよび再実行されたので、出力されたデータを変更するためにクライアントを修正することができます。

3.2.1. TDSPROVIDERCONNECTION クライアント

公開された TDataSetProvider コンポーネントを使用するように DataSnap クライアントを変更するには、クライアントフォームから TSqlServerMethod と TDataSetProvider コンポーネントを削除して、代わりに TDSProviderConnection コンポーネントを追加します。

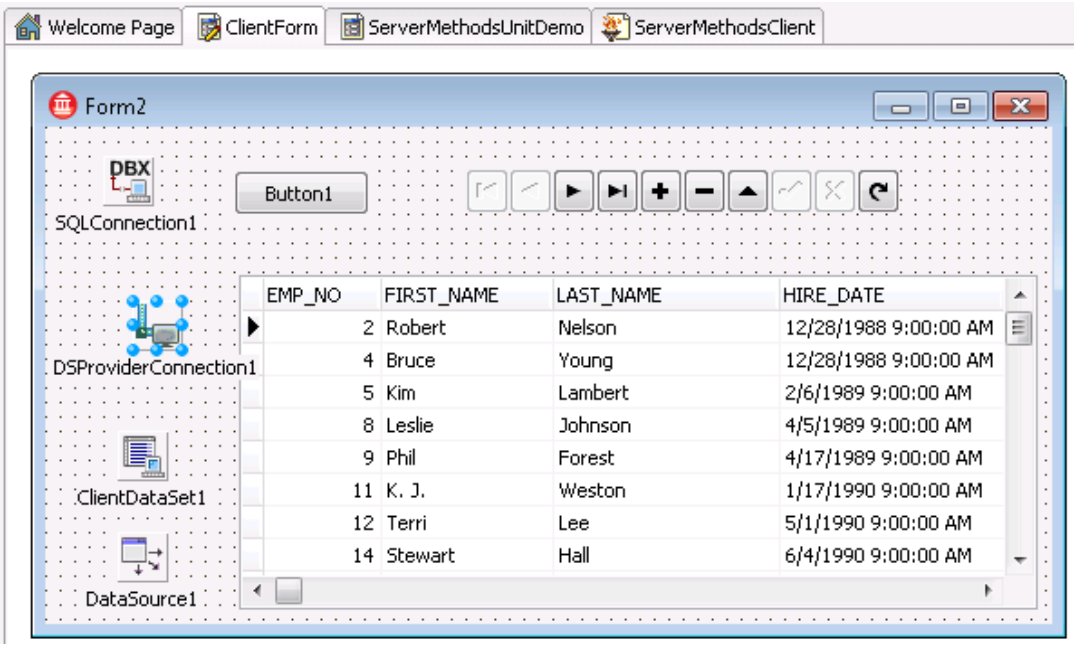
TDSProviderConnection の SQLConnection プロパティには、TSQLConnection コンポーネントを指定して DataSnap サーバーに接続しています。TDSProviderConnection の ServerClassName プロパティにも、値を入力しなければなりません。ここでまだドロップダウンリストが使用できないことは、

ちょっと残念です。現時点では、TDSServerModule の名前（今回の場合は TServerMethods1 でした）を手動で入力しなければなりません。

以前の例では、TClientDataSet は ProviderName を DataSetProvider1 に接続しているだけでした。しかし、TDSProviderConnection コンポーネントを使用して、最初に TDSProviderConnection コンポーネントの RemoteServer プロパティに TClientDataSet を割り当てなければいけません。それから ProviderName プロパティに、新しい値を選択します（これは DataSetProvider1 の値にまだ割り当てられていますが、今は dsEmployees を指していなければなりません。これは、DataSnap サーバーから公開された TDataSetProvider コンポーネントのより説明的な名称です）。

ProviderName プロパティのドロップダウンコンボボックスでは、dspEmployees を選択しておきます（ServerDataMod ユニットで公開された TDataSetProvider コンポーネントの名称です）。

そして、設計時にライブデータを表示するため、TClientDataSet コンポーネントの Active プロパティを再び True に設定します。



TClientDataSet コンポーネントの Active プロパティを設定すると、TSQLConnection コンポーネントの Connected プロパティも True に設定されます。設計時にこれら 2 つのプロパティをクライアントフォーム上で True に設定したままにしておくのは、よくありません。はじめに、Delphi IDE で DataSnap クライアントプロジェクトを開き、DataSnap サーバーへの接続を試行します。これは、サーバーが起動および動作中でないと失敗します。次に、アプリケーションを開始し、動作時に利用できる接続がないと、アプリケーションは例外を発生させます。これはローカルデータを利用したアプリケーション使用時に、接続がない場合の無駄なレンダリングを防いでくれます。

よりよい解決策は、いくつかのメニューオプションやボタンで、TSQLConnection コンポーネントを明示的に接続するか、TClientDataSet をアクティベートすることです。ここではユーザー名／パスワード情報を含む点も重要ですが、これについては後ほど触れます。今のところは TSQLConnection コンポーネントの Connected プロパティと同様に、TClientDataSet の Active プロパティが False に設定されていることを確認してください。そして、クライアントフォームのボタンを配置し、OnClick イベントハンドラで TClientDataSet をオープンする処理を記述します。

```
procedure TForm2.Button2Click(Sender: TObject);
begin
  ClientDataSet1.Open;
end;
```

実際に、（クライアントサイドで）データを変更するいくつかのコードを追加しましょう。そして、その変更をサーバーに送信します。

3.2.2 データベースの更新

クライアントで行った変更をサーバーに送信するには、実際、自動と手動の 2 種類の方法があります。両方とも最終的には同じメソッドを呼び出しますが、起動が自動であるか、ユーザーによって行われるかのどちらかで、それぞれにメリットとデメリットがあります。

自動アプローチでは、データの変更を行うメソッドである TClientDataSet の OnAfterInsert、OnAfterPost、そして OnAfterDelete イベントを利用することができます。それらのイベントハンドラ内で（単一の実装として共有することも可能です）TClientDataSet の ApplyUpdates メソッドを呼び出し、そしてサーバーに、「Delta」とも呼ばれる差分情報を送信し、データベースに反映されます。

```
procedure TForm2.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  ClientDataSet1.ApplyUpdates(0);
end;
```

もし、更新の間に（レコードが見つからないというような）何らかの問題が発生したら、TClientDataSet の OnReconcileError イベントからフィードバックを得ることができます。これについては、3 章の 3.2.3.で詳細に説明します。

更新を DataSnap サーバーに送信する手動アプローチでも、TClientDataSet の ApplyUpdates メソッドを使用しますが、この場合には、OnAfterInsert、OnAfterPost や OnAfterDelete イベントハンドラではなく、ユーザーがサーバーに更新を送信できるボタンを、クライアントフォーム上に追加します。

```
procedure TForm2.btnUpdateClick(Sender: TObject);
  ClientDataSet1.ApplyUpdates(0);
end;
```

ApplyUpdates を自動的に呼び出すときのメリットは、ユーザーがどんな変更であってもサーバーに送信するのを絶対に「忘れる」ことがないだろうという点です。また、デメリットとしては取り消しができません。一度送信したら、そのデータはサーバーに適用されます。これに対し、手動アプローチを使用した場合には、すべての変更がクライアントサイドで、まず TClientDataSet コンポーネントのメモリーの内部で保存されます。これにより、ユーザーは、変更箇所を一部取り消すこともできるようになります。最後に行った変更でも、特定のレコードに対する変更でも、保留中の更新はすべて取り消すことができます。このアプローチでは、ユーザーが更新を行う準備ができたときに、［update］ボタンをクリックして ApplyUpdates メソッドを呼び出すことができます。危険があるとすれば、ユーザーがアップデートボタンをクリックするのを「忘れる」ことなので、フォームにチェックを追加するか、TClientDataSet に変更が残されたままであるときにはアプリケーションを閉じることができないようにするのがいいでしょう。後者は TClientDataSet の ChangeCount プロパティでチェックできます。

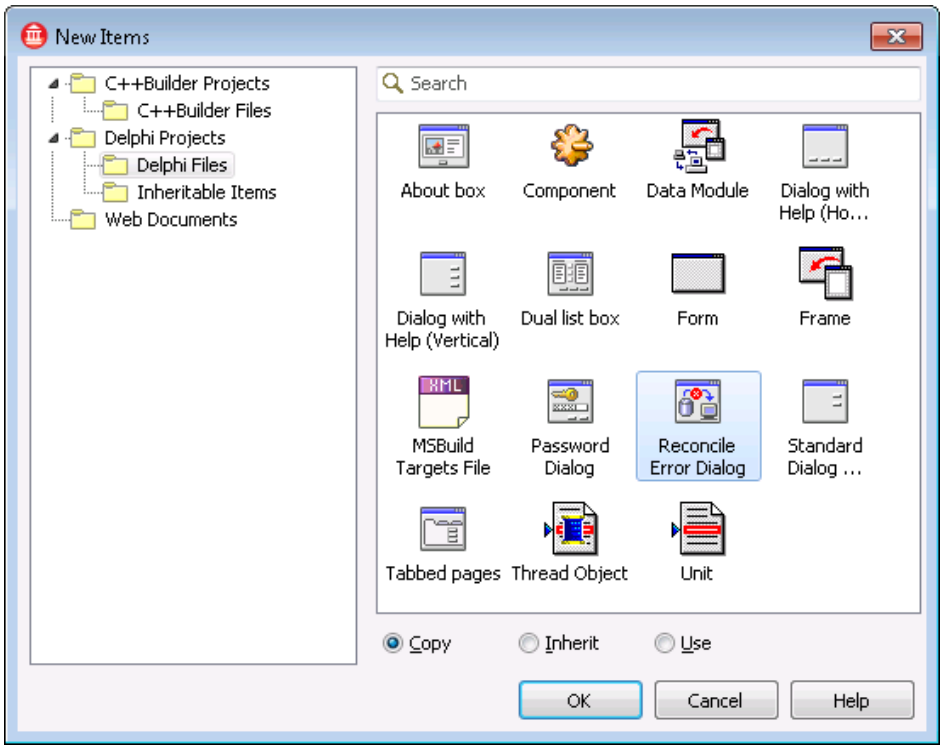
3.2.3. 調停エラー

TClientDataSet コンポーネントの ApplyUpdates メソッドには、引数が 1 つあります。これは、エラー発生時に更新の適用の継続を許可するエラーの最大数です。

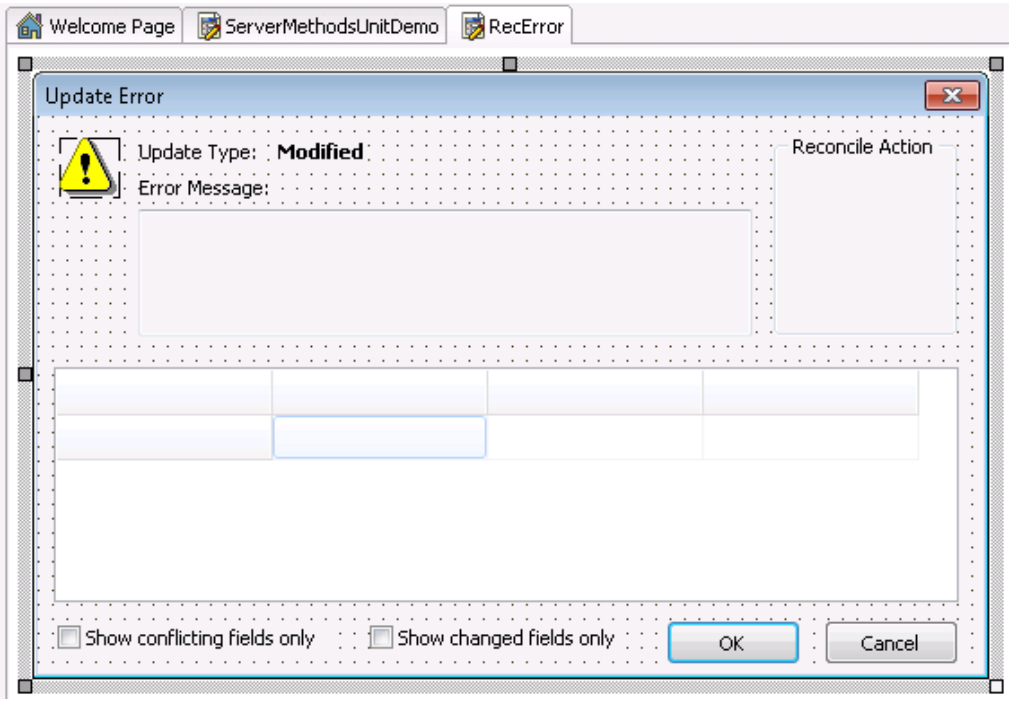
2 人のユーザーが DataSnap サーバーに接続し、Employees データを取得して両者とも最初のレコードに変更を加えた場合を考えてみましょう。これまで構築してきた内容に従えば、両方のクライアントは、その TClientDataSet コンポーネントの ApplyUpdates メソッドを使用して DataSnap サーバーに更新されたレコードを返すことができます。両者とも ApplyUpdates の引数「MaxErrors」として 0 を渡すと、更新を行おうとする 2 番目の処理は中止されるでしょう。2 番目のクライアントでは、更新を停止するまでの回数として、エラー／コンフリクトの数を 0 以上の固定値を渡すことができます。しかし、たとえ 2 番目のクライアントが-1（これは、何回エラーが起きても更新を継続することを示します）を引数として渡したとしても、前のクライアントによって変更されたレコードの更新は行われません。言い換えればすでに更新されたレコードとフィールドの更新を行うためには、いくつかの調整を行う必要があるのです。

幸運にも、Delphi にはこの目的のために特別に作られた便利なダイアログがあります。そしてエラーの調整をする必要があるときにはいつでも、このダイアログを DataSnap クライアントアプリケーションに追加することを考慮するべきです（あるいは、最低でもこの問題への対応を自ら記述しなければなりません）。

Delphi で用意されている機能を使用するには、［ファイル｜新規｜その他］メニューを選択して、オブジェクトリポジトリの「Delphi ファイル」サブカテゴリーを表示して、「エラー調停ダイアログ」アイコンを選択します。



このアイコンを選択して［OK］をクリックすると、新たに RecError.pas ユニットが DataSnapClient プロジェクトに追加されます。このユニットには、データベース更新エラーを解決する Update Error ダイアログの定義と実装が含まれます。



ReconcileErrorForm のインスタンスは、必要なときに動的に作成されます。それでは、いつ、どのようにしてこの特別な ReconcileErrorForm を使用するのでしょうか？実は、実際にはとてもシンプルです。更新が成功しなかったレコードごとに（理由はなんであれ）、TClientDataSet コンポーネントの OnReconcileError イベントハンドラが呼び出されます。この TClientDataSet のイベントハンドラは次のように定義されます。

```
procedure TForm2.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
```

これは 4 つの引数を持つイベントハンドラです: 最初にエラーを引き起こした TClientDataSet コンポーネント、2 番目にエラー状態の原因についてのメッセージを含む特定の ReconcileError、3 番目にエラーを生成した UpdateKind（insert、delete、または modify）、そして最後の引数は、あなたがとるべき Action です。

Action として、次の列挙値 (順番はこれらの実際の列挙値のとおりです)を返します:

- raSkip – このレコードを更新しません。しかし変更ログの中の適用されていない変更は残します。次回再試行する準備ができています。
- raAbort – すべてのハンドリングの調整を停止します。OnReconcileError イベントハンドラにこれ以上レコードが送られることはありません。
- raMerge - 更新されたレコードを(リモート) データベースの現在のレコードにマージします。クライアント側で変更した（リモート）フィールドのみ変更します。。
- raCorrect - 更新されたレコードを、イベントハンドラ（または ReconcileErrorDialog 内で）作成したレコードの訂正された値で置き換えます。これはオプションであり、ユーザーの介入が必要（例:タイプによる入力）となります。
- raCancel – このレコード内部のすべての変更を取り消し、持っていたオリジナルの（ローカル）レコードへと戻します。

- raRefresh - このレコード内部のすべての変更を取り消しますが、現在の（リモート）データベースから（持っていたオリジナルのローカルレコードからではなく）レコードの値をリロードします。

ReconcileErrorForm のメリットは、これらすべてを自分で配慮しなくてもよいということです。しなくてはいけない事は 2 つだけです。最初に DataSnap Client メインフォームの定義内に、ReconcileErrorDialog ユニットを含めるようにします。DataSnap Client Form をクリックし、［ファイル｜ユニットの使用］メニューを選択して「ユニットの使用」ダイアログを表示します。現在のユニットを Client Form とすると、「ユニットの使用」ダイアログでは、その他に使用できるユニットのみをリストします。ここでは ReconcileErrorDialog が表示されるので、これを選択して［OK］をクリックします。

次にする必要があるのは、ReconcileErrorDialog ユニット（ClientMainForm のインポートリストにたった今加えました）の、HandleReconcileError 関数を呼び出すために OnReconcileError イベントハンドラのコードに 1 行記述することです。HandleReconcileError 関数は OnReconcileError イベントハンドラとして 4 つの引数を持っています（もちろん偶然の一致ではありません）。そのため、ここで行う作業は、一方から他方への引数を受け渡すだけに過ぎません。TClientDataSet コンポーネントの OnReconcileError イベントハンドラは、次のように記述します。

```
procedure TFrmClient.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action := HandleReconcileError(DataSet, UpdateKind, E)
end;
```

3.2.4. 調停エラーのデモ

さて、ここで大きな疑問が沸き起こります。これらは実際にはどのように動くのでしょうか？ これをテストするためには、明示的に 2 つ（またはそれ以上）の DataSnap クライアントアプリケーションを同時に実行しなければなりません。現在の DataSnap クライアントと DataSnap サーバーアプリケーションを使った完全な検証を行うため、以下の手順を実行します。

- DataSnap サーバーアプリケーションを起動します。
- 最初の DataSnap クライアントを起動し、［Connect］ボタンをクリックします。
- 2 つめの DataSnap クライアントを起動し、［Connect］ボタンをクリックします。すでに実行中の同じ DataSnap サーバーからデータが取得できます。
- 最初の DataSnap クライアントを使用して、最初のレコードの「FirstName」フィールドを変更します。
- 2 つめの DataSnap クライアントを使用して、やはり最初のレコードの「FirstName」フィールドを変更します。
- 最初の DataSnap クライアントの［Apply Updates］ボタンをクリックします。すべての更新が何の問題もなく適用されます。
- 2 つめの DataSnap クライアントの［Apply Updates］ボタンをクリックします。今回は 1 つまたはそれ以上のエラーが発生します。なぜなら（最初の DataSnap クライアントによって）最初のレコードの「FirstName」フィールドの値が変更されているからです。そして、このレコードや他に競合しているレコードがあれば、それに対して OnReconcileError イベントハンドラが呼び出されます。

- このとき Update Error ダイアログで、skip、abort、merge、correct、cancel、refresh といったアクションの動作を確認するテストを実行できます。Skip と Cancel の違い、Correct と Refresh、Merge の間の違いには特に注意してください。

「Skip」は次のレコードへと移動し、リクエストされた更新を（とりあえず）スキップします。適用された変更は変更ログに残されます。「Cancel」もまたリクエストされた更新をスキップしますが、（同じ更新パケット内での）これ以上の更新をキャンセルします。現在の更新リクエストは両方のケースでスキップされますが、Skip は他の更新リクエストについては続行し、Cancel はすべての ApplyUpdates リクエストをキャンセルします。

「Refresh」はレコードに行ったすべての更新を単純に忘れて、サーバーデータベースからの現在の値にリフレッシュします。「Merge」はサーバーのレコードと更新されたレコードとの統合を試み、サーバーレコードの内部に変更を設置します。Refresh と Merge はこれ以上の変更リクエストを進めることはしないので、レコードは Refresh と Merge の後は同期しています（一方変更リクエストは、Skip または Cancel の後でも再度実行させることができます）。

「Correct」は、もっとも強力なオプションで、イベントハンドラ内部の更新レコードを実際にカスタマイズするオプションを与えてくれます。このためには自分でいくつかのコードを書くか、ダイアログに値を入力する必要があります。

3.3. DATASNAP「データベース」の配布

データベースを使った DataSnap サーバーの配布は、私たちがスタートしたシンプルな DataSnap サーバーよりも少し複雑になります。クライアントアプリケーションは変更がありません。これはいまだにシンクライアント／スマートクライアントアプリケーションなので、MidasLib ユニットを uses 節に加えれば、スタンドアロンの実行可能ファイルとして配布できます。

DataSnap サーバーでは、データベースドライバの配布も必要です。ドライバもファイルも選択したデータベースに依存します。DBX4 を使用するときには、TSQLConnection コンポーネント同様に以下のディレクトリの dbxconnections.ini と dbxdrivers.ini ファイルを確実にチェックしてください。

Windows XP：

C:\Documents and Settings\All Users\ Documents\RAD Studio\dbExpress\7.0 directory

Windows Vista および Windows 7：

C:\Users\Public\Documents\RAD Studio\dbExpress\7.0 directory

dbxdrivers.ini ファイルでは、指定したドライバ用に、DriverPackageLoader と MetaDataPackageLoader を指定します（たいていは同じパッケージを指しています）。BlackfishSQL では、DBXClientDriver140.bpl になり、Blackfish 本体とともに配布する必要があります。BlackfishSQL の配布についての詳細情報については、RAD Studio\7.0 ディレクトリの deploy_ja.htm ファイルをご覧ください。

3.4. 既存のリモートデータモジュールの再利用

既存の TRemoteDataModule クラスがある場合には、現在でも新しい DataSnap と組み合わせて使用することができます。しかし、特に COM 関連のいくつかの機能を、サーバーから削除する必要があります。

まず、移行させたい既存の DataSnap サーバーアプリケーションがあり、それがリモートデータモジュールでない場合には、/unregister コマンドラインオプションを用いてコマンドラインから DataSnap サーバーの登録を解除する必要があります。もしスタート直後にこれを行わなければ（後でプロジェクトのバックアップを修復しない限りは）、レジストリからリモートデータモジュールの登録を抹消することはできないでしょう。

リモートデータモジュールユニットでは、initialization 部からコードを取り除く必要があります。Delphi 2007 以前と 2009 以降とで互換性を保持した状態でユニットを保存した場合は、{$IFDEF}内に、次のようなコードを記述します。

```
{$IF CompilerVersion >= 20}
initialization
  TComponentFactory.Create(ComServer, TRemoteDataModule2010,
    Class_RemoteDataModule2010, ciMultiInstance, tmApartment);
{$IFEND}
end.
```

また、UpdateRegistry ルーチンをプロジェクトから削除するか、同じように{$IFDEF}内に移動します。

```
  {$IF CompilerVersion >= 20}
    class procedure UpdateRegistry(Register: Boolean;
      const ClassID, ProgID: string); override;
  {$IFEND}
```

このプロジェクトを COM レスの DataSnap サーバーに変換する上での最も重要な変更には、タイプライブラリ（もしくは .ridl ファイル）やタイプライブラリの重要なユニットの廃止が含まれます。これらは{IFDEF}には残すことができないため、もし Delphi 2007 以前（COM ベース）と Delphi 2009 以降（COM レス）のバージョンの DataSnap サーバーを保存する必要があるのであれば、ここでプロジェクトをコピーする必要があります。DataSnap サーバーアプリケーションの TDSServerClass コンポーネントを使用しなければならず、そしてたった今行ったように TRemoteDataModule クラスを返します。

最後に、TRemoteDataModule にすべてのカスタムメソッドが追加され、protected 部（デフォルトでは COM ベースの DataSnap）から public 部（これにより、メソッド情報は COM レスな DataSnap アーキテクチャーの中で生成されます）に移動されたことを確認します。

4. DataSnap フィルター

この章ではフィルターの働きについて、そして、（圧縮などの）既存のフィルターの使用法、また新しい DataSnap フィルターの作成方法について説明します。DataSnap フィルターは通信のバイトストリームを遮断する特別な DLL で、実際にフィルターのチェーンのひとつとして動作させることができます。そのため、例えば圧縮と暗号化、またはロギングと圧縮などを組み合わせて使用することができます。

DataSnap サーバーとクライアントがどのフィルターを使うかを指定する箇所は 2 つあります。サーバーでは、TDSTCPServerTransport コンポーネントの Filters プロパティ内でリストを指定します。そしてクライアントでは、DataSnap クライアントプロジェクトの uses 節の中の類似したフィルターリストを指定しなければなりません。それぞれの DataSnap フィルターを使用するには、自動的にそれ自身を登録しなければならないので、クライアントにはこれで十分です。

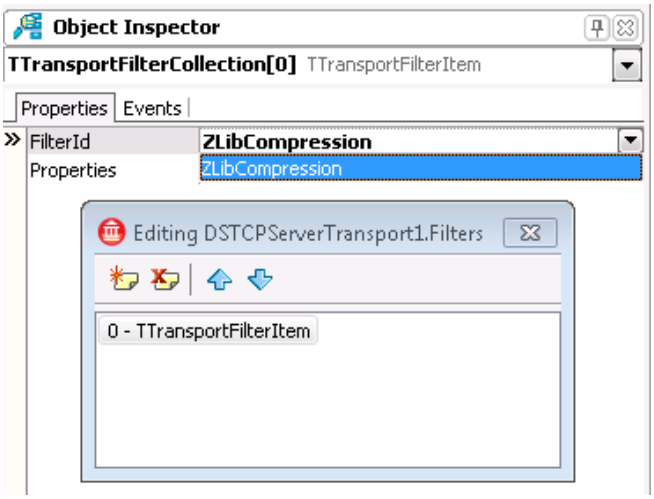
OnConnect イベントハンドラ内では、接続に使用されたフィルターの登録を検証できます。以下はその例です（情報をログファイルに書き込むために、カスタム関数 LogInfo を使用しています）。

```
procedure TServerContainer1.DSServer1Connect(
  DSConnectEventObject: TDSConnectEventObject);
var
  i: Integer;
begin
  LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
  for i:=0 to DSConnectEventObject.Transport.Filters.Count-1 do
    LogInfo('  Filter: ' +
      DSConnectEventObject.Transport.Filters.GetFilter(i).Id);
end;
```

4.1. ZLIBCOMPRESSION フィルター

例として、Delphi 2010 ですでに提供されている DataSnap サーバー とクライアントの間のデータストリームを圧縮するために用いることができる（逆もまた同様です）既存の DataSnap フィルターの検証をしてみましょう。ここでは、DbxCompressionFilter ユニットにある ZlibCompression フィルターについて説明をします。

TDSTCPServerTransport コンポーネント（TCP/IP 用）と DSHTTPService コンポーネント（HTTP 用）には、両方とも TTransportFiltersCollection を持つ Filters プロパティがあります。フィルターのコレクションを編集するには、Filters プロパティの省略アイコンをクリックします。このダイアログでは新しい TTransportFilterItem を追加することができ、その後、オブジェクトインスペクタを使って FilterId と他の任意のプロパティを設定します。Delphi 2010 には、すぐに利用可能な ZLibCompression フィルターが含まれており、FilterId としてここで設定することができます。



サーバーサイドの TDSTCPServerTransport コンポーネントの Filters プロパティとは別に、クライアントサイドでもこのフィルターを使用するように指定する必要があることに注意してください（出力されるリクエストの圧縮と入力されるレスポンスの解凍）。この指定は、DbxCompressionFilter ユニットを ClientForm の uses 節に加えるだけです。これにより、TTransportCompressionFilter が自動的に登録されるので、サーバーとの通信に使用されているかどうか確認してください。

DbxCompressionFilter ユニットを uses 節に追加しなかったら、実行しているクライアントは例外を生成し、*「Communication filter ZLibCompression is not registered. Filter class needs to be registered in order to communicate with the server」*というメッセージを表示します。

4.2. LOG フィルター

Delphi 2010 DataSnap は、自分でトランスポートフィルターを定義できるように開放されています。これを行うには、TransportFilter 型から新しいクラスを派生させます。この新しいクラスでは、ベースメソッドをオーバーライドして、それを実装できます。例えば、以下のように TLogFilter クラスを作成できます。

```
unit LogFilter;
interface
uses
  SysUtils, DBXPlatform, DBXTransport;

type
  TLogFilter = class(TTransportFilter)
  private
  protected
    function GetParameters: TDBXStringArray; override;
    function GetUserParameters: TDBXStringArray; override;
  public
    function  GetParameterValue(const  ParamName:  UnicodeString):  UnicodeString;
override;
    function SetParameterValue(const ParamName: UnicodeString;
      const ParamValue: UnicodeString): Boolean; override;
    constructor Create; override;
    destructor Destroy; override;
    function ProcessInput(const Data: TBytes): TBytes; override;
    function ProcessOutput(const Data: TBytes): TBytes; override;
    function Id: UnicodeString; override;
  end;

const
  LogFilterName = 'Log';
```

このクラスの実装は、ほとんどの場所が空のまま残されています。このログフィルターのただひとつの目的は、ProcessInput と ProcessOutput メソッドの間で送られたデータのログを取ることなので、他の大部分のメソッドについては空のままでかまいません。空ではないメソッドの実装は、次のとおりです。

```
function TLogFilter.SetParameterValue(const ParamName, ParamValue:
UnicodeString): Boolean;
begin
  Result := True;
end;

constructor TLogFilter.Create;
begin
  inherited Create;
end;

destructor TLogFilter.Destroy;
begin
  inherited Destroy;
end;

function TLogFilter.ProcessInput(const Data: TBytes): TBytes;
begin
  Result := Data; // log incoming data
end;

function TLogFilter.ProcessOutput(const Data: TBytes): TBytes;
begin
  Result := Data; // log outgoing data
end;

function TLogFilter.Id: UnicodeString;
begin
  Result := LogFilterName;
end;
```

最後に、DataSnap トランスポートフィルターの実装の重要な箇所は、initialization 部と finalization 部の登録を行っている箇所です。これによって DataSnap クライアントがトランスポートフィルターを「見つける」ことができ、必要なときに使用可能にしています。

```
initialization
  TTransportFilterFactory.RegisterFilter(LogFilterName, TLogFilter);
finalization
  TTransportFilterFactory.UnregisterFilter(LogFilterName);
end.
```

DataSnap サーバーでトランスポートフィルターを使用するには、上述したとおり、TDSTCPServerTransport または TDSHTTPService コンポーネントの Filter のリストにこれを追加しなくてはなりません。ZLibCompression フィルターは、設計時に既知の状態ですが、他の新しいフィルターについては（設計時パッケージの中に追加して、インストールしない限り）そうではありません。幸いなことに、実行時でもトランスポートフィルターを追加することができるので、ServerContainerUnitDemo の uses 節にフィルターユニットを追加して、以下の例のように、手動でフィルターリストにフィルターの名前を追加します。

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
  DSTCPServerTransport1.Filters.AddFilter(LogFilterName);
  DSHTTPService1.Filters.AddFilter(LogFilterName);
  DSHTTPService1.Active := True;
end;
```

これにより、サーバーが LogFilter を使用していることを確実にし、そしてクライアントの uses 節に LogFilter ユニットが追加された後で、クライアントがそれを自動的に使用するようになります。そうでなければ次のようなエラーメッセージが表示されるでしょう。

それぞれのアプリケーション、DataSnap サーバーとクライアントはそれぞれ自分のログファイルを持つため、同じロギングフィルターが使われていても、ParamStr(0)のようなどのターゲットが実際にログメッセージを作成しているかといった情報を付加する必要がないことに注意してください。

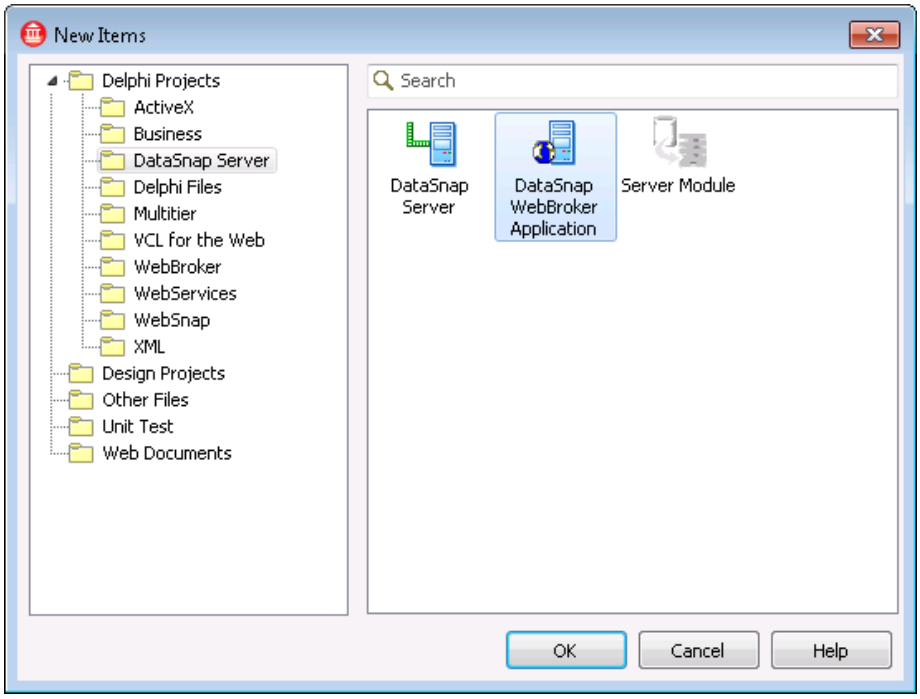
4.3. ENCRYPTION フィルター

第 4 章の 4.2 で簡単なフィルターの例を紹介したので、これを拡張して、より複雑な DataSnap フィルターを自作することがそれほど複雑なものではないことは明らかでしょう。実際すでに多くのサードパーティーによるフィルターを入手することができますし、Daniele Teti による「DataSnap Filters Compendium」には http://www.danieleteti.it/?p=168 からアクセスできます。DataSnap 2010 用の追加のフィルターは 9 つ以上ありますが、3 つのグループに分かれています。Hash グループは MD5、MD4、SHA1 と SHA512 をサポートし、Cipher グループは Blowfish、Rijndael、3TDES と 3DES をサポートし、Compress グループは LZO をサポートしています。すべてのソースコードは入手可能です。

5. DATASNAP WEB ターゲット

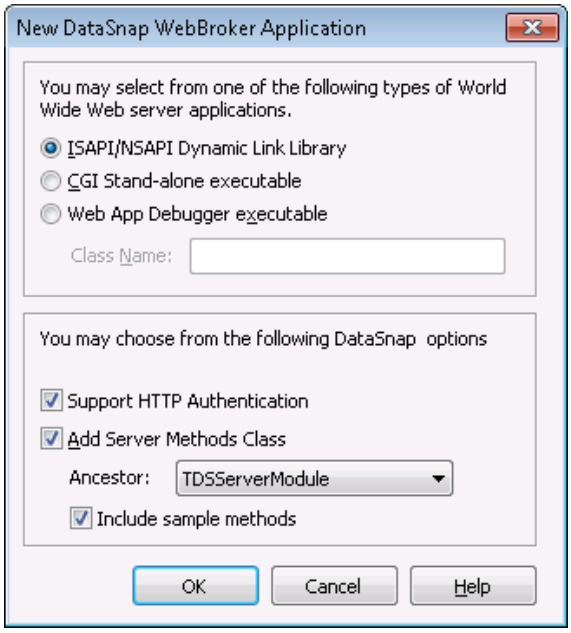
Windows ターゲットとは別に、ISAPI、CGI または Web アプリケーションデバッガターゲットを作成するウィザードもあります。初めに、これらのターゲットそれぞれのメリットについて説明し、DataSnap カスタムユニットを共有する 3 つのターゲットを、どのように単一のプロジェクトグループで作成するのかを紹介します。それは、単に同じ DataSnap サーバーオブジェクトのために、違うターゲットを作成する 3 つのプロジェクトを持つプロジェクトグループを作成することです。

ここまで構築してきた DataSnap サーバーアプリケーションが順調に動作したとしても、これらのサーバーアプリケーションを配布できないケースがあります。例えば、クライアントにサーバーへの接続を許可するための必要なポートをファイヤーウォール上に開くことができない、または許可されていない場合です。幸いなことに、このケースの起きるほとんどの状況では Web サーバーによって Web サイトがホスティングされているため、ポート 80 がたいていの場合（Web サーバー用に）開いています。そしてもし Microsoft Internet Information Services (IIS)が Web サーバーとして使用されていると仮定したときには、IIS 上に配布することが可能なプロジェクトを DataSnap WebBroker アプリケーションの新規作成ウィザードを使用して作成することができます。



DataSnap WebBroker アプリケーションウィザードは 3 つのオプションを提供していますが、そのうちの 1 つは実際には本当の WebBroker アプリケーションではなく、しかし単にデバッグ目的だけに使用される Web デバッグクライアントです。Web アプリケーションデバッガクライアントは、Web アプリケーションデバッガ（Delphi IDE のツールメニューから利用できます）を、Web アプリケーションデバッガクライアント DataSnap アプリケーションのデバッグをしている間のホストアプリケーションとして使用できる強力なクライアントです。CGI や ISAPI/NSAPI Web アプリケーションのデバッグは、容易ではありません。そのため、アプリケーション開発中は、Web アプリケーションデバッガが強力なオプションになります。

他の ISAPI/NSAPI DLL(Dynamic Link Library)と CGI 実行形式ターゲットは、DataSnap サーバープロジェクト用に選択できます。

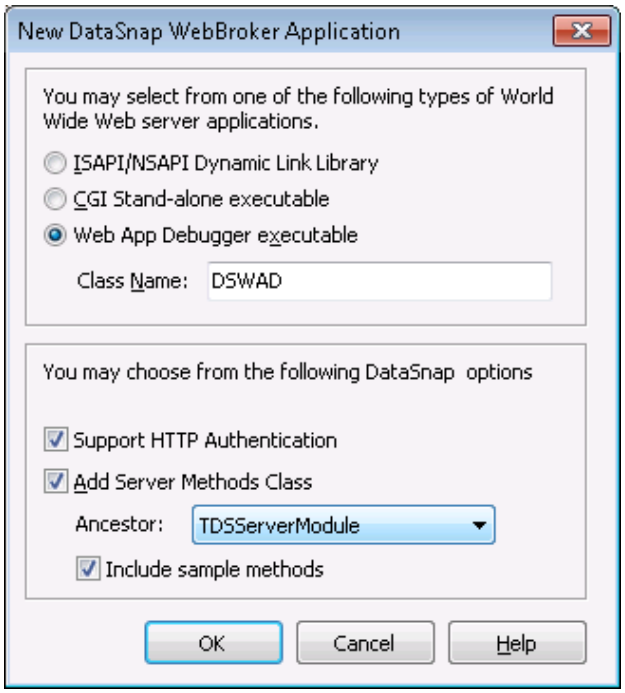


しかしながら CGI 実行形式を選択することは、あまりよい選択ではありません。なぜなら、この実行可能ファイルは受信したリクエストごとに、それぞれロードとアンロードを繰り返すからです。さらに、いくつかの作業を実行するためにデータベースに接続する時間も追加され、この種のアプリケーションではパフォーマンス劣化を引き起こすであろうと予想がつきます。ISAPI ターゲットを使用すれば、DLL がロードされるのは一度だけで、後に続くリクエストでは（他のユーザーからであっても）メモリー内にロードされたものが残されるため、さらなるパフォーマンスペナルティに悩まされることはありません。ISAPI DLL の主な不都合な点は、（Web サーバーへのアクセスに FTP しか利用できないと）置き換えが簡単ではないことです。しかし、このタスクを解決するのに十分な ISAPI Managers があります（詳細はお使いの Web ホストプロバイダーに相談してください）。

その他の ISAPI DLL ターゲットの不都合な面としては、デバッグが容易ではないことがあげられます。IIS をホストアプリケーションとしてロードしなければなりませんが、これは常に計画通りには動きません。しかしこの特定の問題は、Web アプリケーションデバッカ用実行形式の存在によって解決されます。両方とも、確実に同じ DataSnap カスタムメソッドとコードを使用している 2 つのプロジェクトを使えばよいだけです。これは最初のデモとしてはよいスタートなので、実行中のスケルトンを確認するために実際のテクニックを追加していきましょう。

5.1. WEB アプリケーションデバッカ用実行形式ターゲット

初めに「DataSnap WebBroker アプリケーションの新規作成」ウィザードを使用して、新しい Web App Debugger アプリケーションを作成します。ここでは、クラス名に DSWAD と指定して、「HTTP 認証をサポート」オプションをチェックします。



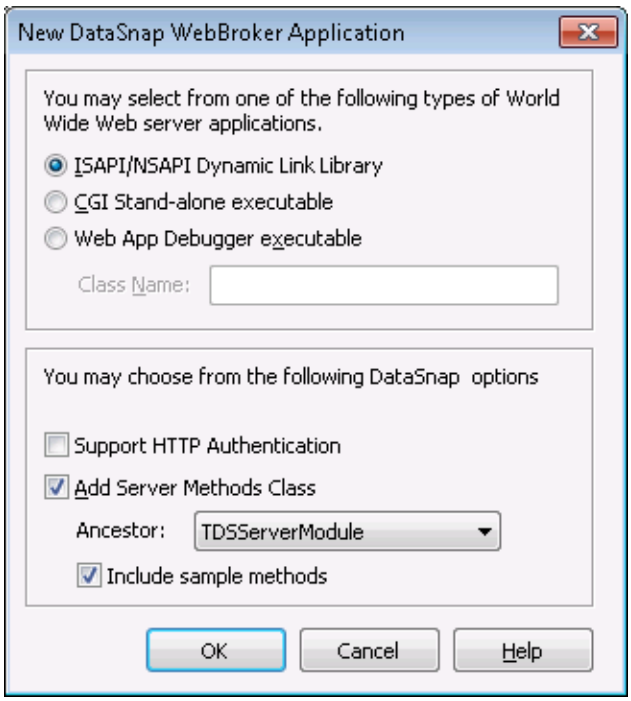
［OK］をクリックすると、3 つのユニットを持った新しいプロジェクトが作成されます。デフォルトのプロジェクトディレクトリに Project1 と Unit1 ファイルが存在していないのであれば、プロジェクト名は Project1、ユニットは Unit1、ServerMethodsUnit1、そして Unit2 と命名されます。初めのユニットは空のフ

ォームでなければなりません。これは Web アプリケーションデバッカ用実行形式固有のもので、もちろん他の Web ターゲットには不要です。このユニットを WADForm.pas として保存してください。2 つ目のユニットは ServerMethodsUnit1.pas と呼ばれ、サーバーモジュールが含まれ、指定されたダイアログ TDSServerModule を継承しています。このユニットにはすぐに戻りますが、今は与えられた ServerMethodsUnit1.pas という名前を使って保存しておいてください。3 つ目のユニットは Unit2 という名称で、4 つのコンポーネントがあらかじめ配置された Web モジュールです（もし「HTTP 認証のサポート」オプションをクリックしていなければ 3 つです）。これは受信したリクエストを受け取り、プロジェクト内の DataSnap サーバーモジュールに配布するユニットです。このユニットを DSWebMod.pas として保存してください。

最後に、このプロジェクトが DataSnap Web アプリケーションデバッカ用実行形式のサーバーだということを示すために、プロジェクトを DSWADServer.dproj という名称で保存してください。

5.2. ISAPI ターゲット

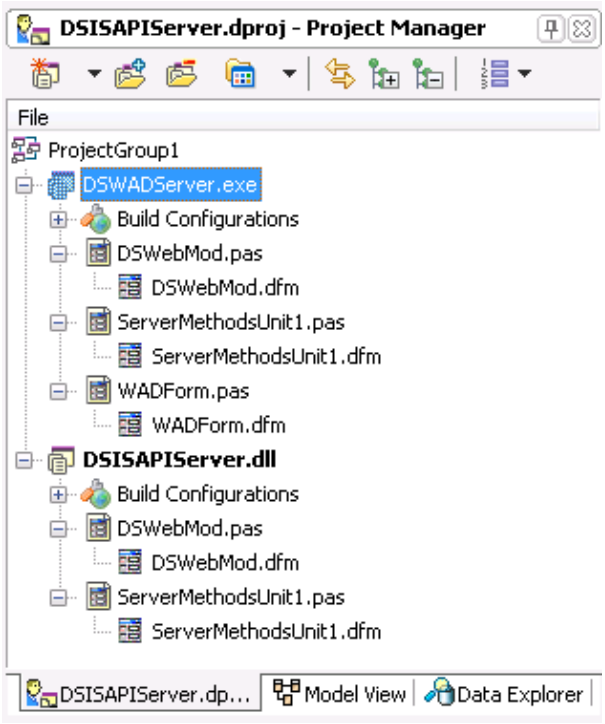
ServerMethodsUnit1.pas および DSWebMod.pas ユニットの修正とカスタマイズを続ける前に、プロジェクトグループに新しいプロジェクトを追加しなくてはいけません。今回は、ISAPI/NSAPI DLL アプリケーションです。プロジェクトグループを右クリックして［新規プロジェクトの追加］を選択し、オブジェクトリポジトリから新規プロジェクトを作成してください。次に、DataSnap Server カテゴリーから［DataSnap WebBroker アプリケーションの新規作成］ウィザードを選択し、新しい ISAPI/NSAPI DLL アプリケーションを作成します。DSWADServer プロジェクトにある既存のユニットを再利用するので、このときにダイアログ下部にあるオプションを修正する必要はありません。



［OK］ボタンをクリックして、新しいプロジェクトを生成します（プロジェクトグループに追加されます）。これには再び Project1 という名称が付けられ、新しいプロジェクトに追加されたユニット ServerMethodUnit2.pas と Unit1.pas が含まれています。

ここでは、新しいユニット ServerMethodUnit2.pas と Unit1.pas を使用する代わりに、すでに DSWADServer プロジェクトの一部であるユニット ServerMethodUnit1.pas と DSWebMod.pas を使用します。そのため Project1.dll ノードの下の ServerMethodUnit2.pas ノードを右クリックし、［プロジェクトから削除］を選択します。確認ダイアログで［OK］をクリックします（まだこれらを保存していなければ、ディスクから ServerMethodUnit2.pas と.dfm ファイルを削除する必要はありません）。Unit1.pas についても同じ操作を行うと、Project1.dll にはもうユニットはありません。次に、Project1.dll を右クリックして DSWebMod.pas と ServerMethodUnit2.pas の両方のユニットをこのプロジェクトに追加します。最後に Project1 を DSISAPIServer.dproj へ名前を変更してプロジェクトグループを完成させます。

次の画面ショットのように、現在、1 つのプロジェクトグループに 2 つのプロジェクトがあり、DSWebMod.pas と DSSererMethodUnit1.pas ユニットを共有している状態になります。



この設定は、2 つのプロジェクトの構築を可能にします。DSWADServer を検証とデバッグのためのターゲットとして使用し、DSISAPIServer を IIS 上の DataSnap サーバーの実際の展開のために使用します。

ServerMethodsUnit1 に Web メソッドを追加する作業を進める前に、ISAPI/NSAPI プロジェクトで、TDSServerModule の自動インスタンスを作成するコードをプロジェクトファイルから取り除く「修正」を加える必要があります。TDSServerModule は、Web モジュールのように最終的にはデータモジュールであり、Web Broker アプリケーション内で Web モジュールが 1 つしか存在できないため、SAPI DLL の実行しようとするとエラーメッセージが表示されます。

DSISAPIServer.dpr プロジェクトソースコードを開き、メインの begin-end ブロックを次のように変更します。

```
begin
  CoInitFlags := COINIT_MULTITHREADED;
  Application.Initialize;
  Application.CreateForm(TWebModule2, WebModule2);
//  Application.CreateForm(TServerMethods1, ServerMethods1);
```

```
  Application.Run;
end.
```

これによってデータモジュールが 1 つしか許されていないというエラーメッセージを回避できます。（配布された）ISAPI DLL を実際に呼び出すときには、このエラーメッセージを目にすることはないかもしれませんが、これは単にサーバーエラーかタイムアウトであり、この問題について覚えておくことはとても重要なことです。

5.3. サーバーメソッド - 配布とクライアント

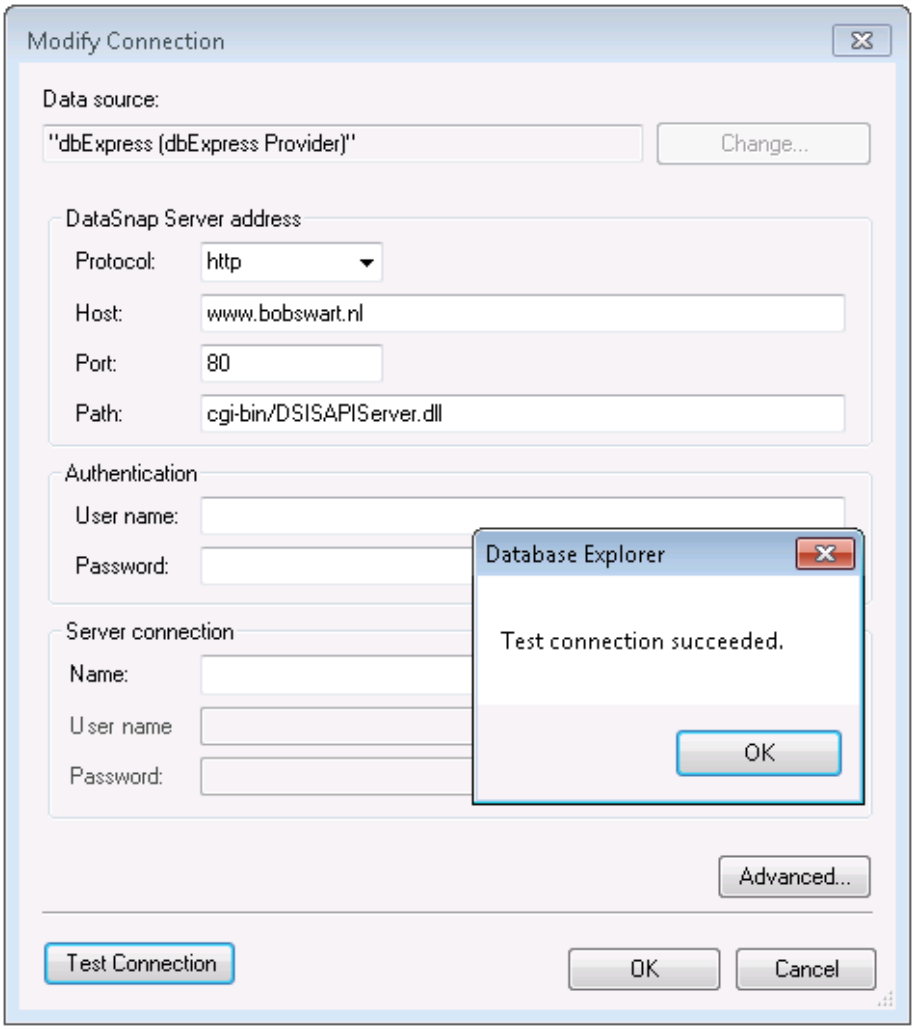
機能を追加するときには、ServerMethodUnit1.pas ユニットのみで作業すればよく、これは両方のターゲットで共有できます。デフォルトではサンプルメソッドが既に 1 つ含まれていますが、DataSnap サーバーの Windows バージョンのように、2 つのメソッドを追加できます（Server Methods ユニットで必要とされるコンポーネントとソースコードについては 2 章の 2.1.4 を参照してください）。

一度サーバーメソッドを実装したら、Microsoft Internet Information Services などの Web サーバーに ISAPI DLL を配布できるようになります。これは Jim Tierney による記事で詳細に説明されているので、ここでは省略します（http://blogs.embarcadero.com/jimtierney/2009/08/20/31502 をご覧ください）。

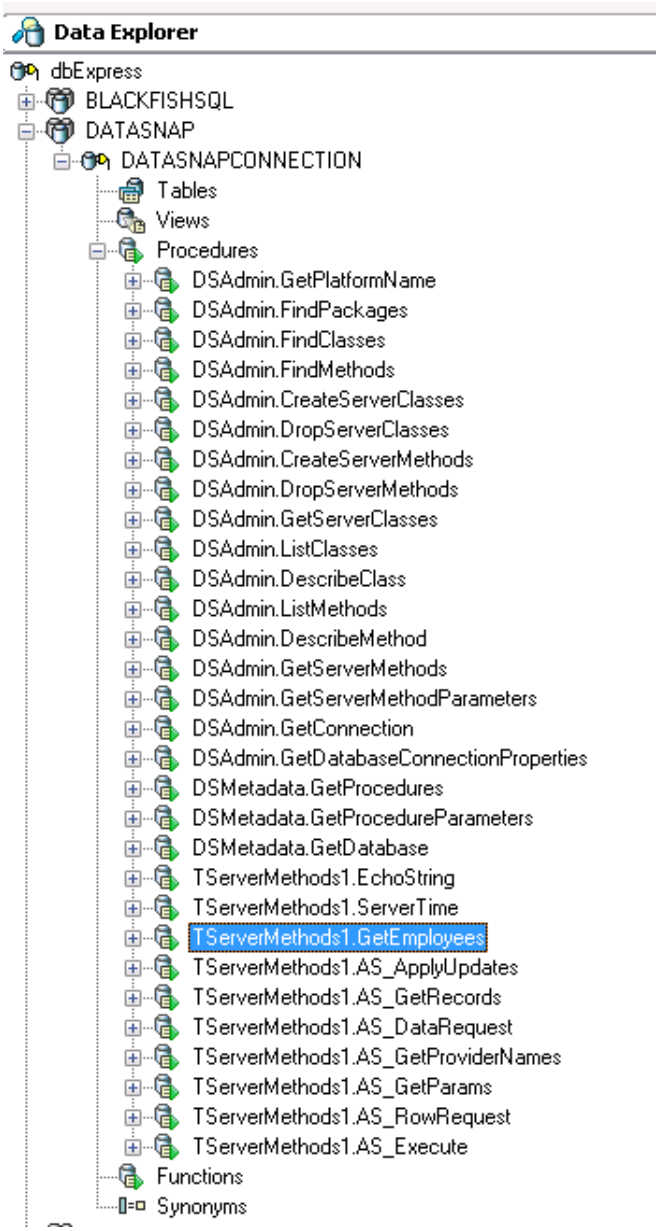
配布用の Web サーバーがない場合には、私のサーバー上に配布された DataSnap ISAPI サーバーと通信することができます。私は TDataSetProvider をエクスポーズしておらず、GetEmployees メソッドは一切データを返しませんが、ServerTime と EchoString メソッドは順調に動作していて、DataSnap クライアントをこのサーバーに対するテストを記述するには十分なはずです。

クライアントアプリケーション内部の ISAPI DataSnap サーバーに接続する前に、ISAPI DataSnap サーバーへ接続できるかどうか見るには、データエクスプローラを使用するといいでしょう。データエクスプローラには、現在 DATASNAP と呼ばれる新しいカテゴリーがあり、これを開けば修正が可能な DATASNAPCONNECTION（右クリックをして［接続を修正］を選択するのみ）という 1 番目のコネクション設定があります。

このダイアログでは、プロトコル、ホスト（もし自分の Web サーバーが使用できないときは www.bobswart.nl を使用できます）、ポート、そしてサーバーの ISAPI DataSnap サーバーアプリケーションへの URL パス（これは cgi-bin/DSISAPIServer.dll になります）を指定することができます。［接続テスト］をクリックして、すべてが動作中であることを確認します。

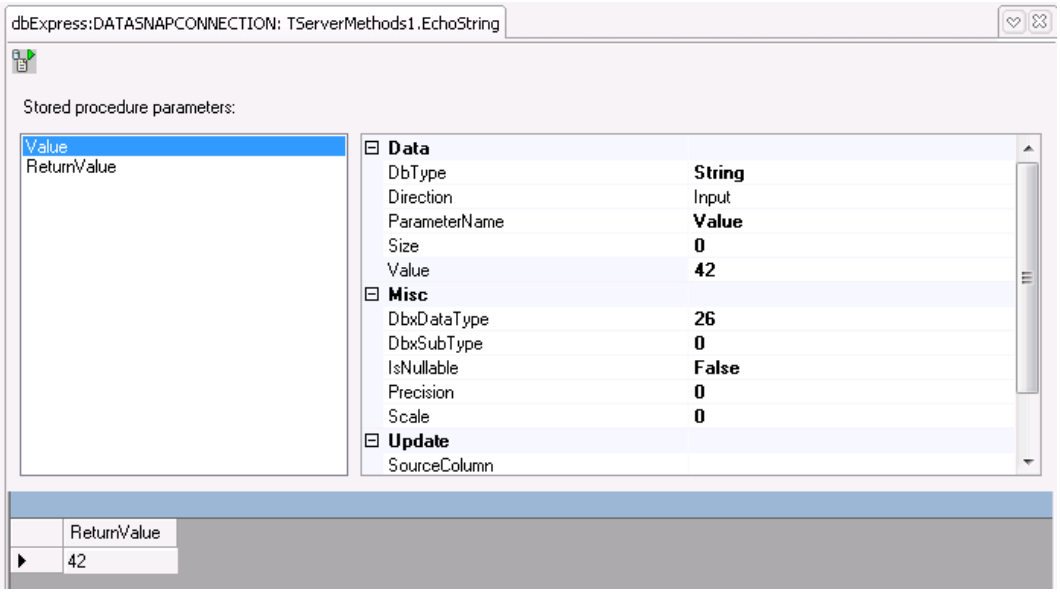


［OK］をクリックしてダイアログを閉じると、データエクスプローラで DATASNAPCONNECTION を開いて、テーブル、ビュー、プロシージャ、関数、シノニムを見ることができます。次のように、プロシージャには、すべての DSAdmin、DSMetaData、TServerMethods1.AS_xxx、そして 3 つのカスタムサーバーメソッドである EchoString、ServerTime、GetEmployees が含まれます。



ここでは、DataSnap クライアントアプリケーションを書くことなく、これらのメソッドのうちのいくつかをテストすることができます。例えば EchoString については、何を送ったのか、そして何が返ってくるのかを確認するためのテストをすることができます。

TServerMethods1.GetEmployees プロシージャを右クリックすると、［パラメータの表示］メニューを選択できます。これは IDE に新しいウィンドウを表示して、Value パラメータの値を入力することができます（例えば「42」など）。この新しいウィンドウを右クリックして［実行］を選択すると、リモートサーバーメソッドを実行できます。この結果は ReturnValue として次のように表示されます。



これにより、DataSnap サーバー経由でカスタムサーバーメソッドを呼び出すことができることが証明されました。DataSnap クライアントアプリケーションをサーバーに接続するには、TSQLConnection プロパティを修正するだけです。以前は DataSnap サーバーの Windows バージョンに接続しましたが、今度は Web バージョンに接続するための設定変更が必要です。

もし、私の Web サーバー経由で DSISAPIServer.dll を使用する場合には、TDataSetProvider が無効になっており、GetEmployees メソッドでまったくデータを返さないこと、ただし ServerTime と EchoString メソッドは呼び出し可能であることを忘れないでください。

6. REST と JSON

DataSnap 2010 は、REST と JSON の両方をサポートしています。DataSnap 2010 では、DataSnap HTTP リクエストをサポートするために REST 機能を備えています。例えば DataSnap サーバーの URL が http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll の場合、この URL に /datasnap/rest を追加して、Server Method クラスの名前、メソッド、そして引数を続ければよいのです。

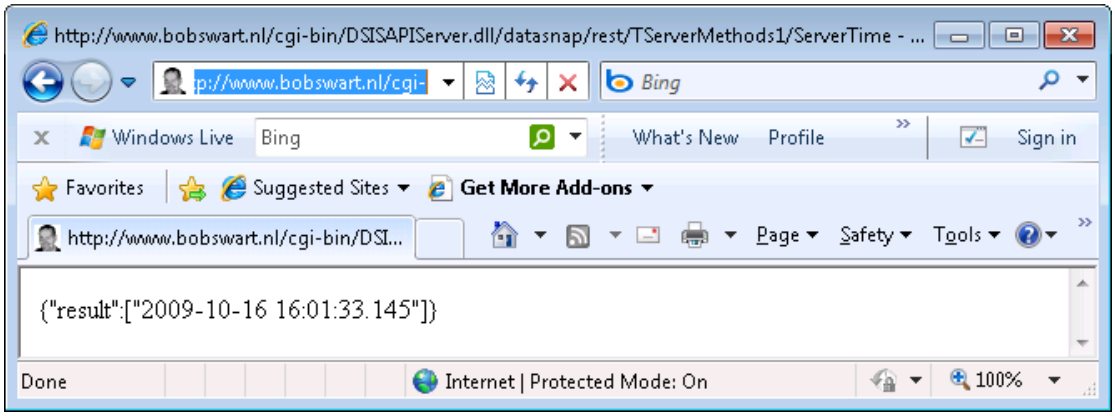
一般的なシンタックスは次のとおりです。

```
http://server/datasnap/rest/<class>/<method>/<parameters>
```

私のサーバー上の DSISAPIServer.dll の TServerMethod1 モジュール内の ServerTime メソッドについては、URL は次のようになります。

```
http://www.bobswart.nl/cgi-
bin/DSISAPIServer.dll/datasnap/rest/TServerMethods1/ServerTime
```

以下は、この REST 対応 URL を呼び出し、その結果を JSON 戻り値で取得した例です。



ブラウザに表示されている以下の結果が、JSON の構造です。

```
{"result":["2009-10-16 16:01:33.145"]}
```

REST の詳細については、Marco Cantù が「Delphi 2010 と REST クライアント」についてのホワイトペーパーで説明しています。

6.1. コールバック

DataSnap サーバーの呼び出しで REST 対応の戻り値が得られるのとは別に、JSON はコールバックメソッドの実装でも利用されています。DataSnap 2010 は、クライアントサイドのコールバックメソッドとサーバーメソッドのコンテキストの実装をサポートしています。これは、（DataSnap クライアントによって呼び出された）クライアントからサーバーメソッドに引数として渡されたコールバック関数を、サーバーメソッドの実行中に、サーバーから呼び出すことができるということを意味しています。

例として EchoString メソッドを、コールバック機能を追加するように修正してみましょう。EchoString メソッドの定義を、次のように修正します。

```
    function EchoString(Value: string; callback: TDBXcallback): string;
```

TDBXcallback 型は、DBXJSON ユニット内で定義されています。新しい EchoString メソッドを実行する前に、クライアントサイドでコールバックメソッドがどのように定義できるのかを確認する必要があります（結局はサーバーが呼び出すことのできるクライアントメソッドだからです）。

クライアントサイドで TDBXCallback を継承した新しいクラスを宣言し、Execute メソッドをオーバーライドします。

```
type
  TCallbackClient = class(TDBXCallback)
  public
    function Execute(const Arg: TJSONValue): TJSONValue; override;
  end;
```

Execute メソッド内で TJSONValue 型の Arg 引数を得ることができるので、これを複製して実際のコンテンツを操作できます。Execute メソッドは TJSONValue 自身を返すこともできるので、ここでは同じ値をただ返すようにします。

```
function TCallbackClient.Execute(const Arg: TJSONValue): TJSONValue;
var
  Data: TJSONValue;
begin
  Data := TJSONValue(Arg.Clone);
  ShowMessage('Callback: ' + TJSONObject(Data).Get(0).JSonValue.value);
  Result := Data
end;
```

この例では、コールバックメソッドはメソッドが実際に返す前の EchoString メソッドに渡された値を表示しています（つまりメソッドが実行中ということです）。

サーバーサイドでの新しい EchoString メソッドの実装には、次のように TJSONObject 内に文字列の値を加えて callback.Execute メソッドに入力しなければなりません。

```
function  TServerMethods2.EchoString(Value:  string;  callback:  TDBXcallback):
string;
var
  msg: TJSONObject;
  pair: TJSONPair;
begin
  Result := Value;
  msg := TJSONObject.Create;
  pair := TJSONPair.Create('ECHO', Value);
  pair.Owned := True;
  msg.AddPair(pair);
  callback.Execute(msg);
end;
```

サーバーサイドで実際の EchoString メソッドが終了する前に、クライアントサイドでコールバック関数が実行されて戻っていることに注意してください。

最後に、新しい TCallbackClient のインスタンスのように、2 つめの引数としてコールバッククラスを入力しなければならないため、コールバッククラスを渡すようにクライアントサイドでの EchoString メソッドの呼び出しにも変更が必要です。

```
var
  MyCallback: TCallbackClient;
begin
  MyCallback := TCallbackClient.Create;
```

```
  try
    Server.EchoString(Edit1.text, MyCallback);
  finally
    MyCallback.Free;
  end;
end;
```

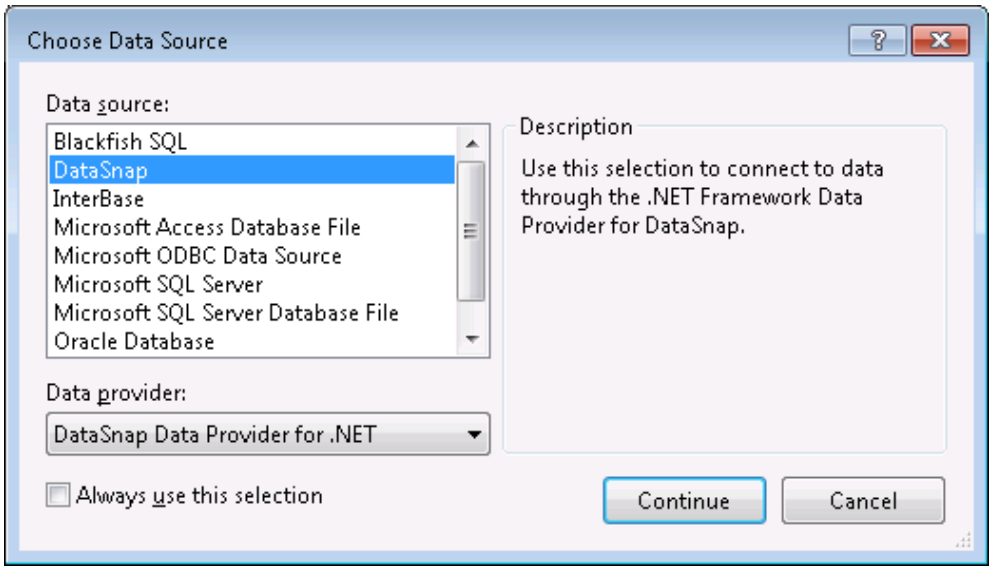
このシンプルな例は、クライアントサイドのコールバックメソッドを DataSnap 2010 でどのように使用すればよいかデモンストレーションしたものです。

7. DataSnap と.NET

Delphi Prism 2010 は、これまで作成してきた Win32 サーバー用の DataSnap .NET クライアントの構築に利用できます。Delphi Prism 2010 DataSnap クライアントを構築するには、DataSnap サーバーが実行中で、すでに設計時に接続できる状態になっていなければなりません。

Delphi Prism 2010 を起動して［表示｜サーバーエクスプローラ］メニューを選択し、Delphi Prism サーバーエクスプローラを表示します。実際に DataSnap サーバーを使用できるかどうか検証するために、ここで最初に接続しなければなりません。

サーバーエクスプローラは「データ接続」と呼ばれるルートノードを持ったツリービューになっています。「データ接続」を右クリックして［接続の追加］を選択します。次のダイアログでは、データソースのリストから DataSnap を選択します（もしデータソースがすでに選択されていたら、［変更］をクリックします）。



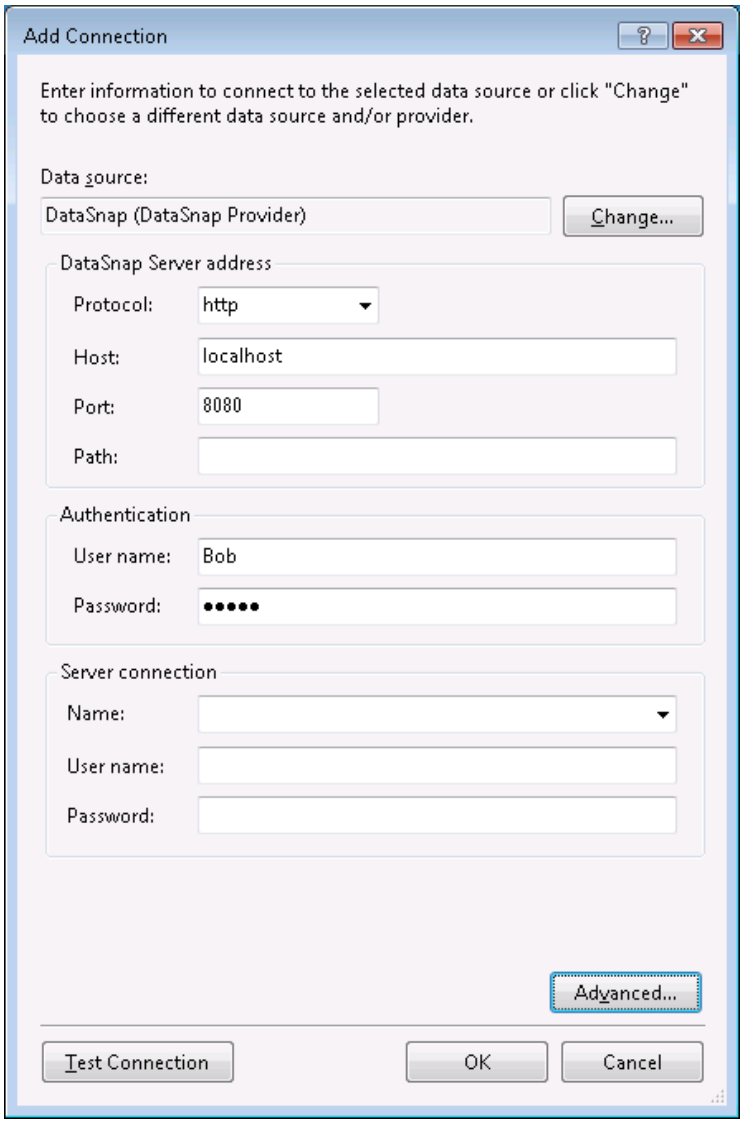
常に DataSnap データ接続しか構築しないという場合を除いては、“常にこれを選択” チェックボックスのチェックをはずしても差し支えありません。

［次へ］をクリックして、ダイアログの次のページを表示します。ここでは、DataSnap サーバーへの接続の詳細を指定できます。「プロトコル」ドロップダウンコンボボックスで「tcp/ip」または「http」を選択します。次に「ホスト」を指定します（DataSnap サーバーが動作中のマシン名。同じローカルマシンでテストしている場合には localhost）。次に「ポート番号」を指定します。デフォルトでは、HTTP の場合の

Port 80、TCP/IP の場合 Post 211 になりますが、このホワイトペーパーを読んでいればこの両方が違う値であることもある（または違う値でなければならない）ことが分かると思います（ここでは、ServerContainerUnitDemo ユニットのトランスポートコンポーネントで指定した値と同じものを必ず指定してください）。

次のプロパティには、「パス」を指定します。これは DataSnap サーバーベースの Web Broker(DataSnap Web サーバーへの URLPath を指定する必要がある場合（http://..../ ドメインパートの後の部分）に重要です。

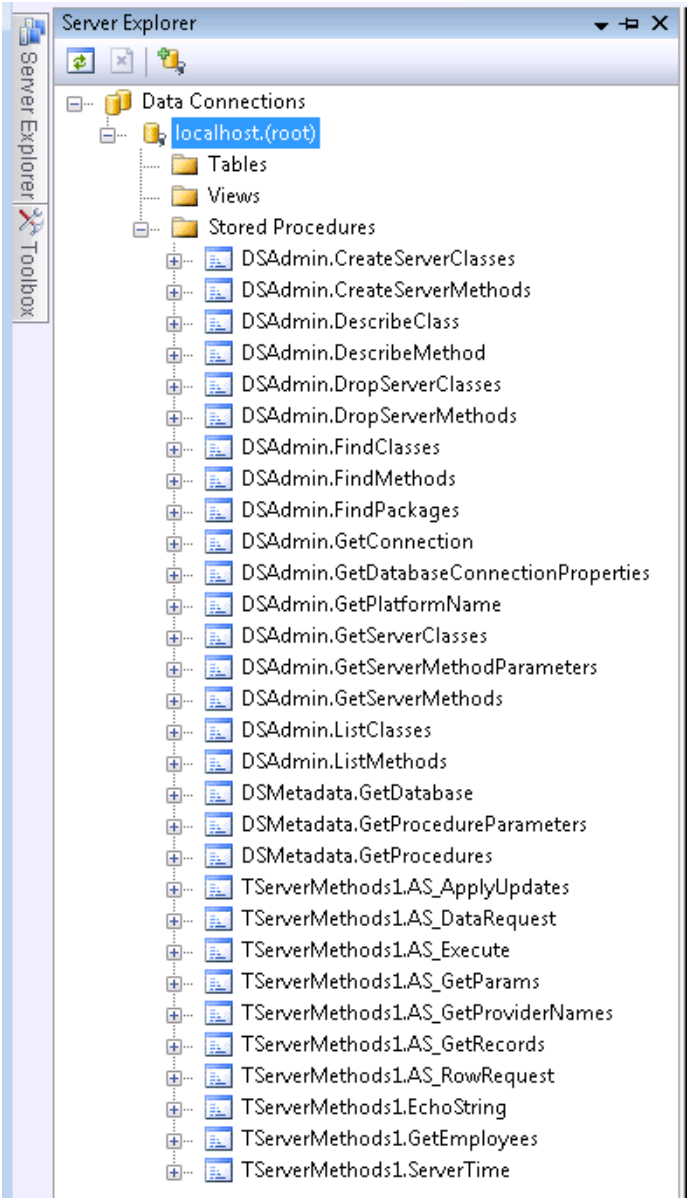
最後に、DataSnap サーバーが HTTP 認証を使用している場合には認証の「ユーザー名」と「パスワード」の指定を忘れないでください。



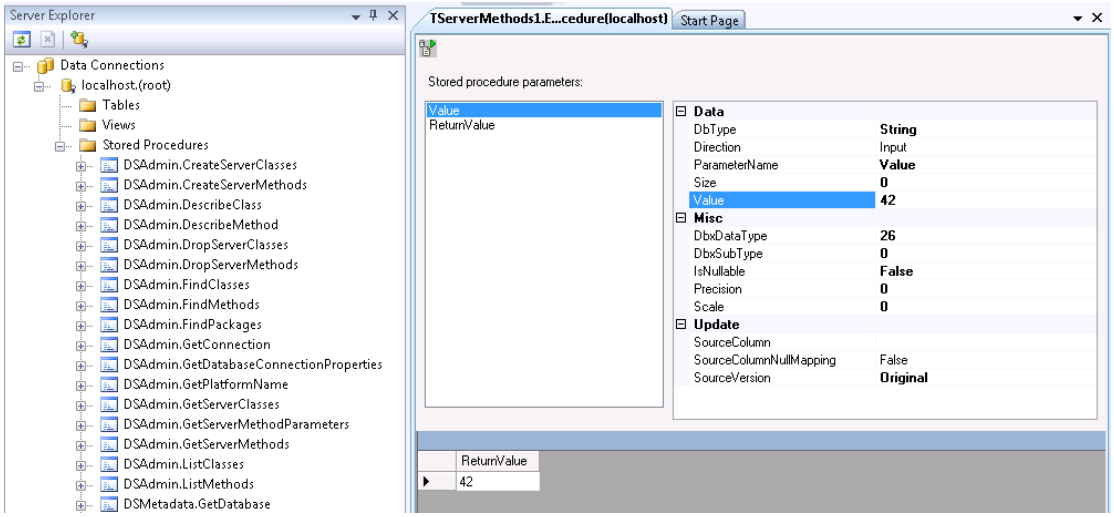
［接続テスト］ボタンをクリックして、DataSnap サーバーに接続できるかを検証してみましょう。すべてが正しく設定されていれば、「テスト接続成功」のダイアログが表示されます。

［OK］をクリックすると、データ接続ツリーに新しい接続エントリー追加されます。この例では、localhost ノードです。新しいノードを開くと、テーブル、ビュー、ストアドプロシージャのサブノードが表示されます。テーブルとビューは空ですが、ストアドプロシージャは DataSnap サーバーで公開された

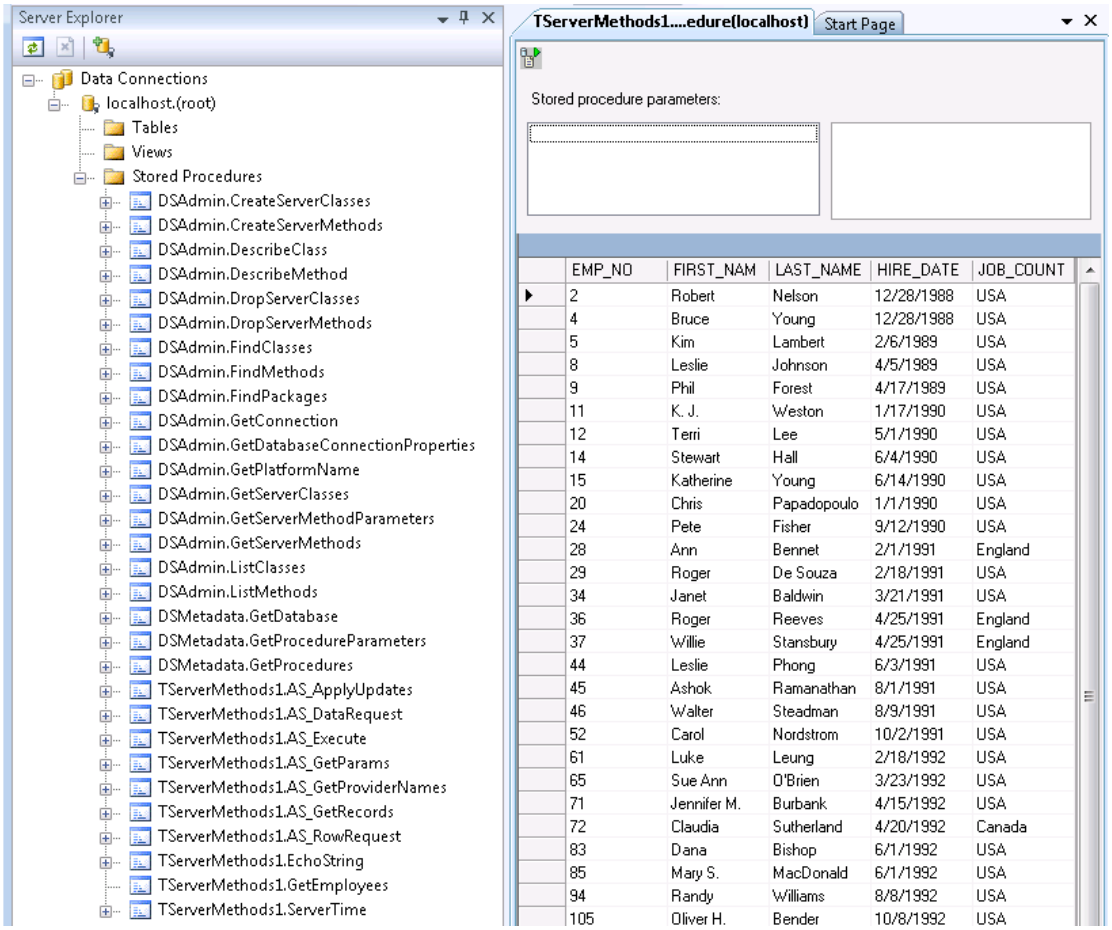
サーバーメソッドがすべて含まれています。ここには、カスタムサーバーメソッドの EchoString、GetEmployees と ServerTime も含まれています。



サーバーエクスプローラのストアドプロシージャの一覧から DataSnap のサーバーメソッドについて、検証してみましょう。例えば、EchoString メソッドを右クリックして［パラメータの表示］を選択します。Value パラメータの値を入力できるウィンドウ表示されるので、42 と入力します。次に、ウィンドウを右クリックして［実行］を選択します。DataSnap サーバーからメソッドを実行し、次のようなストアドプロシージャパラメータウィンドウが結果として表示されます。



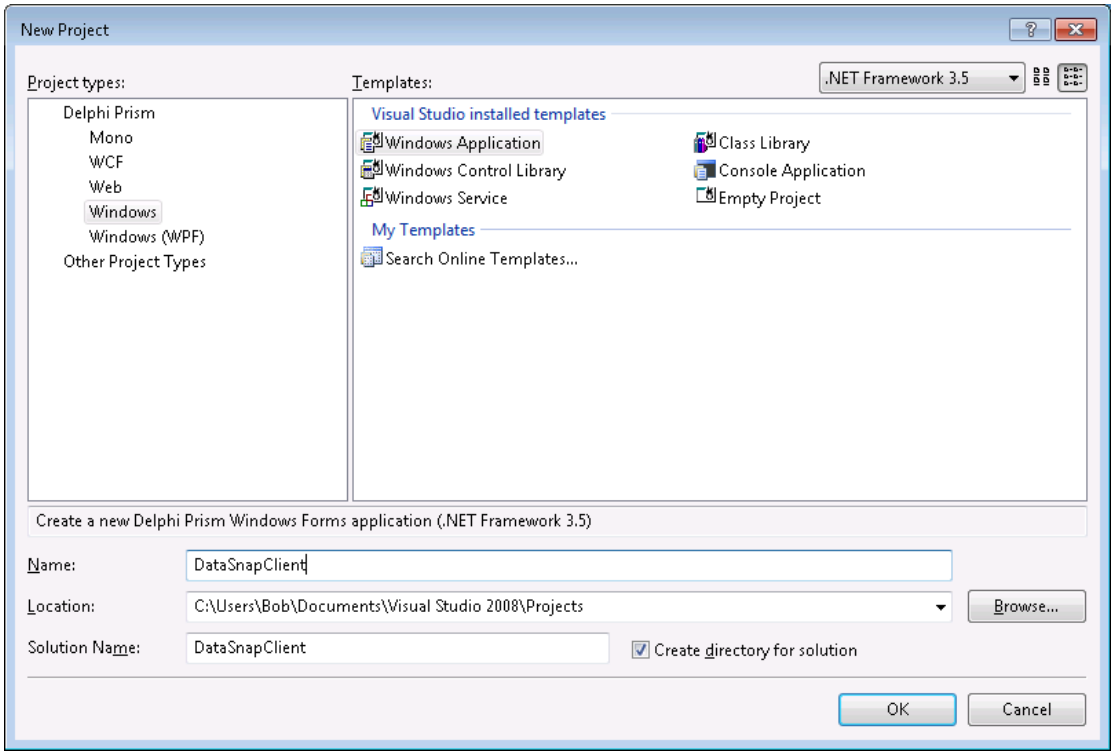
これが正しく表示されたら、続いて getEmployees メソッドを使用して Employees テーブルからデータを取り出す方法を説明します。このストアドプロシージャにはパラメータがありませんが、［パラメータの表示］コマンドを選択することができます。これはストアドプロシージャのパラメータが空のリストとして表示されます。ここでもウィンドウを右クリックして、［実行］を選択します。今回は GetEmployees メソッドから、Employees テーブルの完全なレコードセットが返されます。



7.1. WINFORMS クライアント

サーバーエクスプローラで DataSnap サーバーメソッドを扱うのは面白いかもしれませんが、.NET アプリケーションからこれらを呼び出す方がより実用的です。最後の例のために、［ファイル | 新規プロジェクト］メニューを選択し、Delphi Prism で新規プロジェクトウィザードを実行します。使用可能なターゲットのオーバービューが表示されます。

プロジェクトタイプとして「Windows」から「Windows アプリケーション」を選択し、名前を WindowsApplication1 から DataSnapClient に変更します。



［OK］をクリックすると、Delphi Prism IDE 上で、メインフォーム用の Main.pas ユニットとともに新規プロジェクト DataSnapClient が作成されます。

サーバーエクスプローラで、前の章で作成した DataSnap サーバーへの新しい接続を選択します。プロパティエクスプローラは ConnectionString を含むプロパティを次のように表示します。

```
communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob;ds
authenticationpassword=Swart
```

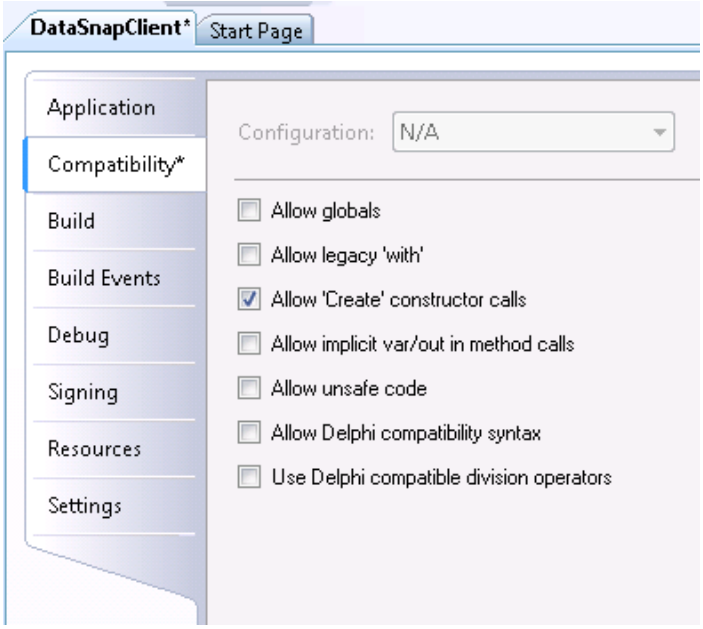
データ接続ノードを右クリックして［Generate Client Proxy］メニューを選択します。これは ClientProxy1.pas ファイルを生成して、EchoString、ServerTime、GetEmployees を含む多くのメソッドを持った TServerMethods1Client と呼ばれるクラスを定義します。クラスの定義からのスニペットは次のとおりです。

```
TServerMethods1Client = class
public
  constructor (ADBXConnection: TAdoDbxConnection);
  constructor (ADBXConnection: TAdoDbxConnection; AInstanceOwner: Boolean);
```

```
    function EchoString(Value: string): string;
    function ServerTime: DateTime;
    function GetEmployees: System.Data.IDataReader;
```

プロキシクラスのほかに、プロジェクトの References ノードに多くのリファレンスも追加されています。具体的には、Borland.Data.AdoDbxClient、Borland.Data.DbxClientDriver です。

TServerMethods1Client のコードスニペットからもわかるように、このクラスには 2 つのコンストラクタがあります。どちらも引数として ADBXConnection があり、2 つ目には、AInstanceOwner Boolean もあります。これは引数を持ったコンストラクタを呼び出す必要があることを意味します。そしてこれをサポートするために、プロジェクトの設定に修正を加えなければなりません。ソリューションエクスプローラの DataSnapClient ノードを右クリックして、［プロパティ］を選択します。表示されたウィンドウで［互換性］タブをクリックして「'Create'コンストラクタ呼び出しを使用可能にする」オプションをチェックします。これによって new 演算子を使用する代わりに、引数を伴う Create コンストラクタを呼び出すことができるようになります。



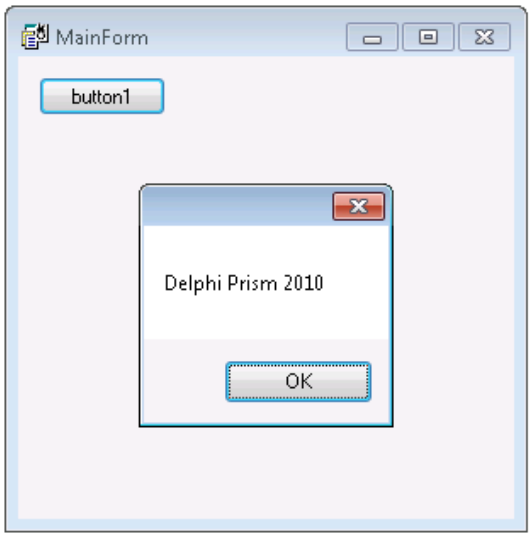
メインフォームに戻り、ボタンを設置しましょう。ボタンの Click イベントでは、DataSnap サーバーへ接続し、そのメソッドの 1 つを呼び出すことができます。

```
method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
var
  Client: ClientProxy1.TServerMethods1Client;
  Connection: Borland.Data.TAdoDbxDatasnapConnection;
begin
  Connection := new Borland.Data.TAdoDbxDatasnapConnection();
  Connection.ConnectionString :=

'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob
;dsauthenticationpassword=Swart';
  Connection.Open;
  try
    Client := ClientProxy1.TServerMethods1Client.Create(Connection);
    MessageBox.Show(
      Client.EchoString('Delphi Prism 2010'));
```

```
  finally
    Connection.Close;
  end;
end;
```

その結果は Delphi Prism 2010 のエコーで、次のとおりになります。



似たような方法で GetEmployees メソッドを呼び出して、DataGridView に結果を割り当てることができます。これは、GetEmployees が DataSet や DataTable ではなく、（TSQLDataSet の結果と同じく）IDataReader を返すため、若干の問題をもたらします。Dataset の新しい DataTable の中の GetEmployees の結果をロードするためのコードに、数行書き加える必要があります（Win32 側での TClientDataSet と同等です）。

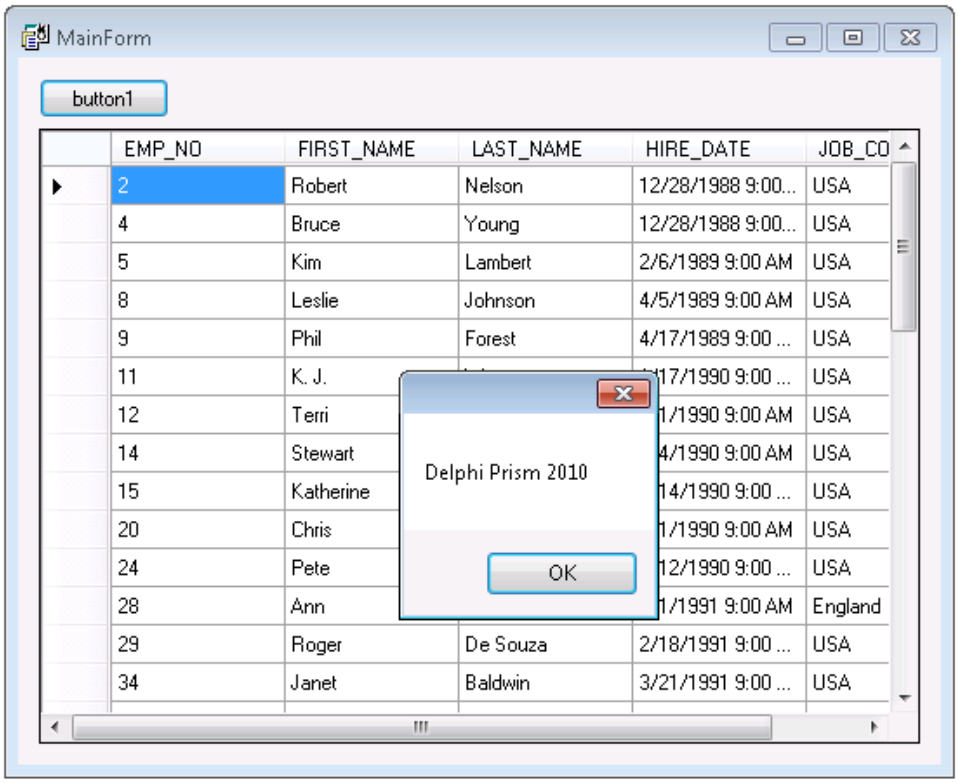
```
method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
var
  Client: ClientProxy1.TServerMethods1Client;
  Connection: Borland.Data.TAdoDbxDatasnapConnection;
  Employees: System.Data.IDataReader;
  ds: System.Data.DataSet;
  dt: System.Data.DataTable;
begin
  Connection := new Borland.Data.TAdoDbxDatasnapConnection();
  Connection.ConnectionString :=

'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob
;dsauthenticationpassword=Swart';
  Connection.Open;
  try
    Client := ClientProxy1.TServerMethods1Client.Create(Connection);
    Employees := Client.GetEmployees;
    ds := new DataSet();
    dt := new DataTable("DataSnap");
    ds.Tables.Add(dt);
    ds.Load(Employees, LoadOPtion.PreserveChanges, ds.Tables[0]);
    dataGridView1.DataSource := ds.Tables[0];

    MessageBox.Show(
      Client.EchoString('Delphi Prism 2010'));
  finally
    Connection.Close;
```

```
  end;
end;
```

結果として、次のように Delphi Prism WinForms アプリケーションの DataGridView にデータが表示されます。これは、DataSnap サーバーに接続する.NET シンクライアントを記述する例となります。



8. まとめ

このホワイトペーパーでは、使いたい場所で DataSnap を活用できるように解説してきました（Windows では、GUI、サービス、コンソールアプリケーションと一緒に、また Web では CGI、ISAPI、Web App Debugger アプリケーションと一緒に）。同様に Win32 や.NET クライアントについて、HTTP 認証付きの TCP/IP、HTTP、そして圧縮や暗号化などの任意のフィルターといった必要と思われることがらについても説明してきました。

DataSnap 2010 は、DataSnap2009 に比べ、大幅な拡張と強化が行われ、かつ DataSnap と MIDAS の COM ベースのオリジナルバージョン以降の多くの改善も取り入れられています。

Bob Swart – Bob Swart Training & Consultancy (eBob4)

エンバカデロ･テクノロジーズについて

エンバカデロ・テクノロジーズは、1993 年にデータベースツールベンダーとして設立され、2008 年にボーランドの開発ツール部門「CodeGear」との合併によって、アプリケーション開発者とデータベース技術者が多様な環境でソフトウェアアプリケーションを設計、構築、実行するためのツールを提供する最大規模の独立系ツールベンダーとなりました。米国企業の総収入ランキング「フォーチュン 100」のうち 90 以上の企業と、世界で 300 万以上のコミュニティが、エンバカデロの Delphi®、C++Builder®、JBuilder®といった CodeGea™製品や ER/Studio®、DBArtisan®、RapidSQL®をはじめとする DatabaseGear™製品を採用し、生産性の向上と革新的なソフトウェア開発を実現しています。エンバカデロ・テクノロジーズは、サンフランシスコに本社を置き、世界各国に支社を展開しています。詳細は、www.embarcadero.com/jp をご覧ください。