



白页

Delphi 2010 DataSnap: 何时需要数据,如何 获取数据.

Bob Swart—Swart Training&Consultancy (eBob42)

2009 年10月

Corporate Headquarters	EMEA Headquarters	Asia-Pacific Headquarters
100 California Street, 12th Floor	York House	L7. 313 La Trobe Street
San Francisco, California 94111	18 York Road	Melbourne VIC 3000
	Maidenhead, Berkshire	Australia
	SL6 1SF, United Kingdom	

在这个白页中我们将讲解Delphi2010 DataSnap架构新的特性和功能.

1. DATASNAP 历史

作为MIDAS起始于Delphi3, Delphi4是MIDAS II, Delphi5中是MIDASIII, 而后基于COM远程数据模块方式使用TCP/IP, HTTP, (D)COM构建出强大的通讯能力. 从Delphi6开始改名为DataSnap, 直到D2007这个框架一直在使用. D2009重新架构了DataSnap—移除COM依赖, 使用TCP/IP以更轻量级的方式生成远程服务对象和客户端连接能力. 同时提供了与Delphi Prism2009开发的.NET程序通讯的功能.

Delphi2010中构建于D2009架构之上, 并对此架构做了进一步的扩展, 包括使用两个向导来创建新的部署目标(VCL窗体, Window服务, 控制台及面向Web的ISAPI, CGI或Web App Debugger). HTTP(S)传输协议, HTTP验证, 客户端回调函数, REST和JSON的支持, 及使用过滤器来支持压缩和解压缩.

1.1 DATASNAP范例数据位置

本白页中我建议您使用Demo和范例来学习. 虽然Delphi支持很多数据库系统, 使用DBX4, ADO dbGo, 或其他数据存取技术, 为了演示方便我这里使用DBX4来操作BlackfishSQL的employee. jds数据库. 见 [C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL\employee. jds]. 在截图中可以看到我使用的是Windows Vista或Win7操作系统, 使用Windows Server 2008 Web编辑器来部署DataSnap ISAPI服务.

2. DATASNAP目标:如何获取数据

DataSnap2010支持三种不同的Windows方式:VCL窗体, Windows服务和控制台应用程序. 本节中我们将讨论他们的好处, 不同和每种方式最适合在什么情况下使用.

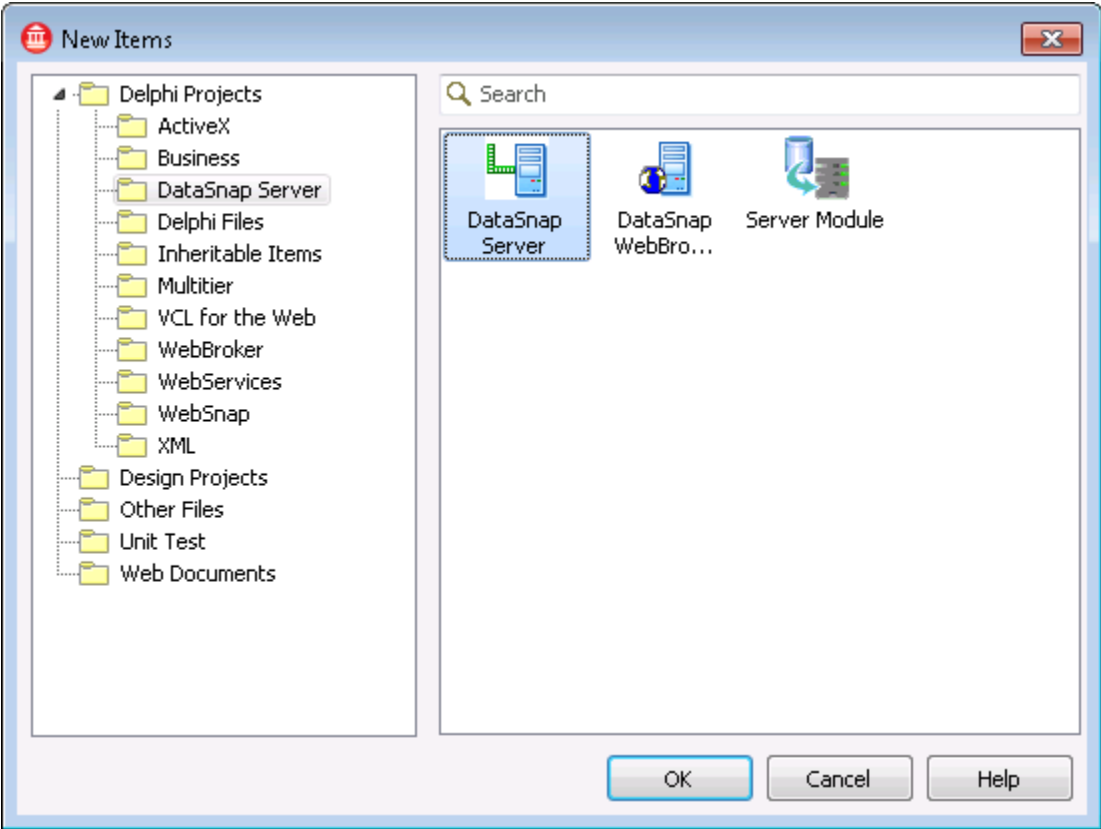
下面会创建一个DataSnap服务端和客户端, 我们将讲解 TDSServer, TDSServerClass, TDSTCPServerTransport, TDSHTTPService, TDSHTTPWebDispatcher和 TDSHTTPServiceAuthenticationManager组件, 以及自定义的服务方法和TDSServerModule类.

将讨论不同的传输协议(TCP, HTTP)的好处及传输效率. 并讨论DataSnap服务对象的不同生命期选项(Server, Session, Invocation), 及他们的效率和使用的建议. 最后, 讨论部署.

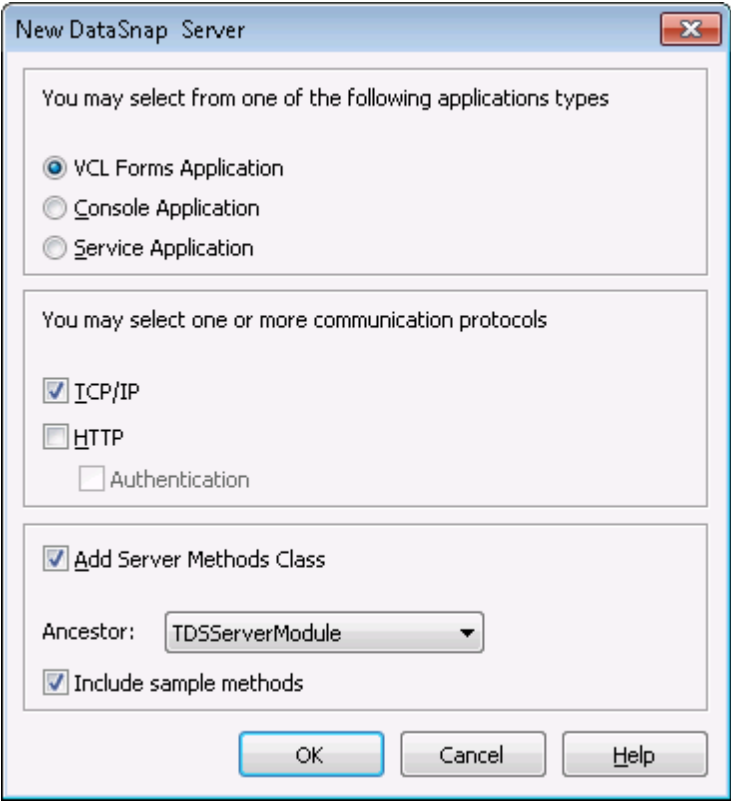
2.1. DATASNAP SERVER EXAMPLE

在Object Repository中有两个不同的DataSnap服务向导:一个是生成基于Windows的Datanap服务项目, 一个是生成基于WebBroker的DataSnap服务项目(需要部署到IIS或Apache). 我们将会演示.

启动了Delphi2010, 点击File→New→Other, 你会在Object Repository中看到DataSnap服务向导中显示三个图标:DataSnap Server, DataSnap WebBroker Server, 和Server Modul.



双击第一个(后面的两个在下面的小结中讲解), 弹出如下对话框:



界面中第一部分是控制项目类型的. 默认可以生成可视化的带有主窗体的VCL窗体应用程序. 第二个选项是创建控制台应用程序, 生成一个控制台窗口—可以用来输出请求应答信息(用WriteLn语句输出服务应用程序正在做什么). 这两种方式都是为了做范例或最初部署, 很少用于最终部署. 由于DataSnap架构不在基于COM, 客户端将不能使服务端启动. 因此为了响应客户端的请求, DataSnap服务端应该一直在运行. 如果你希望请求基于7x24全天候运行, DataSnap服务端必须同时也在运行中. 对应VCL窗体或控制台应用程序, 需要一个账户登录到Windows中后才能启动DataSnap服务, 背离了这种要求. 第三

者选择在这时最适合:一个Windows服务应用程序, 安装后配置成为自动启动, 当计算机启动后将自动运行(不在需要账户登录). 服务应用程序不会弹出界面, 很难调试Bug. 然而, 为了整合这三种的优势, 我将用几分钟创建一个项目组, 包括VCL窗体应用程序的DataSnap服务, 控制台DataSnap服务, 及Windows服务Datanap服务, 都共享供同一个自定义的服务方法, 这样就可以开发一个Datanap服务应用程序, 在需要的时候编译出三个不同类型的服务.

第二部分是选择使用的Datanap服务的通讯协议. 和DanaSnap2009相比, 我们可以看到多了一个HTTP通讯, 及HTTP验证. 为了更加灵活, 这里建议选择全部选项, 我们可以同时使用TCP/IP, HTTP, 及使用HTTP引入的HTTP验证.

第三部分已经为我们配置好了, 如果我们要提供一个服务方法类, 我们可以选择它的基类:TPersistent, TDataModule或TDSServerModule. 最后的一个选项最好, 使用RTTI来启动函数执行(也可能你觉得使用TDataModule更合适—不操作数据库, 或不适用其他非可视控件, 这是使用TPersitent也够用了).

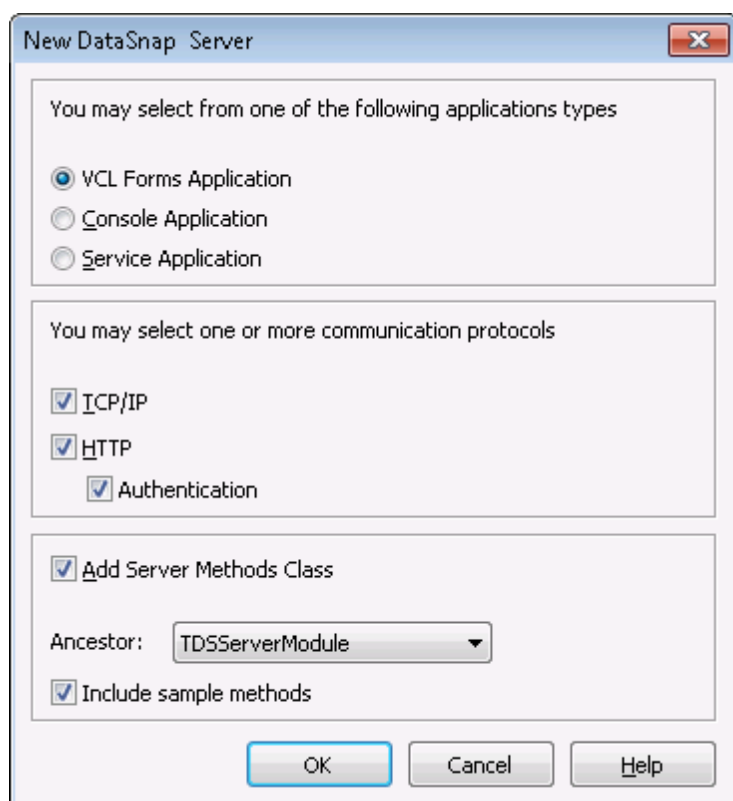
现在是从DSServer.pas中贴出来的一小段代码, 来说明TDSServerModule和TProviderDataModule(也是继承于TDataModule)之间的关系.

```
TDSServerModuleBase = class(TProviderDataModule)
public
  procedure BeforeDestruction; override;
  destructor Destroy; override;
end;
{$MethodInfo ON}
TDSServerModule = class(TDSServerModuleBase)
end;
{$MethodInfo OFF}
```

当不确定是使用TDsServerModule选择作为基类.

2.1.1. 创建多目标项目组-- VCL 窗体项目

如上面所说, 这里创建多目标的Datanap服务项目组. 首先创建一个VCL窗体应用程序作为Datanap服务, 选择所有的通讯协议.



默认创建了一个叫做Project1.dproj的项目,并带有三个单元文件,ServerContainerUnit1.pas,ServerMethodUnit1.pas和Unit1.pas.首先File→Save Project As保存项目,并输入有实际意义的文件名称.将Unit1.pas保存为MainForm.pas,ServerMethodsUnit1.pas保存为ServerMethodsUnitDemo.pas文件,保存Project1.dproj为DataSnapServer.dproj.

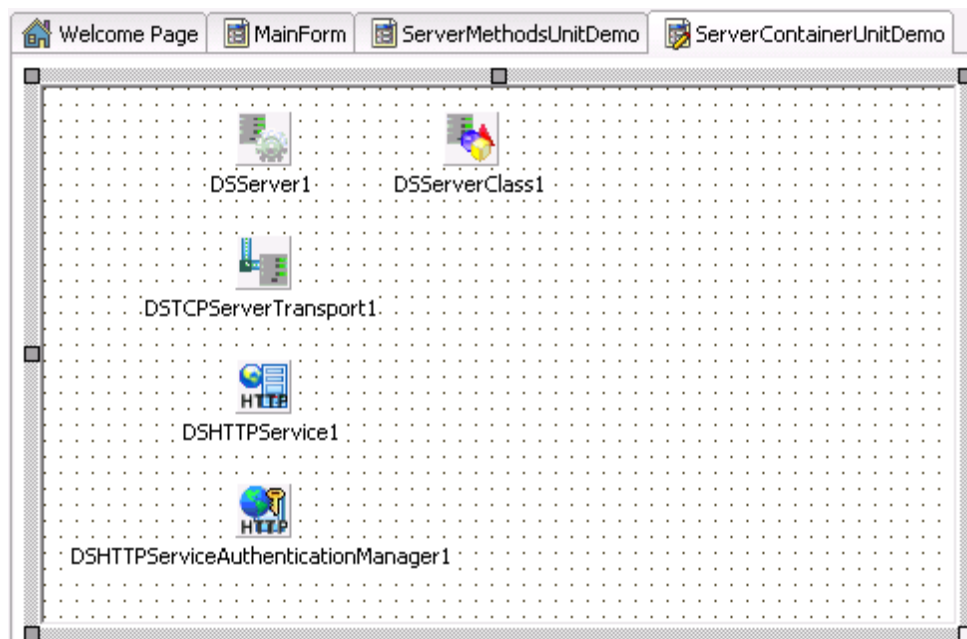
稍后我们将向项目组添加控制台应用程序和Window服务应用程序.首先我们来检查一下项目,并编译工程.如果你编译DataSnapServer项目,将会出现一个错误信息(由于我们将ServerMethodsUnit1.pas改名所致).错误原因是由于ServerContainerUnitDemo.pas单元中的Implementation部分引用了ServerMethodsUnit1.pas单元.为了修复这个冲突,修改引用单元的文件名称,从新编译.这是发现在第37行出现错误,使用了ServerMethodsUnit1中的TServerMethods1类型.修改ServerMethodsUnit1为ServerMethodsUnitDemo.这是可以正确的编译项目了.

ServerContainerUnitDemo的引用部分应该向下面代码所示:

```
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

打开ServerContainerUnitDemo单元,将会看到不少于五个组件:一个TDSServer,一个TDSServerClass,一个TDSTCPServerTransport(用于TCP/IP通讯),一个TDSHTTPService(用于HTTP通讯),一个TDSHTTPServiceAuthenticationManager组件(用于HTTP验证).



前面两个一直会存在,其他的三个则是根据选择的通讯协议生成的.

2.1.1.1.1. TDSSERVER

TDSServer组件只有四个属性,AutoStart,HideDSAdmin,Name和Tag. AutoStart属性默认设置为True,意味着在窗体创建后自动启动DataSnap服务.如果将AutoStart设置为False,需要手动调用Start方法启动服务,并调用Stop方法停止服务.可以调用Started方法验证DataSnap服务是否已经启动.

HideAdmin属性默认设置为False. 如果设置为True, 连接到DataSnap服务的客户端将无法调用Datanap服务中的TDSAdmin类的内置方法. TDSAdmin不是一个真正的类, 我们可以调用的TDSAdmin方法定义在DSNames单元:

```
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组件有五个事件: OnConnect, OnDisconnect, OnError, OnPrepare和OnTrace. 我们可以实现这五个事件来响应不同的情况, 例如像日志文件中写入日志.

OnConnect, OnDisconnect, OnError和OnPrepare事件有一个继承于TDSEventObject的参数, 包含了DxContext, 传输, 服务和DbxConnection组件的属性, 在OnConnect和OnDisconnect事件中TDSEventObject类型还包含了ConnectionProperties和ChannelInfo属性. TDSEventObject也包括了由错误引起的异常, TDSEventObject还包括了我们要使用的MethodAlias和ServerClass属性.

OnTrace事件有一个TDBXTraceInfo类型的参数. 注意由于这个OnTrace事件处理程序也会包含一些代码错误, 如TDBXTraceInfo和CBRTYPE是编译器未知的. 为了解决这个问题, 我们需要引用DBXCommon单元(为识别TDBXTraceInfo类型)和DBComonTypes单元(为识别CBRTYPE类型).

在OnConnect事件处理中, 我们可以通过ChannelInfo来查看连接信息, 例如(使用自定义的函数LogInfo向日志文件中写入信息):

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

在OnTrace事件处理程序中我们可以使用TraceInfo.Message中的信息记录服务端正在做什么.

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

注意, 在客户端也可以使用连接到TSQLConnection组件的TSQLMonitor组件来跟踪DataSnap服务端和客户端直接的通讯(在创建这个DataSnap服务的客户端时讲解).

一个跟踪日志输出如下所示:

```
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
```

如你所见, TraceInfo.Message中包括了传输信息的字节数和被调用的方法名称等信息。

2.1.1.1.2. TDSSERVERCLASS

TDSServerClass组件将服务端特定的类发布给远程客户端(使用动态方法调用)。

TDSServerClass组件有一个Server属性指向TDSServer组件。其他除了Name和Tag外的重要属性是LifeCycle。默认是Session, 但是也可设置为Server或Invocation。从长到短, Server, Session和Invocation的意思是一个类的实例在服务端的生命周期为整个服务, 一个DataSnap会话或一次方法调用。Session表示每个连接将获取其自己的服务类实例。如果将其改为Invocation, 将会得到一个无状态的服务类—可用于部署CGI Web服务应用程序(其也是无状态的, 每个请求都进行加载卸载)。将LifeCycle改为Server, 则所有的连接请求使用一个服务类实例。这可以用于计算请求数量, 但是必须自己保证线程安全。

TDSServerClass有四个事件: OnCreateInstance, OnDestroyInstance(当实例创建和注销时触发)。OnGetClass和OnPrepare。OnPrepare事件可用于准备服务方法。使用D2009或使用D2010手动向容器中添加TDSServerClass时, OnGetClass事件必须由我们自己实现, 以便于指定一个可远程调用的类。在D2010的向导中, 已经自动为我们实现了OnGetClass事件, 如下:

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

注意: 当我们重命名了自动生成的代码单元ServerMethodsUnit1为ServerContainerUnitDemo.pas后必须修改这里。

2.1.1.1.3. TDSTCPSERVERTRANSPORT

TDSTCPServerTransport组件负责在DataSnap服务端和客户端进行通讯, 使用TCP/IP协议。

TDSTCPServerTransport组件有五个重要的属性: BufferKbSize, Filters (D2010新特性), MaxThreads, PoolSize, Port和Server。

BufferKbSize属性指定通讯缓冲区大小, 默认设置为32KB。Filters属性可以包含一个传输过滤器集合, 将在第四节讲解。MaxThreads属性定义最大线程数(默认为0不限制)。PoolSize可用于连接池(如果修改了这里, 也需要相应的修改DataSnap客户端)。

Server属性指向TDSServer组件。TDSTCPServerTransport组件没有事件。

2.1.1.1.4. TDSHTTPSERVICE

TDSSHttPService组件负责使用HTTP协议组织DataSnap服务端和客户端通讯。

TDSSHttPService组件有十个属性(除了Name和Tag): Active, AuthenticationManager, DSHostName, DSPort, Filters, HttpPort, Name, RESTContext, Server, 和只读的ServerSoftware属性。

Active属性指定DSHTTService开始侦听请求。可以在设计时设置, 但是这会影响到DataSnap服务在运行时启动(由于DSHTTService组件侦听了同一个端口—在设计时启动侦听, 在运行时就不能再启动一个侦听了)。最好的方式是在TServerContainer的OnCreate事件中激活TDSSHttPService:

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
  DSHTTService1.Active := True;
end;
```

AuthenticationManager属性用于定义处理HTTP验证的管理组件, 这里指向了TDSSHttPServiceAuthenticationManager组件。这个组件在下节详述。

DSHostName和DSPort属性用于定义DataSnap服务端连接, 但只有在没有指定Server属性时生效。

通常都是使用Server属性.

Filters属性可以包含一系列传输过滤器, 在第四节详述.

HttpPort属性定义DSHTTPService组件侦听的特定端口以响应连接. 注意这个属性默认是80端口, 通常在发布时必须做修改(IIS等Web服务已占用了80端口).

RESTContext属性指定REST上下文URL, 这样就可以以REST服务的方式调用DataSnap服务. 默认, RESTContext属性设置为rest, 我们可以用<http://localhost/datasnap/rest/>... 来调用服务. 在第六节详述REST, JSON和客户端回调函数.

最后, Server属性指向同一个容器中的TDSServer组件. 如果没有指定Server属性, 也可以使用DSHostName和DSport属性连接到使用TCP的DataSnap服务. 当设置了Server属性, DSHostName和DSport属性失效.

TDSHTTPService组件有五个事件: 四个是REST相关的, 一个是跟踪事件, REST相关事件将在第六节详述.

OnTrace事件可用于跟踪对DSHTTPService组件的调用, 例如:

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

注意HTTP跟踪信息只有当客户端使用HTTP连接到服务端是才会触发(默认使用TCP/IP协议). 跟踪输出如下所示:

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

从中可见, AContext设置为TDSHTTPContextIndy, ARequest设置为/DataSnap/tunnel, AResponse设置为OK.

2.1.1.1.5. TDSHTTPSERVICEAUTHENTICATIONMANAGER

当选中HTTP通讯协议的验证复选框后, TDSHTTPServiceAuthenticationManager组件将自动出现在服务容器中. 也可手动添加到服务容器中, 当然TDSHTTPService组件的AuthenticationManager属性必须指向TDSHTTPServiceAuthenticationManager组件.

TDSHTTPServiceAuthenticationManager组件有一个事件: OnHTTPAuthenticate事件, 可以验证DataSnap客户端到服务端连接的HTTP信息.

```
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;
```

当然, 你可以使用数据库技术来扩展验证方式. HTTP验证使用客户端发送到服务端的用户名和密码信息最好使用HTTPS方式, 所以我希望易博龙可以在现有HTTP和TCP/IP基础上在添加一个HTTPS协议.

HTTPS可以确保连接安全和数据包加密, 数据包被窃取也不会泄露用户和密码信息. 与你所在域

的ISP或Web管理员协商是否有可能使用HTTPS——强烈推荐(如我使用的是<https://www.bobswart.h1>).

DataSnap服务应用程序另一个好的方法是将HTTP验证信息(所用协议和上下文信息)记录下来. 以便于查找谁登陆过, 谁试图登录(如做欺骗登录操作试图获取DataSnap服务).

2.1.1.2. SERVERMETHODSUNITDEMO

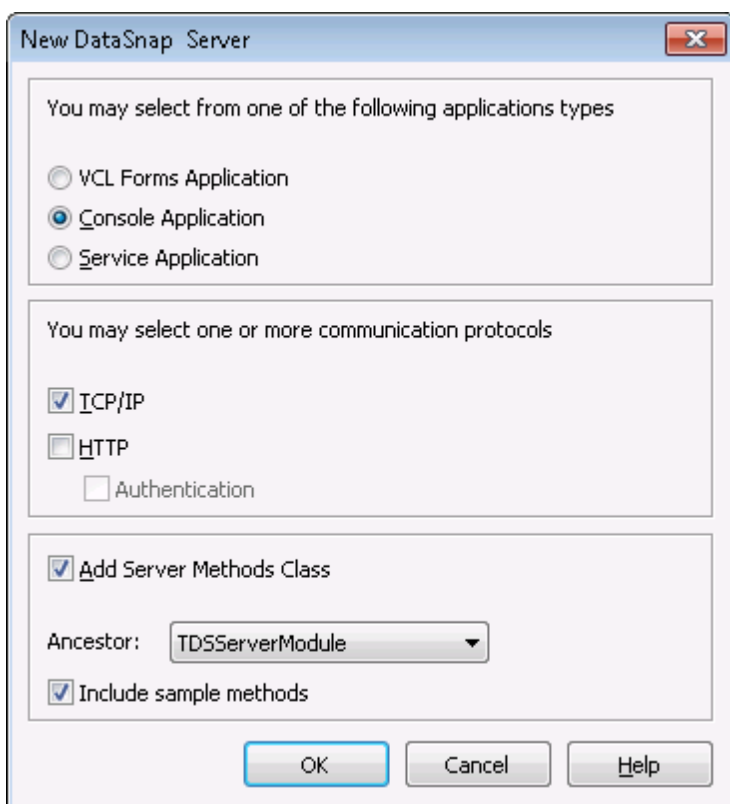
注意我们已经检查了ServerContainerUnitDemo. pas单元, 现在看一下DataSnap服务的另外重要的单元:ServerMethodsUnitDemo. pas单元. 在新建DataSnap服务对话框中我们指定了TDSServerModule类作为基类, 因此TServerMethods1类型继承于TDSServerModule(其继承于TDSServerModuleBase, 又继承于TProvideDataModule, 添加一个析构函数和BeforeDestruction过程. TProvideDataModule继承于正常的TDataModule, 增加了运行提供者的能力 更多信息见后面).

由于TServerMethods1继承于TDataModule, 设计时可以看到一个数据模块的可视区域. 我们可以向其中添加一下不可视组件, 如数据存取控件等. 在第三节将操作数据库, 现在保持设计区为空, 仅向TServerMethods1类添加方法.

如果不想向项目组添加其他类型的项目—DataSnap控制台应用程序及DataSnap Windows服务应用程序, 可跳到2. 1. 4 节查看实现服务方法.

2.1.2. 创建多目标项目组—控制台

再次回到项目组, 添加第二个项目: 控制台项目. 右击项目组节点, 选择添加新项目. 在Object Repository中选择DataSnap服务目录, 双击DataSnap服务图标. 这次我们在新建DataSnap服务对话框中选择控制台应用程序.



注意这里其他选项都不重要, 我们将重用DataSnapServer项目中的ServerContainerUnitDemo. pas和ServerMethodsUnitDemo. pas单元.

点击Ok按钮生成新的项目. 默认项目名称为Project1. dproj, 自动生成一个ServerContainerUnit2. pas和ServerMethodsUnit2. pas. 在项目管理器中右击ServerMethodsUnit2. pas, 将其从项目中移除. 保留ServercontainerUnit2. pas文件, 因为我们需要复制里面的一个方法. 保存项目为DataSnapConcoleServer. dproj.

如果比较一下ServerContainerUnitDemo.pas和新生成的ServerContainerUnit2.pas文件, 将会发现后者有一个叫做RunDSServer的全局函数. 这个全局函数是在控制台总需要使用的, 将其拷贝到DataSnapConsoleServer.dproj中, 放在begin主块前. 这时可以看到几处错误, 解决方法是向DataSnapConsoleServer.dpr中添加Windows单元引用. DataSnapConsoleServer如下所示:

```
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.

不幸的是我们先在要做三个动作解决错误:DataSnapConsoleServer项目还引用这ServerContainerUnit2.pas, 我需要移除它(右键点击选择移除, 右键DataSnapConsoleServer项目阶段, 选择添加, 选中ServerContainerUnitDemo.pas和ServerMethodsUnitDemo.pas单元将他们添加到项目中).

这时, 所有对TServerContainer2的引用都显示了语法错误. ServerMethodsUnitDemo.pas中定义了TServerContainer1类型, 为了修复最后一个错误:修改TServerContainer2为TServerContainer1(总共与三处).

现在可以编译这个新的DataSnapConsoleServer.dproj项目和原来的DataSnapServer.dproj项目. 两个项目间共享了ServerContainerUnitDemo.pas和ServerMethodsUnitDemo.pas.

2.1.3. 创建多目标项目组—Windows服务

在项目组中已经有了VCL窗体服务和控制台服务项目, 现在添加另一个项目:DataSnap Windows服务应用程序. 右键点击项目组节点, 选择添加新项目, 在Object Repository中双击New DataSnap Server图标. 这次选择Service应用程序并选择所有的通讯协议(TCP/IP, HTTP和HTTP验证). 在下面我们将看到Windows服务应用程序的服务容器和基于VCL窗体和控制台的应用程序有明显的不同. 所以这里我们要选择所有的通讯协议. 向导结束后自动生成一个新项目和ServerContainerUnit, ServerMethodsUnit. ServerMethodsUnit单元和原来生成的一样, 我们将之移除, 并添加ServerMethodsUnitDemo.pas文件替代.

新的ServerContainerUnit1.pas单元有很多不同之处:取代在TDataModule中放置TDSServer, TDSServerClass和通讯组件, 而是使用继承于TService的类来放置这些DataSnap组件. 除了继承于TService, 还有几个特殊方法, 实现服务的Stop, Pause, Continue和Interrogate事件:

type

```
TServerContainer3 = class(TService)
  DSServer1: TDSServer;
  DSTCPServerTransport1: TDSTCPServerTransport;
  DSHTTPServicel: 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;
```

换句话说, 我们不能使Windows服务项目和VCL窗口项目共享同一个ServerContainerUnitDemo.pas单元, 我们需要将ServerContainerUnit1.pas重命名为ServerContainerUnitServiceDemo.pas文件, 并将项目文件命名为DataSnapServiceServer.dproj.

我们需要修改一下ServerContainerUnitServiceDemo.pas中对ServerMethodsUnit2的引用, 改为对ServerMethodsUnitDemo.pas的引用, 最终我们将同一个服务方法在三个项目中共享.

2.1.4. 服务方法

当你有了一个或多个DataSnap服务项目, 都共享同一个ServerMethodsUnitDemo单元, 现在要检验一下服务方法了.

正如上面提到的, TServerMethod1类是一个发布方法(使用RTTI)供客户端调用的DataSnap服务对象. 如果你选中了'包括服务方法'选项, 将会有有一个范例方法包含在TServerMethod1中: function EchoString. 现在定义另一个, 返回DataSnap服务器的当前时间. 修改tservermethod1类包含两个public方法:

type

TServerMethods1 = **class**(TDSServerModule)

private

{ Private declarations }

public

{ Public declarations }

function EchoString(Value: string): string;

function ServerTime: TDateTime;

end;

实现ServerTime方法非常简单, 和EchoString一样简短.

function TServerMethods1.EchoString(Value: string): string;

begin

Result := Value;

end;

function TServerMethods1.ServerTime: TDateTime;

begin

Result := Now

end;

现在编译并运行服务应用程序. 如果你有多个目标项目, 最简单的就是直接使用DataSnapServer可执行项目. 基于操作系统安全级别的设置, 可能会弹出一个安全提示对话框, 让你允许DataSnapServer应用程序访问网络.

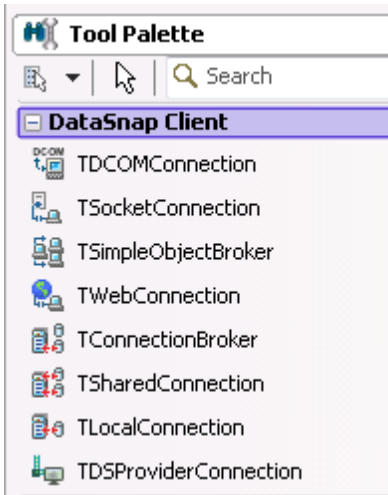


这时DataSnapServer应用程序开始侦听基于TCP/IP和HTTP的请求. 点击允许按钮继续启动DataSnap服务.

2.2. DATASNAP 客户端

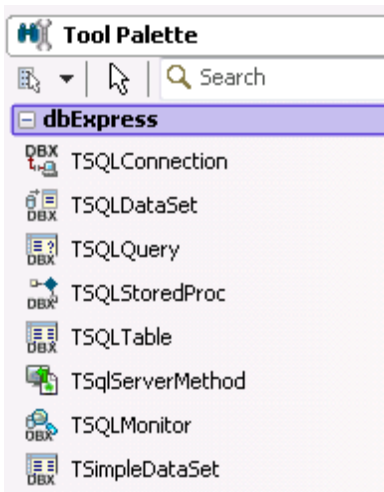
第一个DataSnap服务端范例启动并侦听请求, 现在创建一个客户端. 本节中, 我们将讲解如何在客户端连接服务端, 如何导入方法生产服务类.

确信DataSnap服务已启动, 我们创建一个DataSnap客户端应用程序项目. 为了在设计时方便切换DataSnap服务项目和DataSnap客户端项目, 将客户端项目添加到同一个项目组. 任何类型的项目都可作为DataSnap客户端项目, 这里我们选择VCL窗口应用程序, 保存为DataSnapClient.dpr, 主窗体为ClientForm.pas. 控件栏中DataSnap服务目录中含有六个DataSnap(服务)组件, DataSnap客户目录中不含我们现在需要的组件.

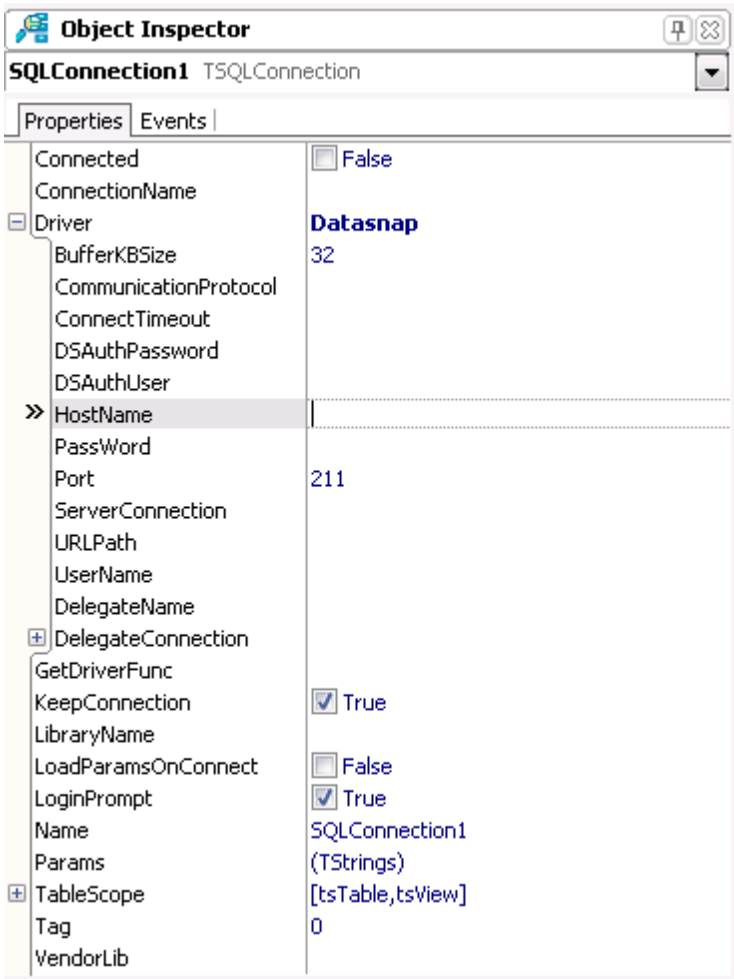


DataSnap客户端目录中的组件都是一些老的数据Snap组件, 可以使用, 但不推荐. 但可使用新的TDSPProviderConnect组件, 可以在新的DataSnap客户端上连接老的数据Snap服务(3.2中详述).

除了DataSnap客户端目录, 我们还有看看dbEpress目录, 可以找到一个新组件叫做TSQLServerMethod(注意: 在下一个截图中这个新组件很容易发现, 其用TSql前缀替代了TSQL前缀).



TSqServerMethod组件可用于调用DataSnap服务的远程方法,但首先需要连接到DataSnap服务. 可以使用TSQLConnect组件建立连接—不在使用原来的TXXXConnection组件. 为了灵活性, TSQLConnection组件的Drive属性下拉框中有个新选项:DataSnap. 在ClientForm上放置一个TSQLConnection组件, 设置其Driver属性为DataSnap. Driver属性将变成可展开的对象, 展开后如下图:



注意:默认CommunicationProtocol属性为空, TSQLConnection将使用TCP/IP作为通信协议(端口211). BufferKBSize为32(KB), 端口设置为211(与服务端设置相同). 实际应用中一般将端口号设置为其他端口(同时修改服务端客户端), 因为211端口是DataSnap默认端口不安全. HostName, UserName和Password用于连接到DataSnap服务. 在本地测试时, 将HostName设置为localhost, 通常可以设置为服务器名称, DNS或IP地址. 不要忘记将TSQLConnection的LoginPropt属性设置为False, 否则将会在连接的时候弹出登录窗

口.

一旦TSQLConnection的Driver属性设置好,就可以设置Connected属性为True去连接DataSnap服务端. 注意这时DataSnap服务必须运行才能连接成功.

2.2.1. DATASNAP客户端类

在确定连接正常后,右击TSQLConnection组件,选择Generate DataSnap Class选项,将生成一个新单元,默认叫做Unit1,含有一个叫做TServerMethods1Client的类(在DataSnap服务方法类名称后加了一个Client). 保存单元为ServerMethodsClient.pas.

这个TServerMethods1Client类如下所示

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;
```

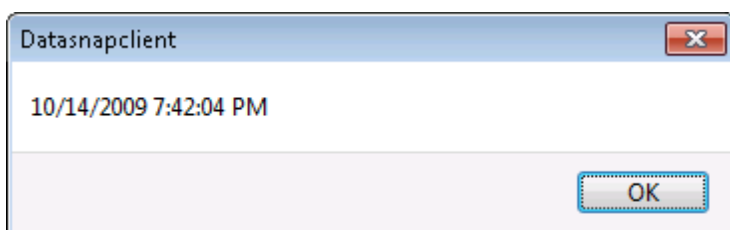
如你所见, TServerMethods1Client类有两个构造方法, 一个析构方法和两个我们在服务端定义的服务方法.

为使用这些方法, 将ServerMethodsClient单元引用到ClientForm, 在客户端窗体上放置一个按钮, 在按钮的OnClick事件中写如下代码:

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

这个代码将创建一个TServerMethods1Client类实例, 然后调用ServerTime服务类, 最后释放这个DataSnap服务的代理对象.

点击按钮将弹出对话框显示服务端的时间.

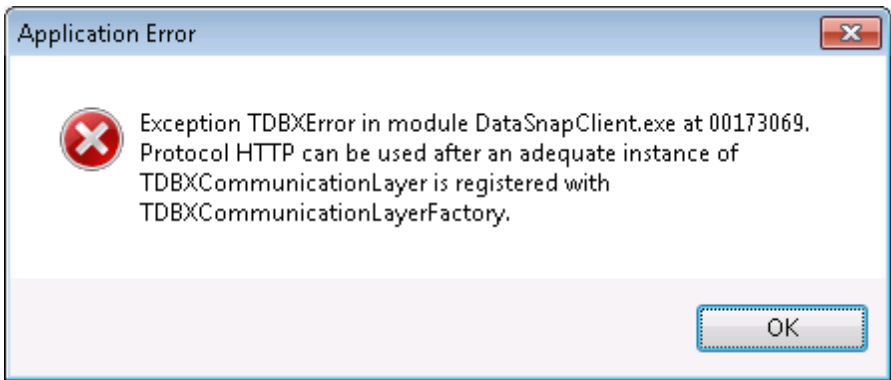


同样方法测试EchoString.

2.2.1.1. HTTP COMMUNICATION PROTOCOL

注意我已经提到TSQLConnection组件以TCP/IP作为默认通讯协议. 这样我们就看不到任何HTTP跟踪信息(在2. 1. 1. 1. 4小结中定义). 然而, 很容易修改通讯协议, 仅需在TSQLConnection的Driver属性中的CommunicationProtocal子属性中输入HTTP即可. 注意这是我们还要修改Port属性, 因为211被TCP/IP占用, 同时要修改服务端的TDSHTTPService组件的端口号.

修改后运行DataSnap客户端, 会出现如下错误:

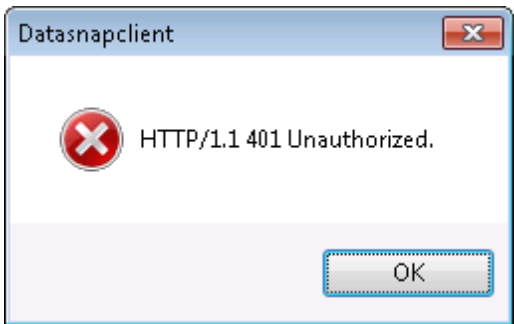


解决方法是在DataSnap客户端向ClientForm中添加DSHTTPLayer单元引用.

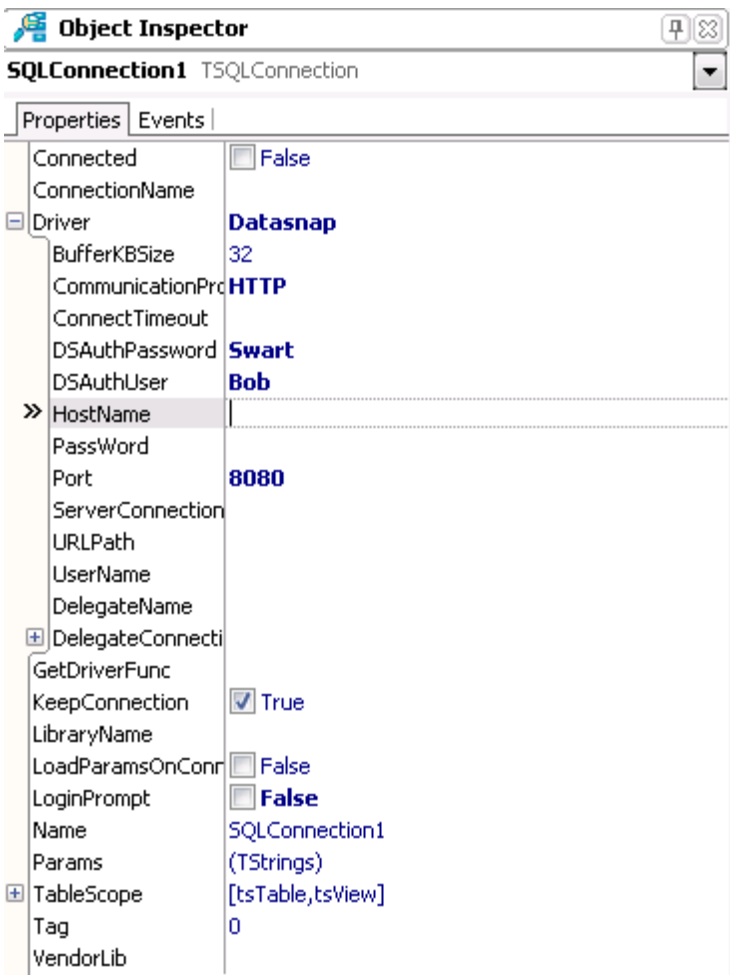
2.2.1.2. HTTP 验证

使用HTTP通讯协议的好处之一是可以使用其包含的HTTP验证. 有DataSnap服务端的TDSHTTPServiceAuthenticationManager组件所支持(详见2. 1. 1. 1. 5).

如果实现了OnHTTPAuthenticate事件处理, 将会核对HTTP验证. 我们必须保证输入正确的信息才能确保TDSHTTPServiceAuthenticationManager验证通过. 否则将会得到一个HTTP/1.1 401未验证错误.



HTTP验证从客户端传递到服务端的用户名和密码, 及其他TDSHTTPServiceAuthentication特定信息, 我们需要在DataSnap客户端的TSQLConnection控件中填写DSAuthUser和DSAuthPassword属性.



注意我们也需要指定HostName的值, 除非是在同一台电脑上测试.

2.3. DATASNAP服务部署

范例的服务端和客户端在同一台电脑上运行良好, 但是实际环境中, DataSnap服务将运行在服务器上, 一或多台客户端通过网络连接服务端. 服务端程序通常部署在没有安装Delphi的电脑上. 这种情况下就需要考虑不用运行时包来编译DataSnap, 而仅生成一个大的可执行文件. 由于我们还没有使用任何数据操作组件, 也不需要任何其他数据库驱动或DLL文件.

2.3.1. DATASNAP 客户端部署

假设客户端与服务端运行在不同的电脑上, 我们必须保证客户端可以连接到服务端. 为了保证这点, 客户端的TSQLConnection组件不能仅仅指定Driver属性中的CommunicationProtocol和端口, 还需要指定HostName属性. 设置IP地址或DNS. 例如连接我的DataSnap服务的HostName属性为 www.bobswart.nl. (注意不需要指定http://前缀, 因为在CommunicationProtocol属性中指定了通讯协议).

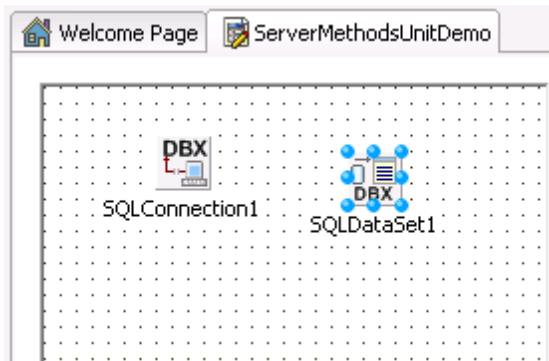
3. DATASNAP和数据库

除了使用Delphi2010 DataSnap框架创建简单的服务方法, 我们还可以在服务端添加数据库操作, 实现多层数据库应用—DataSnap服务连接到数据库, 但DataSnap客户端是瘦客户端, 不含任何数据库驱动.

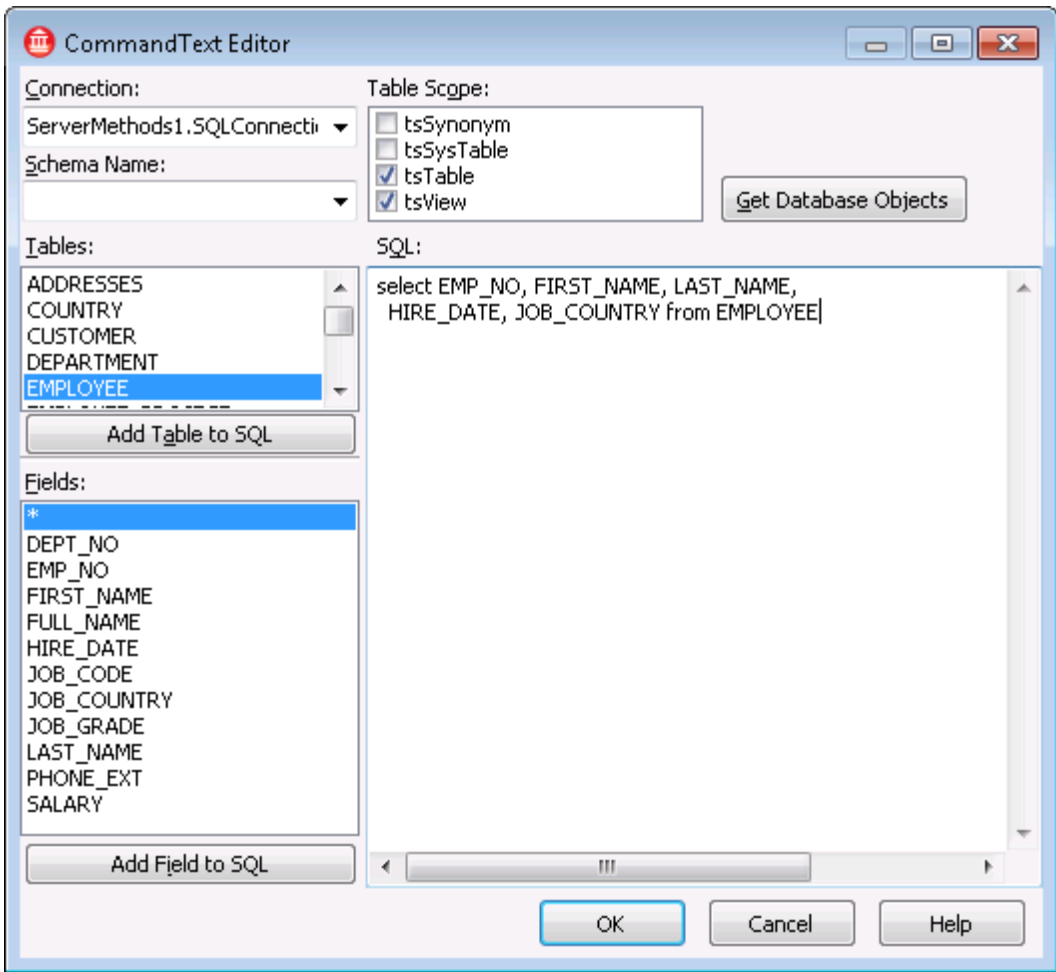
我们同样可以快速创建数据库操作的DataSnap范例, 只使用SQLConnection组件和以创建的客户端类. 也可使用另外两个DataSnap组件TsqlServerMethod和TDSProviderConnection.

首先, 我们要确保服务端开发了一个TDataSet, 打开ServerMethodsUnitDemo并在数据模块中添加一个TSQLConnection组件. 连接TSQLConnection组件到数据库和表(本例我连接到BlackFishSQL的

Employees表). 将TSQLConnection组件添加到数据模块的设计区域. 设置Driver属性为BlachFishSql. 然后设置Database属性为employee.jds文件的路径: C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL on Windows XP, or C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL. 将TSQLConnection的LoginPrompt属性设置为False. 设置Connected属性为True验证是否能正确连接. 下一步, 添加一个TSQLDataSet组件, 将其SQLConnection设置为TSQLConnection组件.



设置其CommandType为ctQuery, 双击CommandText属性输入SQL语句.



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

确保在设计时设置了TSQLConnection组件的LoginPrompt和Connected属性为False, 及TSQLDataSet的Active属性为False.

现在在ServerMethodsUnitDemo单元的TServerMethods1类中添加一个public方法返回一个TSQLDataSet组件.

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;

```

如你所见, TServerMethods1类中定义了一个叫做GetEmployees的方法, 实现如下:

```

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

```

重新编译服务并运行. 注意如你关闭了服务单无法从新编译, 可能是DataSnap服务还在运行.

3.1. TSQLSERVERMETHOD

回到DataSnap客户端, TSQLConnection组件应该没有在连接到DataSnap服务端(如果仍然连接, 说明你没有重新编译服务端). 重新将Connected设置为True, 从服务端更新信息, 并重新从服务端生成客户端类(使用右键菜单).

为了重写原来的ServerMethodsClient单元, 推荐从项目中移除原来的ServerMethodsClient后在选择'Generate DataSnap Client Classes'方法. 保存了新的单元文件后不需要修改客户端代码. 生成了新文件后, 可以从中看到GetEmploreees方法.

```

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;

```

为使用GetEmployees方法获取TDataSet, 我们可以使用TsqlServerMethod组件. 将TsqlServerMethod组件添加到客户端窗体, 设置其SQLConnection属性为SQLConnection1, 然后打开ServerMethodName下拉框显示可用的方法: 一些DSAdmin方法(可通过设置TDSServer的HideDSAdmin属性为True禁止), 接下来是三个DSMetaData方法, 七个TServerMethods.As_XXX方法(由原来的IAppServer提供), 和最后我们自己的TServerMethods1的方法: EchoString, ServerTime, 和GetEmployees方法.

本例我们选择TServerMethods1.GetEmployees作为ServerMethodName属性的值. 根据ServerMethodName属性值, SqlServerMethod控件中包含查询结果记录集.

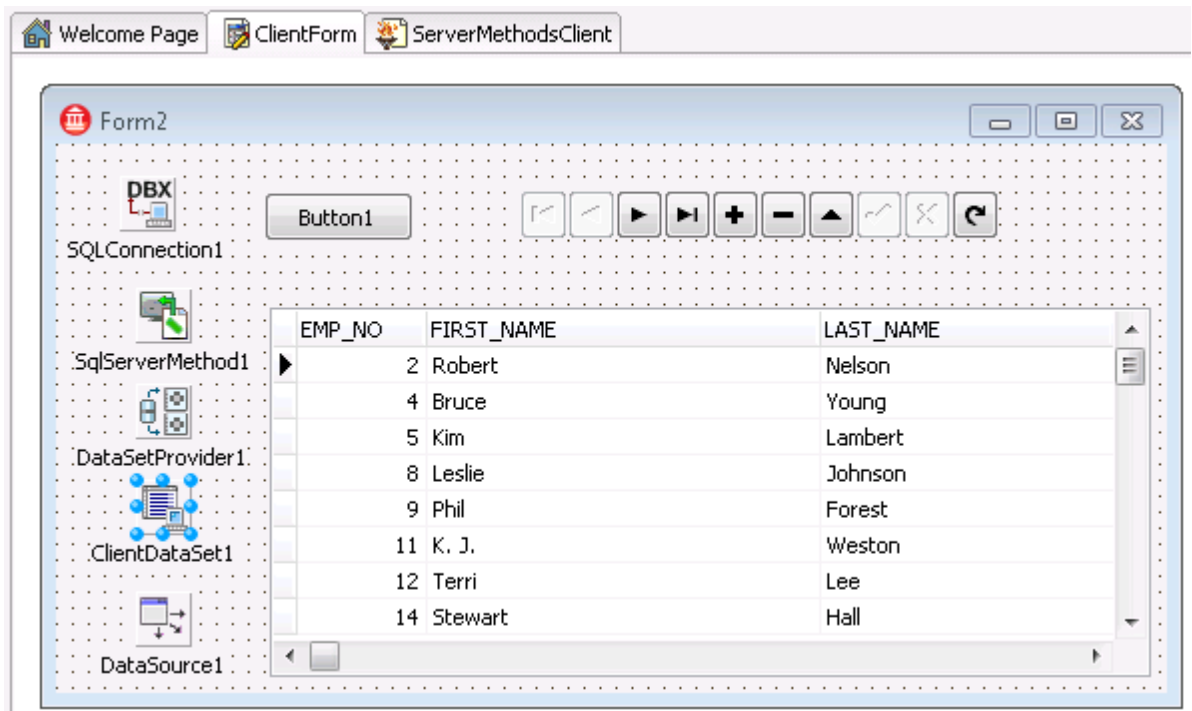
我们使用TDataSetProvider, TClientDataSet和TDataSource来讲数据显示在TDBGrid中.

在客户端添加一个TDataSetProvider控件, 将其DataSet设置为SqlServerMethod控件. 下一步添加一个TClientDataSet控件, 将其ProviderName设置为DataSetProvider控件. 设置其RemoteServer属性为空—这是用于原有DataSnap的方法, 在新的DataSnap架构中不再使用.

最后, 放一个TDataSource控件, 设置其DataSet为TClientDataSet控件, 然后就可以放TGBGrid和TDBNavigator控件了. 设置其DataSource为TDataSource组件, 将可以在客户端窗体中看到数据.

为了在设计时验证连接, 我们设置ClientDataSet的Active属性为True. 将触发SqlServerMethod的Active为True (开始检索数据, 然后自动将Active设置为False), 同时会将TSQLConnection.Connected设置为True连接到服务端.

这种方式连接一直保持, TClientDataSet和TSQLConnection的被激活, 数据显示在TDBGrid中.



这种方式很容易以只读方式查看数据. 注意这里说只读, 因为TSQLServerMethod不允许TDataSetProvide-TClientDataSet组合将客户端的数据变更传回服务端.

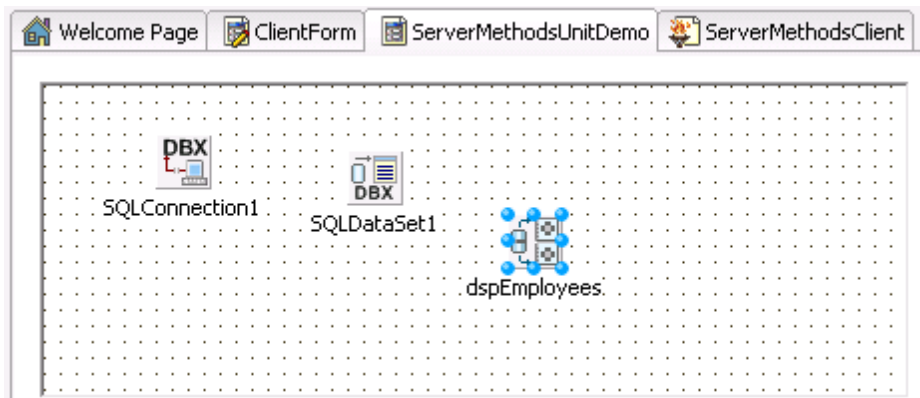
TSQLServerMethod是轻量级方式连接获取只读数据. 这样, 如果你需要获取DataSnap服务器上的数据但不需要做修改, 当前这种发布一个返回TSQLDataSet结果集方法的架构就很好.

然而, 有时我们需要更新数据, 这是就必须使用不同的方法获取数据.

3.2. TDSPROVIDERCONNECTION

如果我们想提交更新, 我们就需要TDSPProviderConnection组件与服务端的TDataSetProvider关联, 因此我们不但可以读取数据也可以修改数据.

首先, 我们需要虚构服务端数据模块, 然而, 现在我们只是返回一个TDataSet, 但是我们必须添加一个实际的TDataSetProvider, 确保将其从DataSnap服务端发布到客户端. 所以, 回到ServerMethodsUnitDemo单元并放一个TDataSetProvider组件, 设置其DataSet属性为TSQLDataSet. 应该将TDataSetProvider重命名, 如dspEmployees;



现在, 重新编译DataSnap服务, 运行. 然后修改客户端.

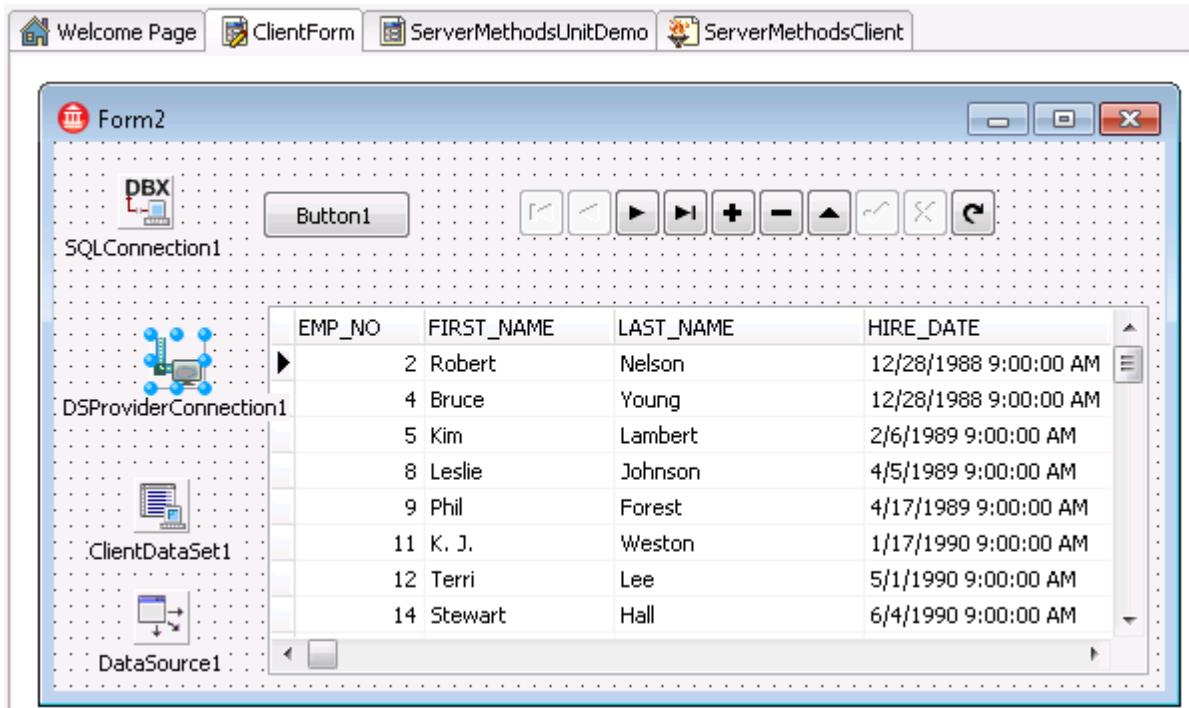
3.2.1. TDS PROVIDER CONNECTION 客户端

为检索到发布的TDataSetProvider组件需要修改DataSnap客户端. 将TSQLServerMethod和TDataSetProvider组件从客户端窗体删除, 添加一个TDSProviderConnection组件. 设置其SQLConnection属性为TSQLConnection组件, 连接到DataSnap服务. 我们也需要设置一个ServerClassname属性值(很不幸不能选择只能输入). 现在只能手动输入TDSServerModule的名字, 这里是TServerMethods1.

在前一个例子中, TClientDataSet只设置了一个ProviderName属性. 然而, 使用TDSProviderConnection组件, 我们必须首先设置其RemoteServer属性为TDSproviderConnection组件, 然后设置ProviderName属性(这个属性还为原料设置的DataSetProvider1, 现在设置为dsEmployeeer----在服务端发布的TDataSetProvider组件的名称).

在ProvideName属性的下拉框中显示dspEmployees选项(在服务端的ServerDataModule单元中发布的名称).

现在我们可以设置TClientDataSet.Active为True, 可以在设计时查看数据.



设置TClientDataSet.Active为True, 同时将TSQLConnection.Connected置为True. 注意在设计时将这两个属性设置为True不好. 首先, 如果你在IDE中打开DataSnap客户端项目, 将会视图连接服务端, 如果服务端没开启将失败. 其次, 如果在运行时启动应用程序, 并且连接不可用, 应用程序将会抛出异常. 从而应用程序不能使用本地数据, 如果远程连接无效将不能使用应用程序.

最好的方式是使用菜单选项或按钮明确的设置TSQLConnection组件进行连接, TClientDataSet组件进行获取数据. 并提交包括username/password的信息, 将在后面说明. 现在确保TClientDataSet.Active属性设置为False, 同时TSQLConnection.Connected属性为False. 在客户端窗体中放一个按钮, 在OnClick事件中明确的打开TClientDataSet.

```
procedure TForm2.Button2Click(Sender: TObject);
```

```
begin
```

```
  ClientDataSet1.Open;
```

```
end;
```

现在添加代码提交时间变更, 保存会服务器.

3.2.2 数据库更新

有两种方法将数据修改保存会服务端: 自动和手动. 都是调用一下方法, 但是会自动调用或手动调用, 各有优缺点.

对于自动方式, 我们可以使用TClientDataSet的数据修改时触发的OnAfterInsert, OnAfterPost和OnAfterDelete事件. 在事件处理程序中, 实现很简单, 调用TClientDataSet的ApplyUpdates方法, 发送变更, 将Delta包发送到服务端保存回数据库.

```
procedure TForm2.ClientDataSet1AfterPost(DataSet: TDataSet);
```

```
begin
```

```
  ClientDataSet1.ApplyUpdates(0);
```

```
end;
```

如果发生了更新错误, 将会触发TClientDataSet的OnReconcileError事件, 更多信息见3.2.3

手动方式发生更新也是使用TClientDataSet的ApplyUpdates方法. 但是这时方法不在OnAfterInsert, OnAfterPost和OnAfterDelete事件中执行. 而是我们添加一个按钮让用户显示的提交更新.

```
procedure TForm2.btnUpdateClick(Sender: TObject);
```

```
begin
```

```
  ClientDataSet1.ApplyUpdates(0);
```

```
end;
```

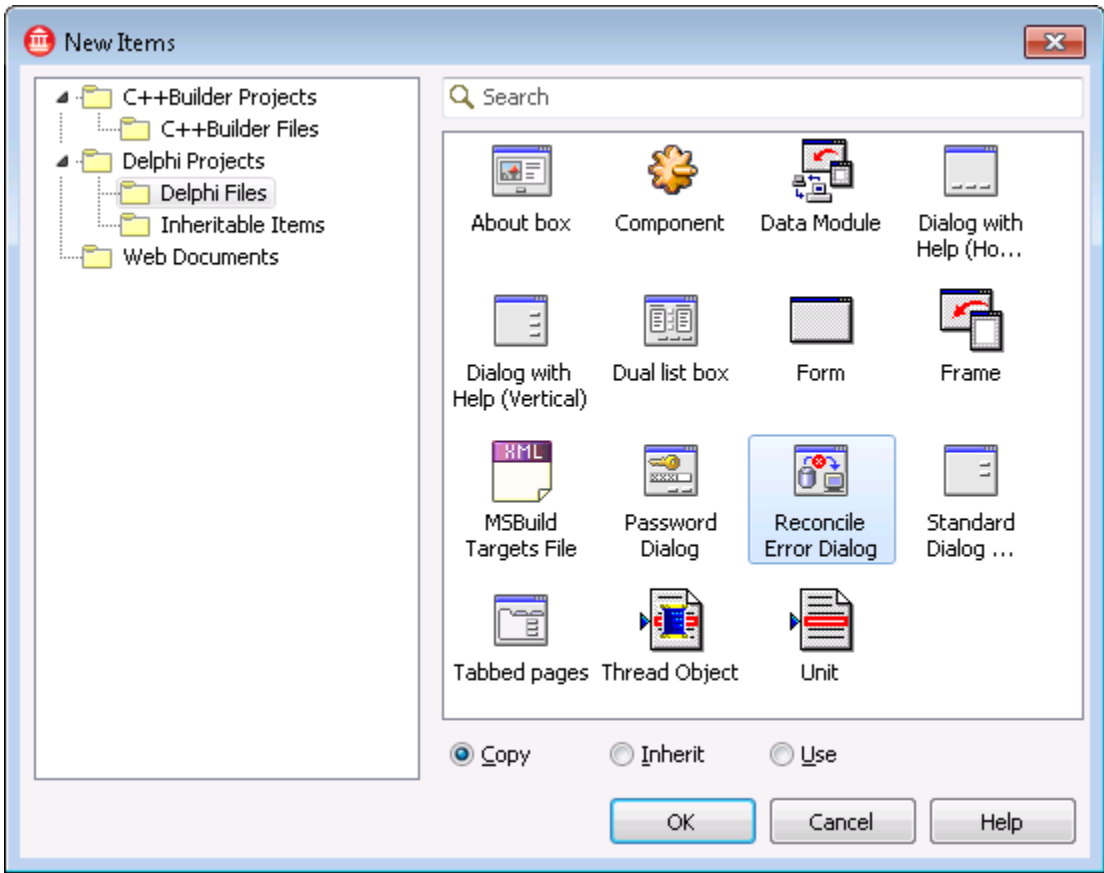
自动提交的好处当然用户不会忘记将变更保存回服务端. 然而, 缺点是无法提供Undo能力. 一旦提交数据就更新会了服务器. 另外, 如果使用手动提交, 所有变更保存在客户端——TClientDataSet组件的内存中. 这样就允许用户Undo部分变更: 特定记录或全部记录放弃更新. 点击更新按钮显示调用ApplyUpdate方法. 可能会导致用户忘记提交修改数据. 我们应该在传统关闭是中添加代码检查TClientDataSet中是否还有未提交数据(检查TClientDataSet.ChangeCount属性).

3.2.3. RECONCILE ERRORS

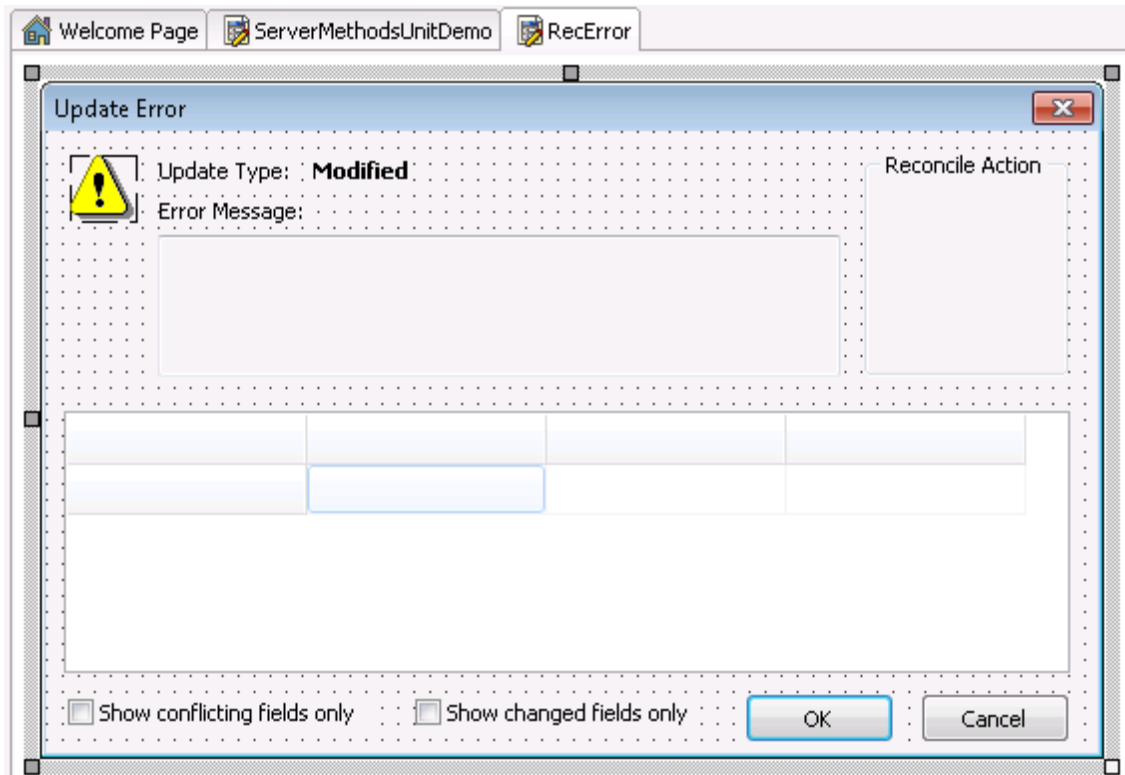
TClientDataSet.ApplyUpdates方法有一个参数: 应用更新时允许发生的最大错误数量. 如果有两个客户端连接到了DataSnap服务端, 获取Employees数据并同时修改了第一行数据. 依据目前为止我们的实现, 两个客户端都会使用TClientDataSet的ApplyUpdates将数据变更到DataSnap服务端. 如果都将ApplyUpdates的参数MaxErrors设置为0, 则第二个客户端的提交将会停止. 第二个客户端应该使用一个大于0的参数指定允许的误差/冲突数. 然而, 即使第二个客户端将MaxErrors设置为-1(不管有多少错误发生都继续提交后面的更新记录), 都不会提交被第一个用户更新的记录. 换句话说, 你需要执行一系列冲突处理来解决这些更新已经被更新的记录或列的问题.

幸运的是, Delphi提供了一个很强大的对话框来处理这个问题. 当在DataSnap客户端需要做一些冲突处理时, 都可以使用这个对话框(或自己实现, 但最终都是处理冲突问题).

使用Delphi提供的功能, File→New→Other, 在Delphi文件子目录中选择Reconcile Error对话框图标.



选中这个图标点击OK, 新的RecError.pas加入到DataSnapClient项目. 这个单元包括了定义和实现更新错误对话框. 来解决数据库更新错误.



ReconcileErrorForm窗体实例将按需要动态创建.那么合适如何使用这个特殊的ReconcileErrorForm窗体呢?好,其实很简单.对于每个没有成功更新的记录,都会触发TClientDataSet的OnReconcileError事件.定义如下:

```
procedure TForm2.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
E: EReconcileError; UpdateKind: TUpdateKind;
```

```
var Action: TReconcileAction);
```

这个事件处理程序与四个参数,第一个是抛出错误的TClientDataSet,第二个参数是引发错误冲突的原因,第三个参数是更新类型UpdateKind(insert, delete, modify),第四个参数是你要如何处理冲突.可以返回如下枚举类型值:

- raSkip:不更新这条记录,但在变更日志中保留未提交的变更,下次提交在试试看.
- raAbort:取消记录冲突处理.所有记录都不向OnReconcileError事件中传递.
- raMerge:将更新记录与远程数据库记录合并,仅在客户端变更修改过的远程字段
- raCorrect:使用正确的值替换更新记录,这需要用户介入.
- raCancel:对本记录的修改全部放弃.回到初始值状态.
- raRefresh:对本记录修改全部放弃,但重新加载当前数据库的记录值.

关于ReconcileErrorForm不需要考虑全部执行选项.只需要多两件事件.一,在DataSnap客户端主窗体中引用Error对话框单元.二,在OnReconcileError事件中写一行代码调用ReconcileErrorForm单元中的HandleReconcileError函数. HandleReconcileError函数也有四个同样的参数,只需要按顺序传递即可.如下所示:

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

3.2.4. 示范冲突错误

现在最大的问题是:实际中是如何工作的?为了测试,需要两个或更多DataSnap客户端同时运行.为使用当前的客户端和服务端进行测试,需要执行如下步骤:

- 启动服务端应用程序
- 启动第一个客户端应用程序,点击链接按钮,获取数据
- 启动第二个客户端应用程序,点击链接按钮,获取数据
- 使用第一个客户端应用程序,修改第一行数据的FirstName列
- 使用第二个客户端应用程序,修改第一行数据的FirstName列
- 在第一个客户端应用程序中点击更新按钮
- 在第二个客户端应用程序中点击更新按钮,这时将会发生一个或多个错误.因为第一个应用程序已经修改了同一行的同一个列.引起冲突. OnReconcileError被触发.
- 进入更新错误对话框,现在可以处理冲突(忽略(Abort),取消(Abort),合并(Merge),更正(Correct),取消(Cancel),更新(Refresh)).测试一下Skip和Cancel的不同,及Correct,Refresh和Merge的不同.

Skip移动到下一行记录,忽略更新请求.为应用的更新将保留在更新日志中. Cancel也忽略更新请求同时清除本记录所有以前的更新记录.

Refresh清除本记录所有的变更记录,并将数据库中的值作为当前记录的值. Merge试图将数据库记录和更新记录合并. 将变更提交到数据库. 更新和合并的记录都不会再以后进行处理,记录已经于数据库同步.

Correct是一个强大的选项,在事件处理中给你一个自定义更新记录的机会.需要写代码或弹出对话框指定新值.

3.3. DATASNAP 数据库部署

部署一个使用数据库的DataSnap服务需要比部署一个简单DataSnap服务的步骤要多些. 客户端,没什么变化,还是一个瘦客户端,如将MidasLib加入到了项目引用就仅需部署一个单一的可执行文件.

服务端,必须部署数据库驱动.及所选数据库依赖的驱动和文件.使用DBX4,确保发布TSQLConnection组件和dbxconnections. ini及dbxgrivers. ini文件(可在C:\Documents and Settings\All Users\

Documents\RAD

Studio\dbExpress\7.0 directory on Windows XP or in the C:\Users\Public\Documents\RAD

Studio\dbExpress\7.0中找到). dbxdrivers.ini文件指定所用驱动, DriverPackageLoader及MetaDataPackageLoader(通常指向同一个包).对于BlackFishSQL,使用DBXClientDriver140.bpl,需要部署到服务端.更多关于部署BlashFishSQL信息见RAD Studio\7.0 目录下的deploy_en.htm文件.

3.4. 重用已有的远程数据模块

如果你有一个远程数据模块类,也可以将其组合到新的DataSnap项目中来.但是必须要牺牲一下特性,尤其是引入了COM.

首先,如果有一个你要迁移的DataSnap服务应用程序,而不仅仅是一个远程数据模块,你需要使用那个命令行/unregister命令注销DataSnap服务.不做这一步将无法注销远程数据模块.在远程数据模块单元,移除initialization区域.如果还希望这个单元在D2007及一下版本重用,可以使用编译开关:

```
{ $IF CompilerVersion >= 20 }
```

initialization

```
TComponentFactory.Create(ComServer, TRemoteDataModule2010,  
Class_RemoteDataModule2010, ciMultiInstance, tmApartment);
```

```
{ $IFEND }
```

end.

从项目中移除UpdateRegistry函数或用编译开关修饰:

```
{ $IF CompilerVersion >= 20 }
```

```
class procedure UpdateRegistry(Register: Boolean;
```

```
const ClassID, ProgID: string); override;
```

```
{ $IFEND }
```

最重要的变更—将项目转换为无COM的DataSnap服务.-及移除类型库(.ridl文件)和类型库导入单元.这无法通过编译器开关修改,因此需要为D2007及一下版本和D2009及以上版本分别生成一个项目文件.在

TRemoteDataModule类中放一个TDSServerClass组件.最后,将所有的自定义方法都转到

TRemoteDataModule单元的public节中.

4. DATASNAP 过滤器[FILTER]用法

本节将说明过滤器工作原理,及如何使用已存在的过滤器(如压缩)或创建新的DataSnap过滤器.DataSnap过滤器是一个特殊的DLL,拦截通讯流,在整个过滤器链中操作通讯流.所以本例中我们可以结合压缩和解压缩过滤器,或记录压缩等.

这里有两个地方需要指定客户端和服务端过滤器.在服务端,必须指定TDSTCPServerTransport组件的过滤器属性列表.在客户端,必须在客户端项目中引用过滤器列表.对于客户端这就足够了,因为每个DataSnap过滤器都会自动注册.

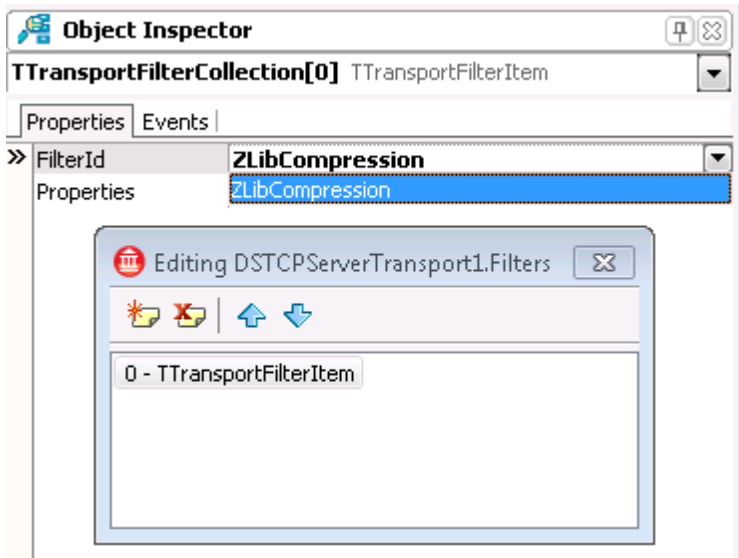
当处理OnConnect事件时,可以检查用于连接的已注册的过滤器,例如使用自定义的日志函数输入日志信息,

```
procedure TServerContainer1.DSServer1Connect(  
    DSConnectEventObject: TDSSConnectEventObject);  
  
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 FILTER

作为范例,我们使用已随D2010提供的DataSnap过滤器.可用于在客户端和服务端压缩数据流.这里说的ZlibCompression过滤器可以在DbxCompressionFilter单元找到.

TDSTCPServer和TDSHTTPService组件都有一个TTransportFiltersCollection类型的Filters属性.点击Filters属性后面的按钮,编辑过滤器列表.在这个对话框中,我们可以加一个新的TTransportFilterItem,然后在Object Inspector中设置FilterID和一些属性.在下拉框中可以找到Delphi2010提供的ZLibCompression过滤器.



注意除了设置服务端TDSTCPServerTransport组件Filters属性外,也需要在客户端指定一个相应的过滤器(压缩请求解压缩应答).这时,我们进需要将DbxCompressionFilter单元引入到ClientForm.其将自动注册一个TTransportCompressionFilter并与服务端通讯.

如果没有在客户端添加DbxCompressionFilter单元引用,运行客户端后将会抛出异常信息:



4.2. LOG FILTER

Delphi 2010 DataSnap 允许自定义传输过滤器.我们可以从TTransportFilter 类型继承自己的类.在这个新类中,可以重写基类中的方法,实现这些方法.例如我们创建一个TLogFilter类:

```
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'
```

这个类的实现很多都是空的:由于仅仅用于记录ProcessInput和ProcessOutput方法发生的数据,我们将很多方法都不用实现.非空方法如下:

```
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;
```

最后,重要的实现部分是在initialization和finalization小结注册DataSnap传输过滤器.确保客户端可以找到这个传输过滤器,并在请求时自动使用.

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

为了在DataSnap服务端使用这个传输过滤器,我们需要将其加入到TDSTCPServer或DSHTTPService组件的Filters属性中,非常简单.在设计时,已知存在一个ZLibCompression过滤器,但无法感知新的过滤器(除非将其添加到设计时包中并安装).幸运的是我们也可以在运行时添加过滤器,在ServerContainerUnitDemo单元中引用过滤器单元,然后手动向Filters属性中添加过滤器.如.

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
```

```
begin
```

```
    DSTCPServerTransport1.Filters.AddFilter(LogFilterName);
```

```
    DSHTTPService1.Filters.AddFilter(LogFilterName);
```

```
    DSHTTPService1.Active := True;
```

```
end;
```

这将确保服务端使用LogFilter,客户端只要在单元中引用LogFilter单元就回自动使用LogFilter.否则将抛出错误信息:



注意每个应用程序—DataSnap客户端和服务端—将获取各自的logfile.虽然使用的同一个过滤器,但是却不用使用ParamStr(0)来区分日志信息.

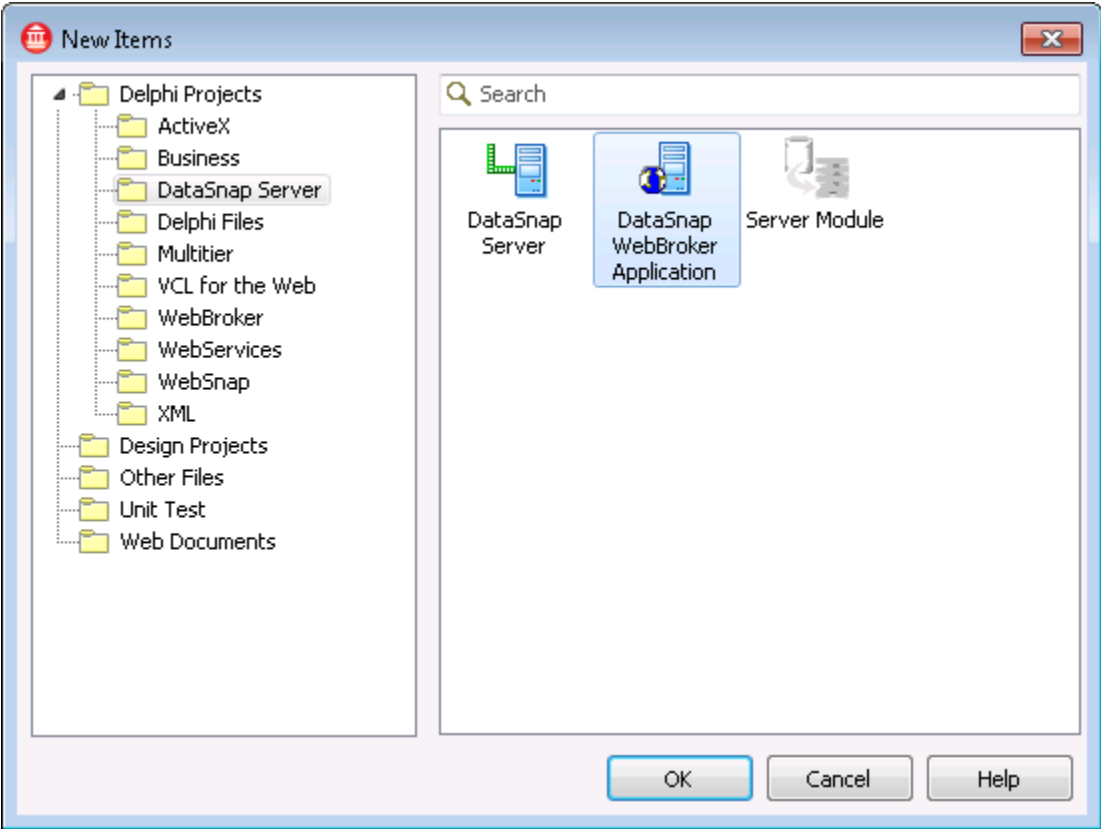
4.3. ENCRYPTION FILTER

即使是一个向4.2中的一个简单的过滤器,也需要自己去扩展,很复杂.DataSnap提供的过滤器不太完整.事实上,有大量的三方过滤器可以使用,有Daniele Teti开发的DataSnap Filters Compendium可以在<http://www.danieleteti.it/?p=168>中获取,其中不少于9个用于DataSnap2010的附加过滤器,分成三个组.Hash组支持MD5,MD4,SHA1和SHA512,Cipher组支持Blowfish,Rijndael, 3DES 和 3DES, Compress组支持LZO.并且是全源码版.

5. 如果构建DATASNAP WEB项目

除了生产Windows项目,还提供了生成ISAPI,CGI或Web App Debugger目标项目.首先我们讨论一下每种项目的优缺点,并展示如何在一个项目组中同时创建出这三个项目,并让他们共享公共的单元文件.这样我们就可以为同一个DataSnap项目产生三个不同的部署目标.

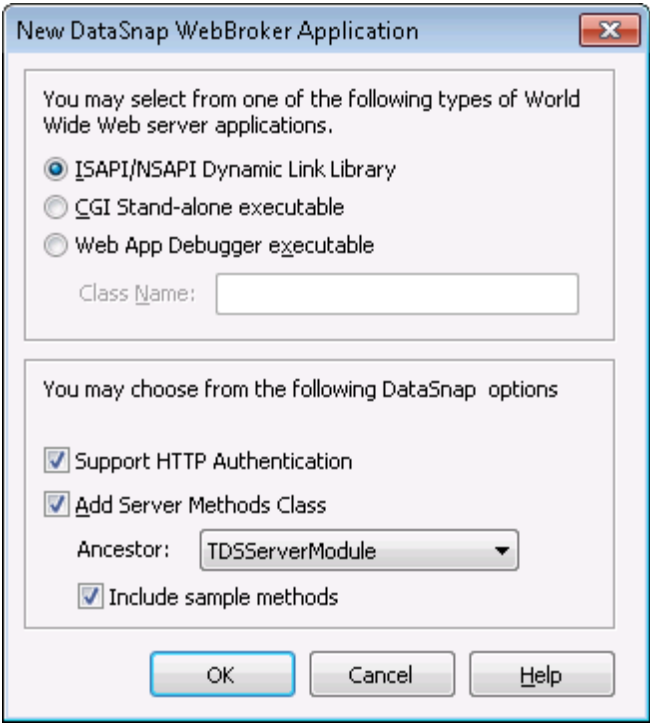
虽然到目前为止我们构建的DataSnap服务应用程序运行良好.但是在有些情况下还不能部署服务程序.例如如果你不能或不允许打开防火墙的请求端口让客户端连接服务器.幸运的是,这种情况下我们可以使用Web服务来部署,而80端口号一直都是打开的.如果我们使用IIS来作为Web服务器,我们就可以使用新的DataSnap WebBroker应用程序向导来创建一个可部署在IIS上的应用.



DataSnap WebBroker应用程序向导提供了三个选项,第一个选项实际上不是真正的WebBroker应用,但仅仅Web App Debugger客户端才可用于调试目的. Web App Debugger客户端很强大,允许我们使用Web App Debugger(Delphi IDE Toolies菜单中)作为调试Web App Debugger客户端应用程序的宿主程序.

调试CGI或ISAPI/NSAPI Web应用不方便因此在开发过程中最好选择Web App Debugger模式.

而ISAPI/NSAPI Dynamic Link Library 和 CGI Stand-alone executable 类型项目可用于真正部署的DataSnap服务项目上.



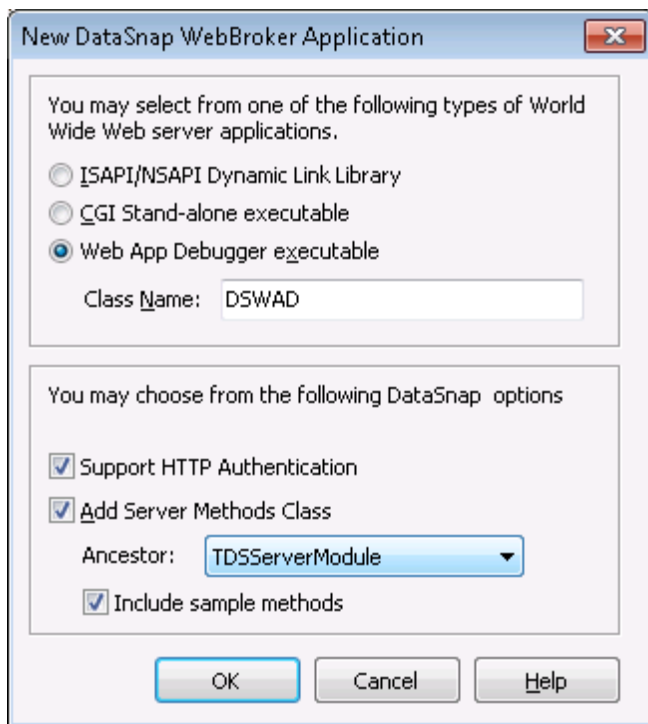
注意,选择CGI Stand-alone executable不是一个好主意,因为这个可执行文件将在每次请求中加载卸载.在加上要连接到数据库执行一些任务,你必须考虑到应用程序的执行效率.使用ISAPI的DLL形式,只需要加载一次,保存在内存中后续请求(可能来自其他用户)不需要再次加载ISAPI DLL的主要缺点是升级困难(如

果你用FTP连接到Web服务器).但可以联系Web服务供应商.

ISAPI DLL的另一个缺点是调试不方便—必须用IIS作为宿主应用,不能总是想计划的那样运行.但可以用Web App Debugger executable来解决这个问题—必须创建两个项目,他们使用公共的DataSnap方法和代码.第一个范例就是这种形式,加入一些使用的功能构建一个框架.

5.1. WEB APP DEBUGGER 项目

首先,使用新建DataSnap WebBroker Application向导创建一个DataSnap WebBroker应用程序.指定适当的类名,如DS-WAD并选中 Support HTTP Authentication选项.

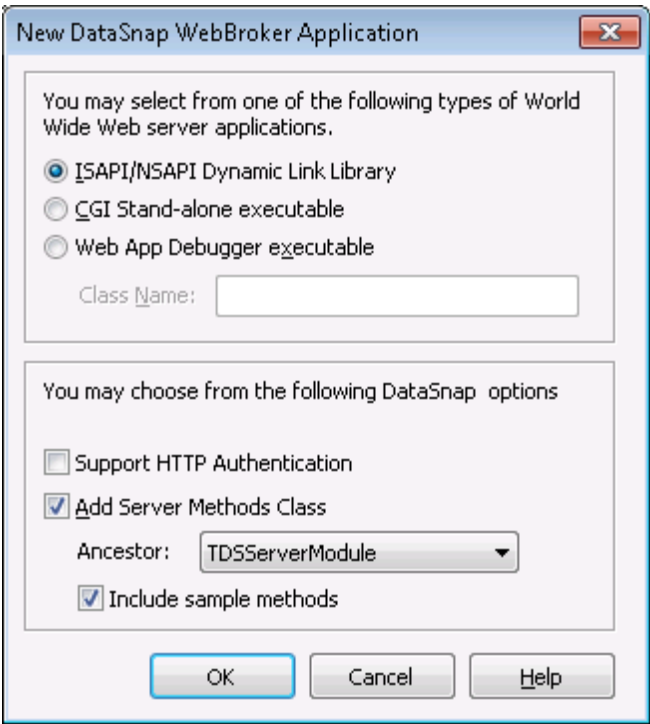


点击OK按钮,创建有三个单元的新项目.如果在默认目录中没有项目和单元被命名为Project1和Unit1,这个项目将被命名为Project1,单元文件分别被命名为Unit1, ServerMethodsUnit1和Unit2.第一个单元是一个空窗体—这个单元是Web App Debugger executable项目独有的,其他类型项目不需要.保存这个单元为WADForm.pas.第二个叫做ServermethodsUnit1.pas的单元包括了服务模块,继承自弹出对话框指定的TDSServerModule基类,稍后继续介绍这个单元,现在将其命名为ServerMethodsUnit1.pas.第三个叫做Unit2的单元是保护了四个组件(如果不选择HTTP支持只有三个组件)的Web模块.这个模块用来接收请求并将请求分布给DataSnap服务模块.保存为DSWebMod.pas.

最后,保存项目为DSWADServer.dproj.

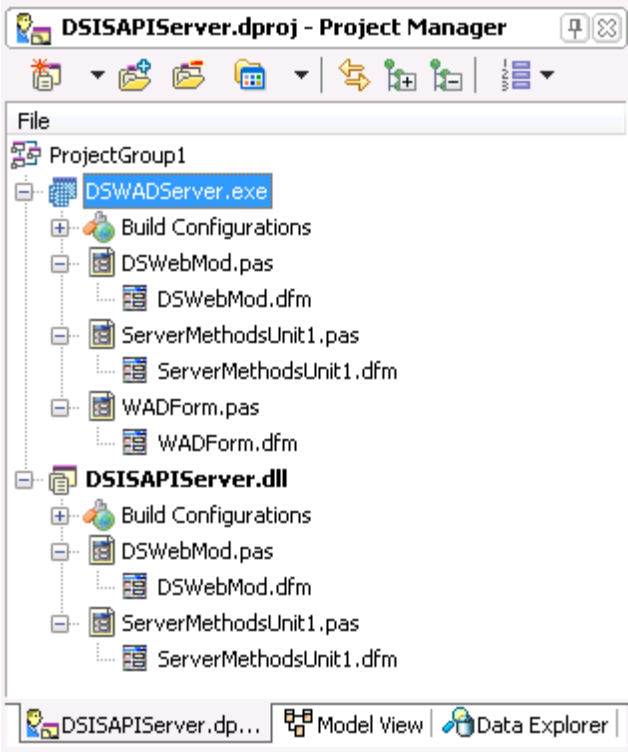
5.2. ISAPI 项目

在我们继续修改和自定义ServerMethodsUnit1.pas和DSWebMod.pas之前,我们首先添加一个新项目到项目组,这次是ISAPI/NSAPI Dynamic Link应用程序.右击项目组,选择Add New Project,弹出Object Repository.在DataSnap Server目录,使用New DataSnap WebBroker Application向导创建一个ISAPI/NSAPI Dynamic Link应用程序.这是不需要设置对话框下面的选项,因为我们将重用DSWADServer项目中已存在的单元.



点击OK按钮创建新项目(将添加到项目),叫做Project1,带有ServerMethodsUnit2.pas,Unit1.pas. 现在,去除新的单元ServermethodsUnit2和Unit1,我们将使用DSWADServer项目中的ServeMethodUnit1.pas和DSWebMod.pas单元.将ServerMethodsUnit2.pas和Unit.pas从项目中移除. 将DSWebMod.pas 和ServerMethodUnit2.pas添加到项目中.最后将Project1 命名为DSISAPIServer.dproj.

现在已经有了含两个项目(共用DSWebMod和DSServerMethodUnit1.pas单元)的项目组.



这时就可以使用DSWADServer项目进行调试,使用DSISAPIServer项目将DataSnap服务部署到IIS.

在向ServerMethodsUnit1单元添加方法前,我们需要修正自动生成的ISAPI/NSAPI项目文件代码来创建TDSServerModule实例. TDSServerModule实际上是一个数据模块,就像一个Web模块,在运行ISAPI DLL项目中将发生错误, 因为在Web Broker项目中只能有一个Web模块.

打开DSISAPIServer.dpr项目代码,修改主begin----end块:

begin

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

end.

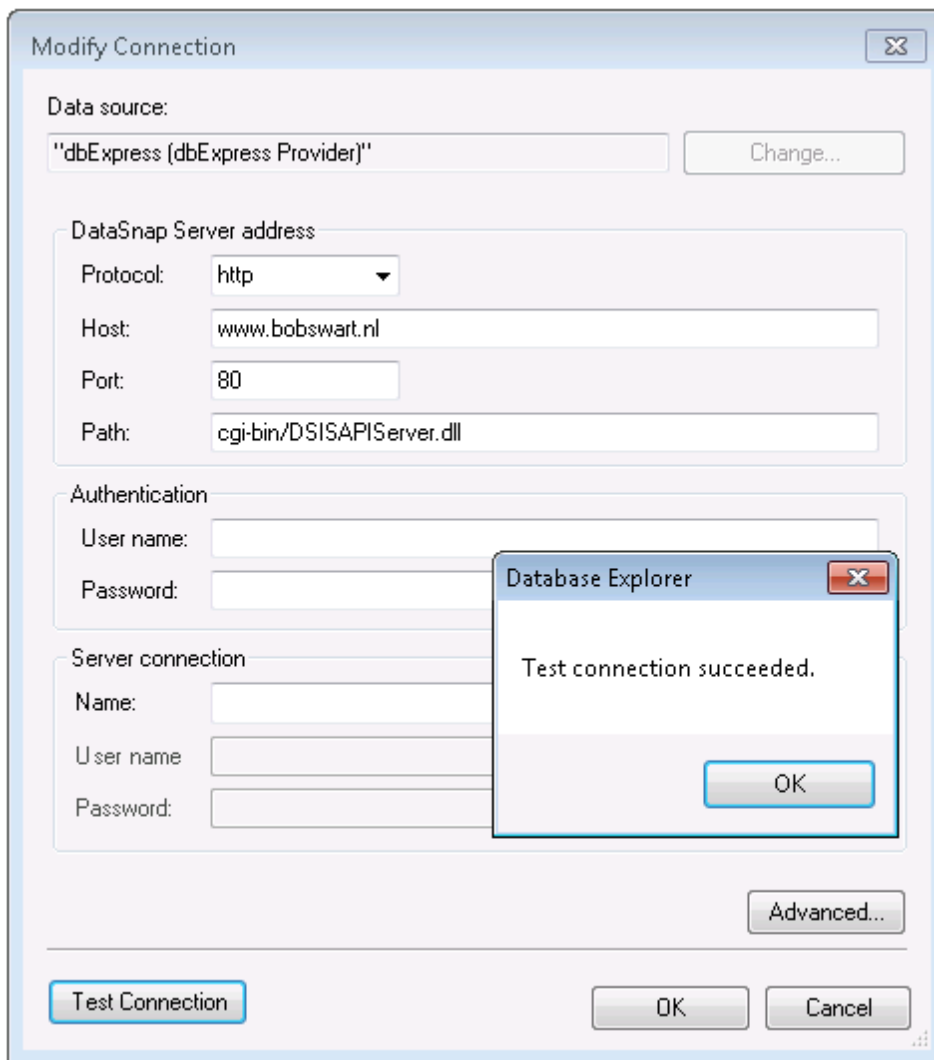
这将避免报只能有一个数据模块的错误.注意在实际部署ISAPI DLL是看不到这个错误信息.只报一个服务端错误或超时,所以记住这个冲突很重要.

5.3. 服务方法,部署,客户端

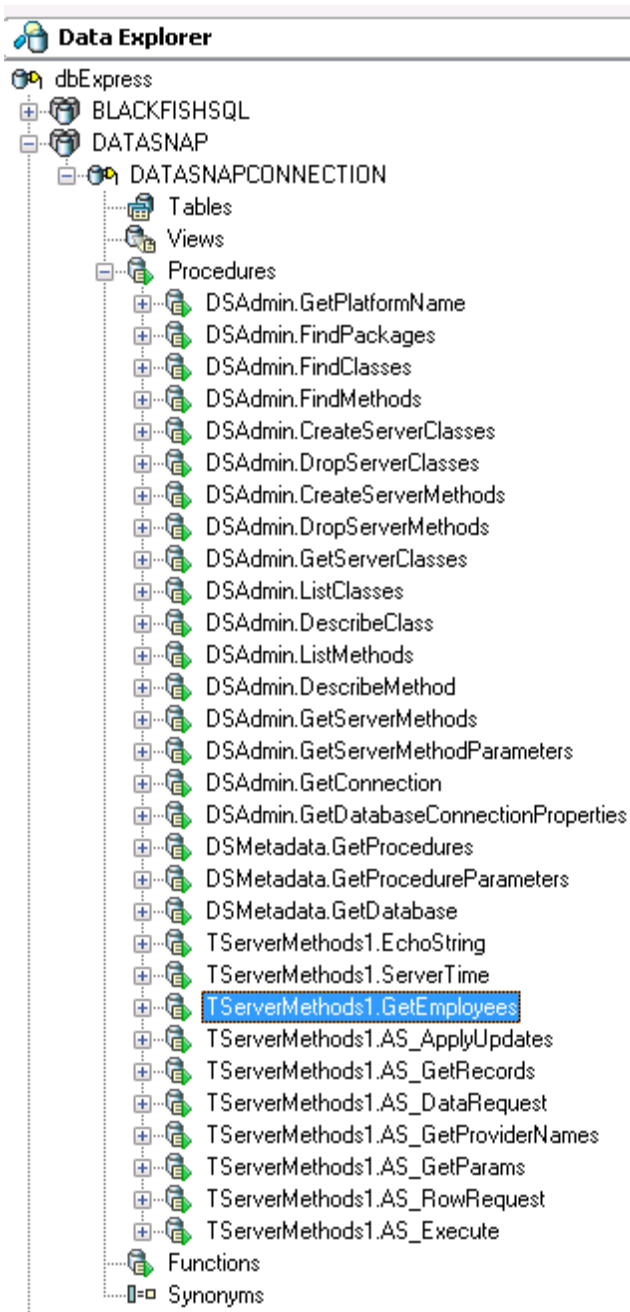
增加功能时只需要修改被两个项目共享的ServerMethodUnit1.pas单元.默认有一个范例函数,就是上面的范例,我们包含多个方法(组件说明和源码见2.1.4.).服务端方法实现后,我们就可以将ISAPI DLL部署到IIS.详细信息可见<http://blogs.embarcadero.com/jimtierney/2009/08/20/31502>.

本例中如你没有Web服务可用于部署,可以使用我以部署的DataSnap ISAPI.注意我没有发布TDataSetProvider,也没有实现返回数据的GetEmployees方法,但是ServerTime和EchoString方法都运行良好,可以用来测试DataSnap客户端了.

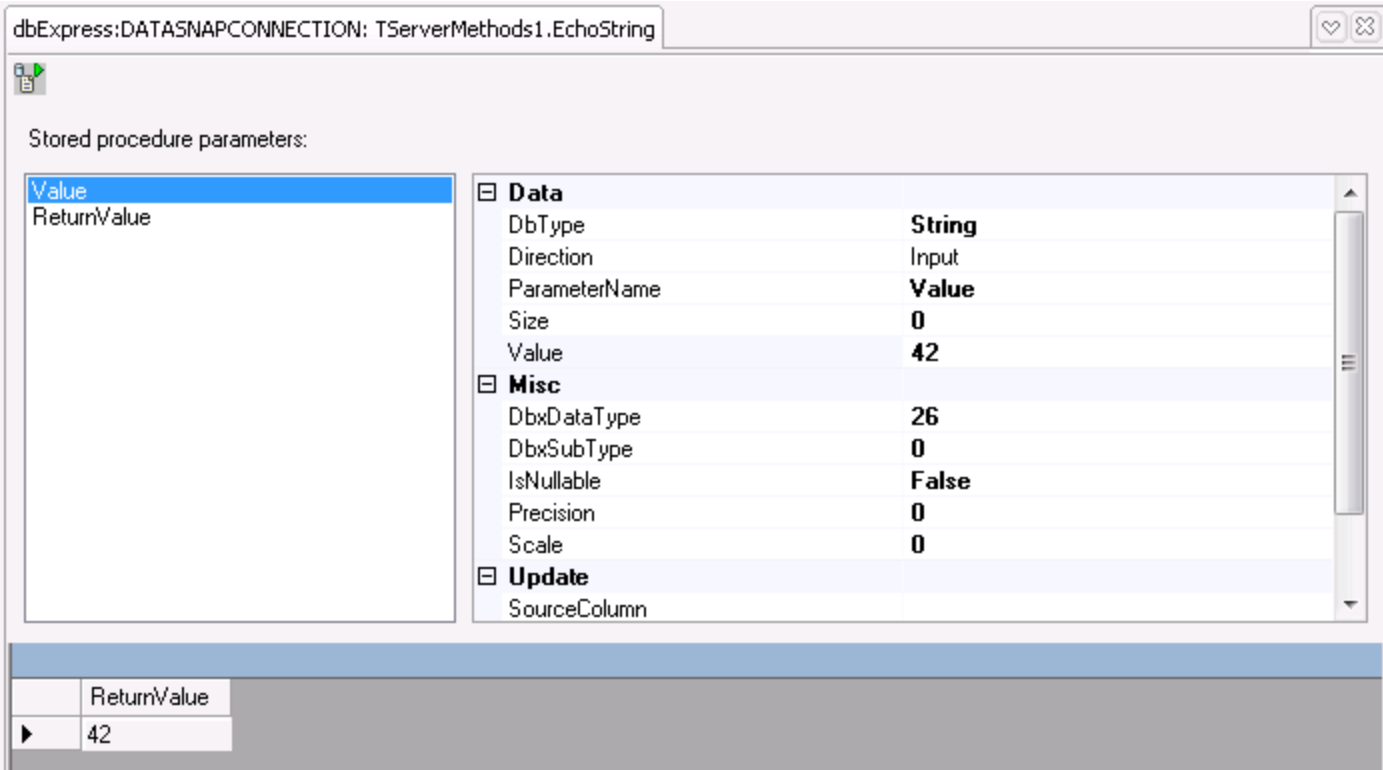
在客户端连接ISAPI DataSnap服务端前,可以使用Data Explorer检查一下是否可以连接到ISAPI DataSnap服务. Data Explorer上有一个新的叫做DATASNAP的目录,展开后,第一个连接叫做DataSnapConnection,右键修改连接.在这个对话框中我们可以选择协议,主机(如你没有自己的Web服务可以使用www.bobswart.nl),端口号,就ISAPI DataSnap服务应用程序在Web服务器上的URL路径,这里是cgi-bin/DSISAPIServer.dll.点击测试连接.



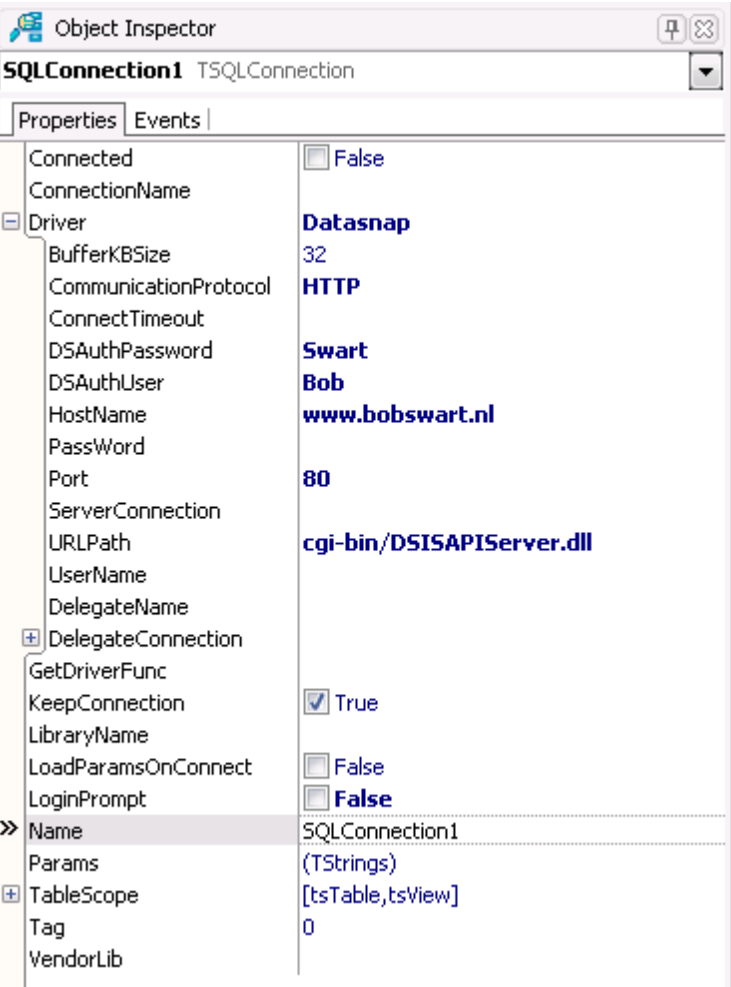
点击OK关闭窗口,在Data Explorer,展开DATASNAPCONNECTION节点查看表,视图,过程,函数和同义词.下图中,过程包括DSAdmin,DSMetaData,TServerMethods1.AS_XXX及我们自定义的三个函数EchoString,ServerTime 和 GetEmployees.



不需要写DataSnap客户端,现在就可以测试这些方法.例如EchoString方法(发送什么返回什么).右键点击TServerMethods1.GetEmployees方法,选择View Parameters,弹出一个新窗口,输入参数(例如42).在这个新窗口中右击,选择“执行远程服务方法”.运行结果将显示在ReturnValue中.



这样我们就可以调用远程DataSnap服务方法.为了在客户端连接到远程服务端,我们只需要修改 TSQLConnection组件的属性.原来我们连接都Windows版本的DataSnap服务,现在我们需要修改设置连接到 Web版本.



注意,如果你使用的是我发布的DSISAPIServer.dll,我已经禁用了TDataSetProvider,而且GetEmployees方法不

返回任何数据,但你可以使用ServerTime 和 EchoString方法.

6.任何使用REST和JSON

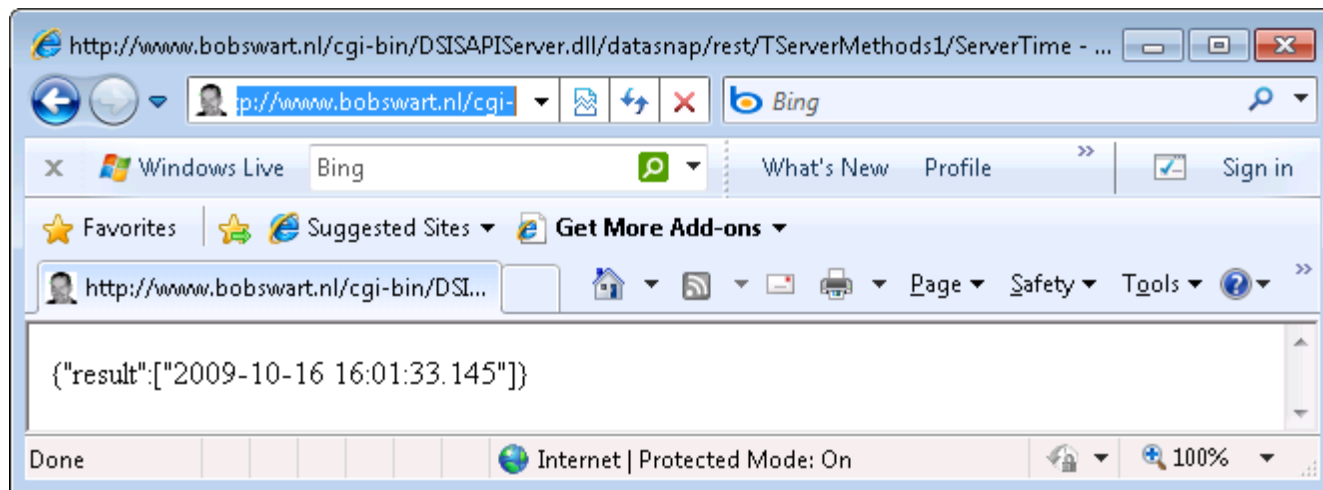
DataSnap2010支持REST和JSON.DataSnap2010特性REST支持DataSnap HTTP请求.例如,如果DataSnap服务的URI是<http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll>.我们可以在此URL后加 /datasnap/rest,后跟服务类名称,方法名称和参数.语法如下:

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

对于我的服务器上的TServerMethod1模块中的ServerTime方法,URL如下:

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

在浏览器中输入这个REST支持的URL,如下图:



在浏览器中返回结果是JSON结构:

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

更多信息见Marco Cantù的Delphi2010和REST客户端白页.

6.1. 回调

除了用REST支持调用DataSnap服务方法外,JSON还用于实现回调方法.DataSnap2010支持客户端回调函数,使其执行在服务方法上下文中.这样就可以实现客户端调用服务端方法时,服务端就可以调用由客户端传递好参数的回调函数.

例如,我们修改EchoString方法,向其中添加回调支持.修改后的EchoString方法如下:

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

TDBXcallback类定义在DBXJSON单元.在我们实现EchoString方法前,先搞清楚如果在客户端定义回调函数(毕竟,这是一个可以让服务端调用的客户端方法).

在客户端,我们必须定义一个新类,继承在TDBXCallback,重新起Execute方法.

type

```
TCallbackClient = class(TDBXCallback)
```

```
public
```

```
function Execute(const Arg: TJSONValue): TJSONValue; override;
```

```
end;
```

在Execute方法中,有一个TJSONValue类型的参数,可以复制(Clone)这个参数然后设置其具体内容.Execute方法也返回一个TJSONValue类型的值,这里我们只返回同样的值:

```
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;

例如, 在方法实际返回前(如方法正在执行),回调函数将显示EchoString方法传递参数的值.服务端新的EchoString方法实现需要将String值赋给一个TJSONObject对象,并将其传递给回调函数.如下:

```
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;

注意这个回调函数将在客户端被执行—然后在服务端Echostring方法执行完毕前并返回最后,在客户端调用EchoString方法也需要修改,因为我们现在要提供一个回调类TCallbackClient的实例,如下所示:

var

```
MyCallback: TCallbackClient;
```

begin

```
MyCallback := TCallbackClient.Create;  
  
try  
Server.EchoString(Edit1.text, MyCallback);
```

finally

```
MyCallback.Free;
```

end;

end;

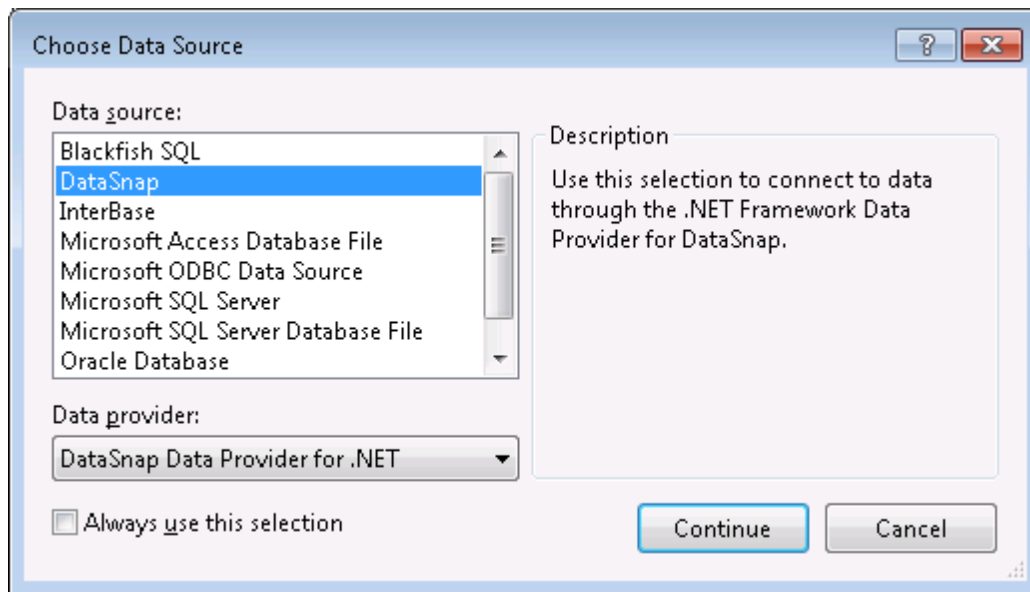
这个范例阐释了如何在DataSnap2010中使用客户端回调函数.

7. 使用DATASNAP 和 .NET

Delphi Prism 2010可用来构建使用我们先前生成Wind32服务的DataSnap .NET客户端.为了构建Delphi Prism 2010 DataSnap客户端,先确保DataSnap服务正常运行.

启动Delphi Prism 2010,点击View→ Server Explorer启动Delphi Prism Server Explorer.首先建立一个连接,验证我们将以使用的DataSnap服务.

Server Explorer的根节点叫做Data Connections.右击Data Connections选择添加连接.对话框如下,在Data Sources列表中选择DataSnap(注意如果数据源已经预选好了我们需要点击变更一下)



不要选中 **Always use this selection**g.除非你一直构建DataSnap数据连接.

点击**Continue**按钮进入下一步.指定连接的DataSnap服务详细信息.在协议下拉框中选择TCP/IP或HTTP.接下来,指定服务器(运行DataSnap服务的主机名称,如在本机测试可指定localhost),然后指定端口号.默认HTTP为80端口,TCP/IP为211端口.但从本白页中可知这两个端口都应该修改,并确保和你在ServerContainerUnitDemo单元中设置的端口号对应.下一个属性包含路径,这在你要连接到基于Web Broker的DataSnap服务上很重要.设置为[http://后面的部分](#).

最后,不要忘记验证用户和密码,本例DataSnap服务使用HTTP验证.

Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
DataSnap (DataSnap Provider) Change...

DataSnap Server address

Protocol: http
Host: localhost
Port: 8080
Path:

Authentication

User name: Bob
Password: ●●●●●●

Server connection

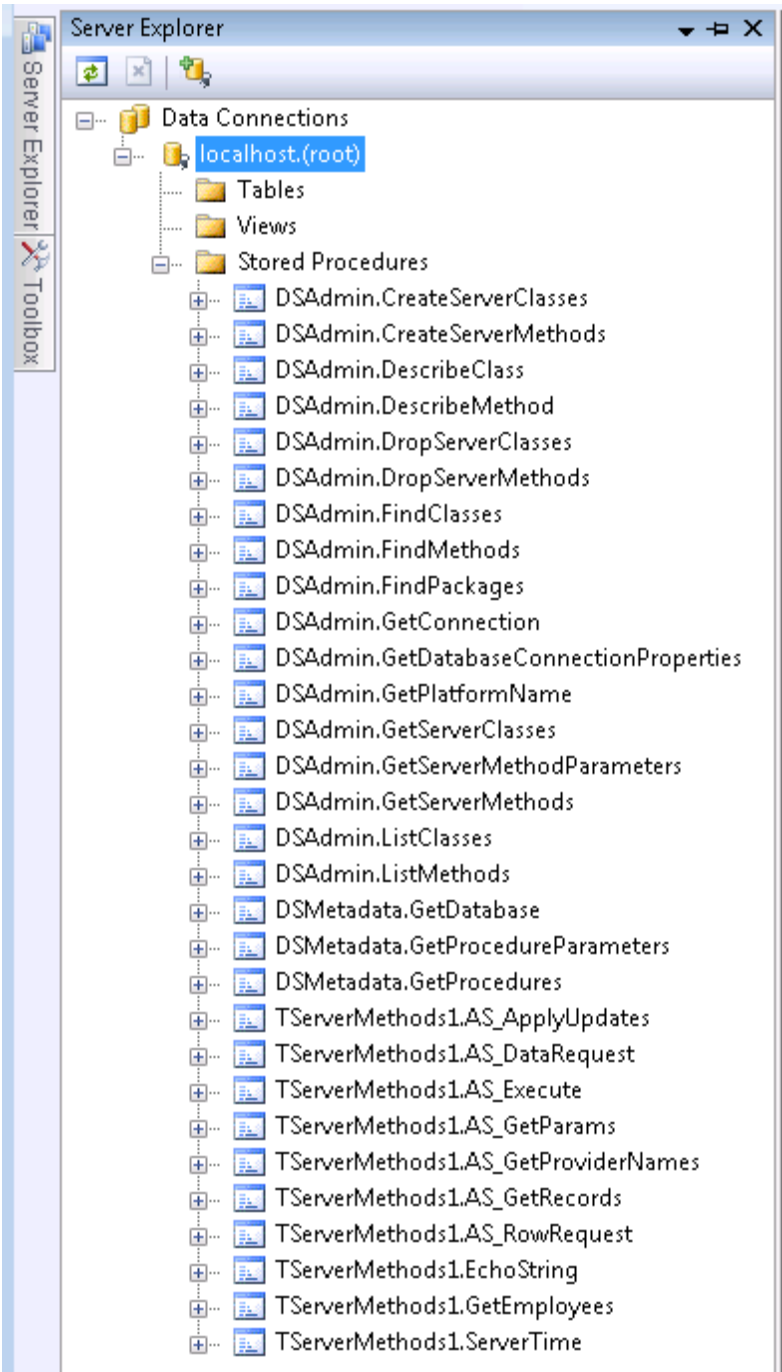
Name:
User name:
Password:

Advanced...

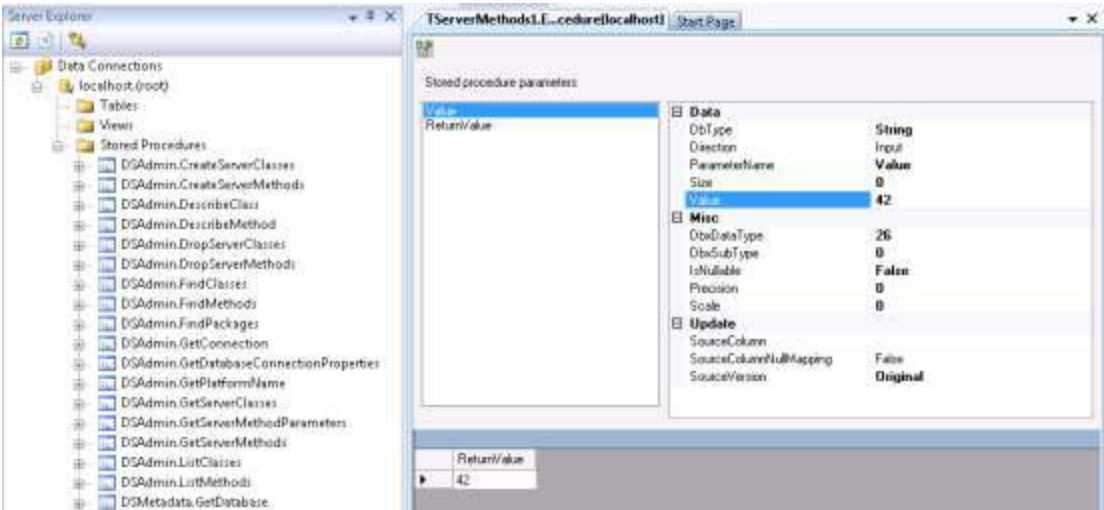
Test Connection OK Cancel

点击测试按钮,验证连接.如果弹出连接成功信息表示连接可用.

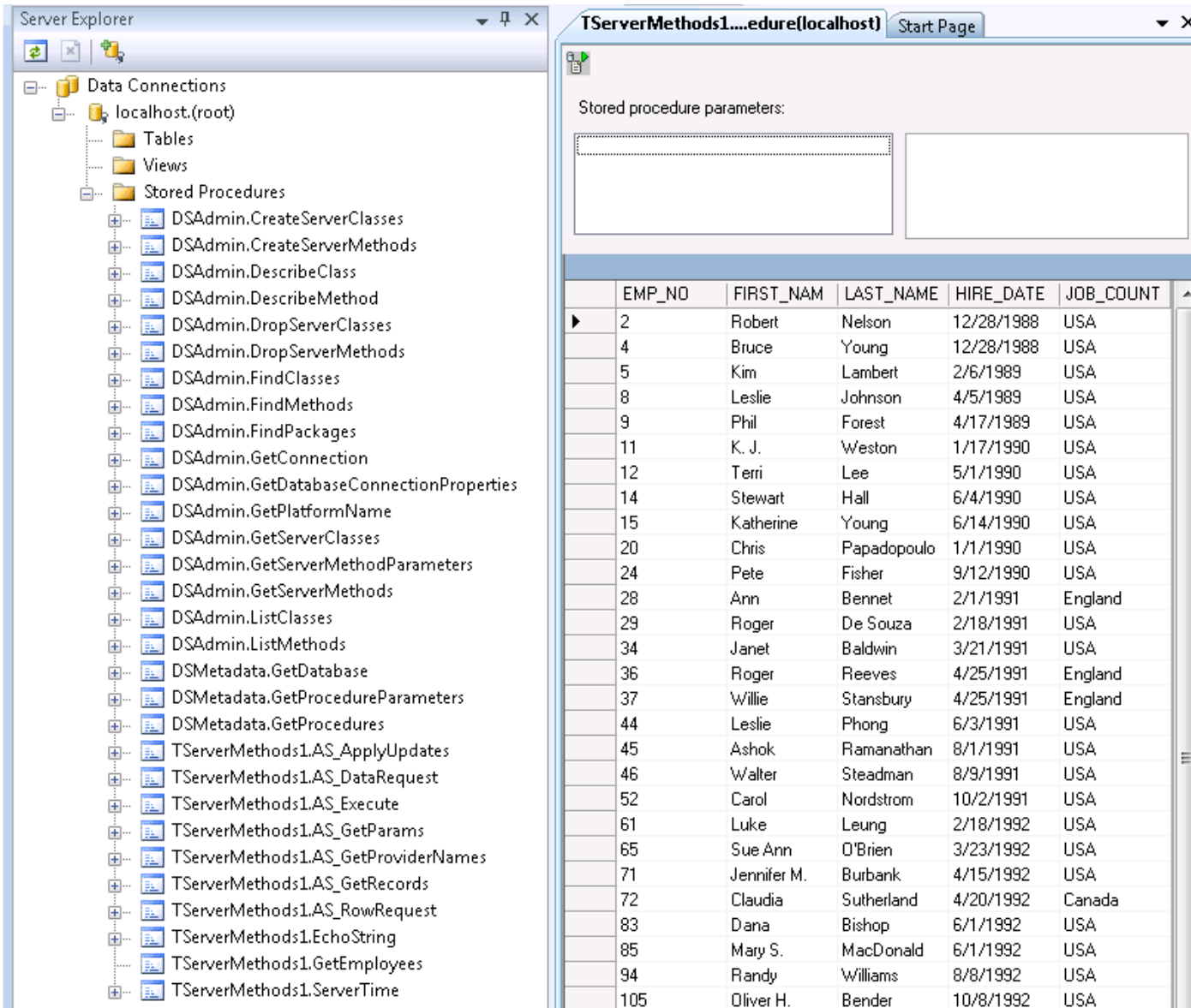
点击OK按钮,在连接树中显示了一个新的DataSnap连接.本例中是localhost节点.展开这个节点,显示表,视图,存储过程子节点.表和视图节点为空,但存储过程节点包括所有在DataSnap服务端定义的服务方法.包括我们自定义的EchoString,GetEmployees和ServerTime.



我们现在可以在Server Explorer中测试服务方法.例如,右击EchoString方法,选择查看参数.弹出新窗口,输入参数.这里输入42.右击窗口选择执行.将执行服务端的EchoString方法.如下图.



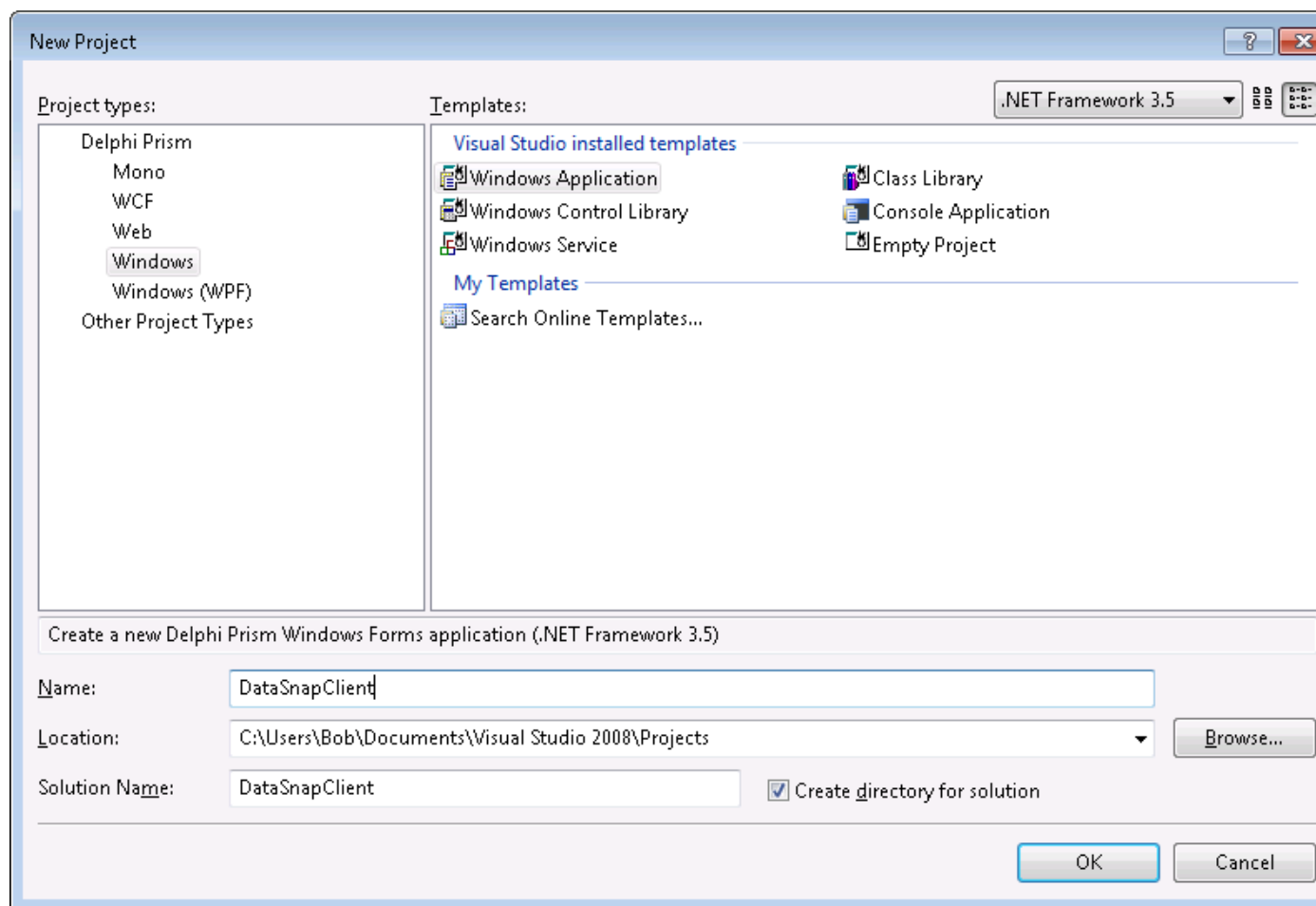
更好的的是可以使用GetEmployees方法演示如何从Employees表中获取数据.这个存储过程没有参数,但还有选择 View Parambers命令,返回一个空参数列表.右击窗体选择 执行.这是返回一个记录集.如下图:



7.1. WINFORMS 客户端

虽然已经可以运行服务端方法了,但更有用的方法是在.NET应用程序中调用这些方法.最后一个范例,File→New Project启动Delphi Prism新项目向导.选择项目类型.

在Windows 项目类型中选择Windows Application,修改WindowsApplication1为DataSnapClient.



点击OK按钮,创建一个带有Main.pas单元的新项目。

在Server Explorer,选择新建的DataSnap服务连接,属性框中找到ConnectionString,如下:

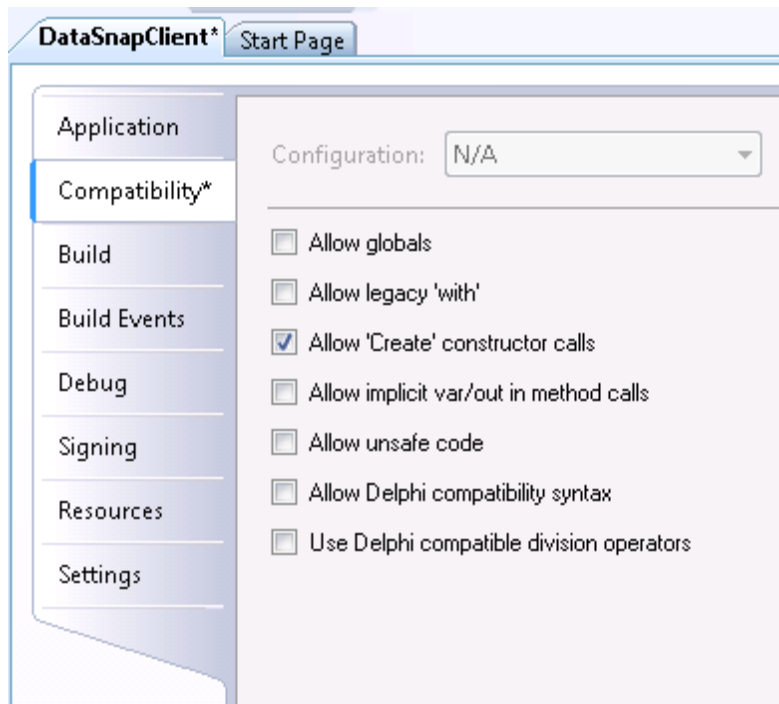
```
communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob;ds  
authenticationpassword=Swart
```

右击数据连接节点,选择生成客户端代理(Generate Client Proxy)选项.生成新文件ClientProxy1.pas,其中定义了TServerMethods1Client类及其中的方法(EchoString, ServerTime, 和GetEmployees)..如下:

```
TServerMethods1Client = class  
public  
    constructor (ADBXConnection: TAdoDbxConnection);  
    constructor (ADBXConnection: TAdoDbxConnection; AInstanceOwner: Boolean);  
    function EchoString(Value: string): string;  
    function ServerTime: DateTime;  
    function GetEmployees: System.Data.IDataReader;
```

除了代理类还在项目的引用节点中添加了Borland.Data.AdoDbxClient 和Borland.Data.DbxClientDriver引用。

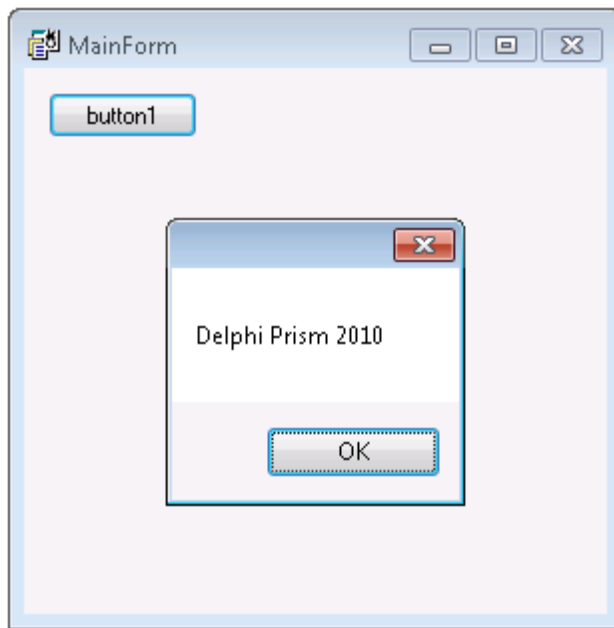
从TServerMethods1Client类代码片段中可见,类有两个构造函数:有使用了一个ADBXConnection参数,第二个构造函数还有一个AInstanceOwner的Boolean类型参数.这意味着我们必须使用参数调用构造函数.为了支持这个功能,必须修改项目属性设置.在解决方案管理器中右击DataSnapClient,选择属性.如下图,点击Compatibility标签,选中"Allow Create constructor calls",将允许我们调用.Create构造方法,传递参数,而不仅仅是使用new关键字。



现在回到主窗体,添加一个按钮.在Click事件中创建一个DataSnap服务连接并调用方法.

```
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=Bo
        b;dsauth
        enticationpassword=Swart';
    Connection.Open;
    try
        Client := ClientProxy1.TServerMethods1Client.Create(Connection);
        MessageBox.Show(
            Client.EchoString('Delphi Prism 2010'));
    finally
        Connection.Close;
    end;
end;
```

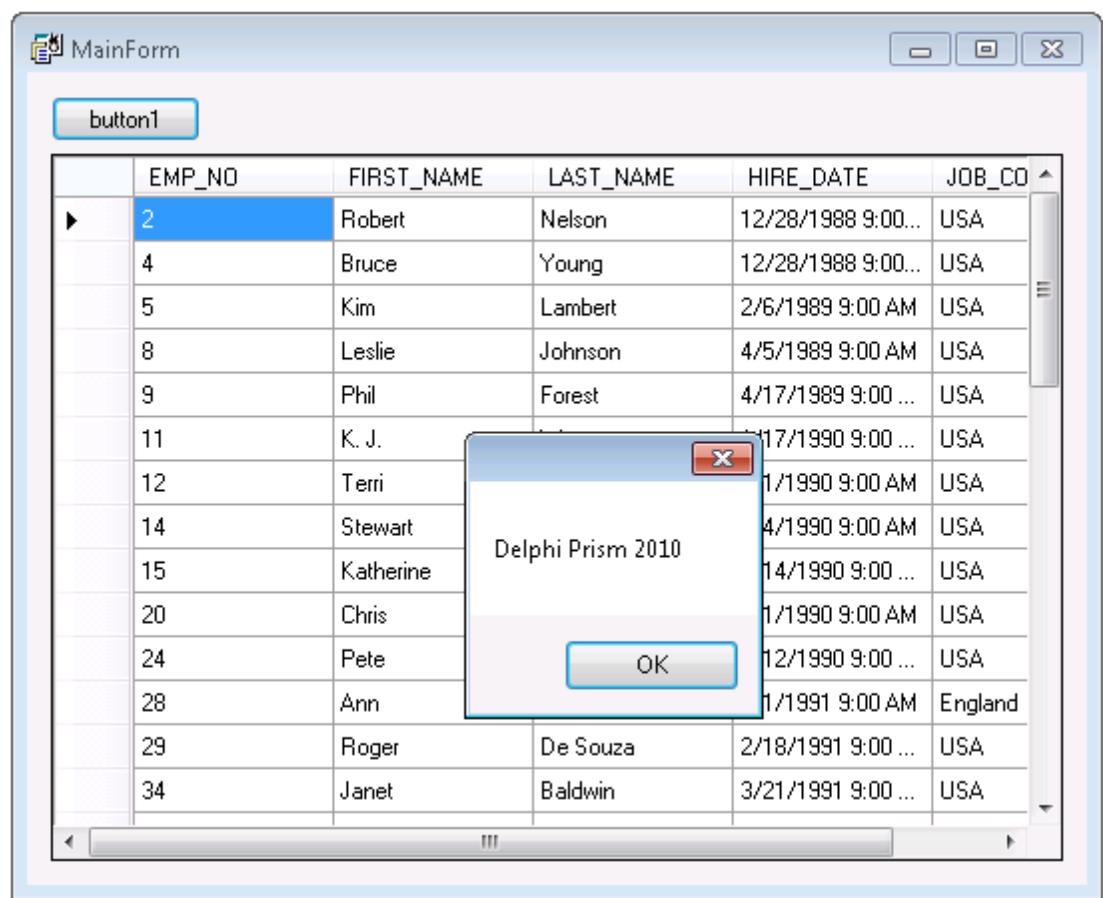
运行结果如下图所示:



同样方式,我们调用**GetEmployees**方法获取结果集并显示到**DataGridView**.这里有个小问题,**DataGridView**方法返回的是**IDataReader**(等价于**TSQLDataSet**结果集),而不是**DataSet**和**DataTable**.我们必须写几行代码将**GetEmployees**返回的结果集保存到**DataSet**的**DataTabl**中(等价于Win32中的**TClientDataSet**).

```
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=Bo
        b;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;
```


结果如下图所示



8. 总结

本白页讲述了如何使用DataSet创建各种类型项目(Windows:GUI,Service,Console.或Web:CGI,ISAPI, Web App Debugger),及Win32或.NET客户端.使用TCP/IP,HTTP及HTTP验证及使用一些过滤器进行压缩加密等.DataSet2010在DataSet2009基础上扩展,比基于COM的DataSnap和MIDAS更优秀.