

Single-tier IntraWeb and RemObjects DataAbstract server

Beigesteuert von Bernhard Fischer

Letzte Aktualisierung

Für diesen Inhalt steht leider keine Übersetzungen zur Verfügung. Originaltext wird angezeigt.

This article was first published in 2006

This is a simple example how to create a basic IntraWeb (IW) standalone server utilizing RemObjects DataAbstract (DA) in a single-tier application. It will show a login page in your web browser. After you could be authenticated, your username and login time is shown on a new page. Additionally you can view the table of users from the database.
{multithumb}

We will use Delphi 7, IntraWeb 5.0, a Firebird 1.5 database and Remobjects DataAbstract 4. Newer versions should also work, of course. There is a download link at the end of this article where you can find the complete source code and a SQL file to reconstruct this example. Basic knowledge of all used technologies is required and assumed. So if you never created an IntraWeb or DataAbstract application before, this is what you perhaps should do first in separated scenarios. Some very basic steps in this article (defining DA connections etc.) are not described in detail.

The screenshots in this article may not be the most brilliant ones - I tried to keep them small to minimize the download time for this page.

First of all, the Firebird database IWRODA, containing a single table USERS should be prepared with the sql script (to keep it simple, I put the database and the code project in a folder C:\temp\IW-RODA. Database user is SYSDBA, password is masterkey. If you want to use a different setup, please adjust the parameters in the SQL script and in the schemas' connection string).

The IW server in the sample zip file is set up to work only on interface 127.0.0.1 and port 8099.

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot lang="sql"}
SET SQL DIALECT 3;
SET NAMES NONE;
CREATE DATABASE 'C:\temp\IW_RODA\IWRODA.fdb'
USER 'SYSDBA' PASSWORD 'mastereky' PAGE_SIZE 16384 DEFAULT CHARACTER SET NONE;
*****/
/**          Tables          ***/
/*****/
CREATE TABLE USERS (
    USERNAME VARCHAR(16) NOT NULL,
    USERPASS VARCHAR(16) NOT NULL,
    FORENAME VARCHAR(16),
    LASTNAME VARCHAR(16)
);

INSERT INTO USERS (USERNAME, USERPASS, FORENAME, LASTNAME) VALUES ('user', 'pass', 'John', 'Doe');
INSERT INTO USERS (USERNAME, USERPASS, FORENAME, LASTNAME) VALUES ('user2', 'pass2', 'George',
'Bush');
INSERT INTO USERS (USERNAME, USERPASS, FORENAME, LASTNAME) VALUES ('user3', 'pass3', 'Who', 'Cares');

COMMIT WORK;

*****/
/**          Primary Keys ***/
/*****/

ALTER TABLE USERS ADD CONSTRAINT PK_USERS PRIMARY KEY (USERNAME);
```

{/geshibot}{/moshide} Then we will run the Delphi project wizard to create a IntraWeb standalone application with datamodule.

Let's give meaningful names to the auto-generated units/forms: the new application is saved as IWRODA.dpr, the datamodule is renamed to SessionDataModule and the application's main form now is called frmLogin, as it will provide login functionality before accessing any other services. The unit filenames are made identical to the new names in Delphi, but the first letter "u" is added. Now we save the server controller unit as uServerController.pas and rename it to

ServerController. From your previous IW applications you know that you must manually adapt some lines of code after renaming units and after changing the auto-generated type declarations (changes are marked in the source code).

In the project unit IWRODA.dpr:

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot: lang="delphi"}
program IWRODA;
{PUBDIST}
uses
  IWInitStandAlone,
  uServerController in 'uServerController.pas' {ServerController: TDataModule},
  uSessionDataModule in 'uSessionDataModule.pas' {SessionDataModule: TDataModule},
  uFrmLogin in 'uFrmLogin.pas' {frmLogin: TIWAppForm};

{$R *.res}

begin
  IWRun(TfrmLogin, TServerController);
end.

{/geshibot}{/moshide}
```

In the frmLogin unit:

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot: lang="delphi"}
unit frmLogin;
{PUBDIST}
interface

uses
  IWAppForm,
  IWApplication,
  IWTypes;

type
  TfrmLogin = class(TIWAppForm)
  end;

implementation

{$R *.dfm}

uses
  uServerController; // **renamed**

end.

{/geshibot}{/moshide}
```

In the ServerController's unit:

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot: language="delphi"}
unit uServerController;
{PUBDIST}
interface

uses
  Classes,
  uSessionDataModule, // **renamed**
  IWServerControllerBase,
  IWAppForm,
  IWApplication,
  SysUtils;

type
  [...]
  TUserSession = class(TComponent)
  public
```

```

SessionDataModule: TSessionDataModule; // **renamed**
Username: string; // **added**
LoginTime: TDateTime // **added**
constructor Create(AOwner: TComponent); override;
end;

[...]

implementation

[...]

{ TUserSession }

constructor TUserSession.Create(AOwner: TComponent);
begin
  inherited;
  SessionDataModule := TSessionDataModule.Create(AOwner); // **renamed**
end;

[...]

```

```
{/geshibot}{/moshide}
```

As you have noticed, we also added a Username and a DateTime field to the TUserSession object. This is where we will store some values after the user has logged in. The UserSession is the right place to store client related information in a IW application. in the SessionDataModule's unit:

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot: language="delphi"}
unit uSessionDataModule;
```

```
interface
[...]
```

```
implementation
```

```
uses
  IWInit,
  uServerController, // **renamed**
  uGlobalDataModule; /**renamed**
```

```
[...]
```

```
function SessionDataModule: TSessionDataModule;
begin
  Result := TUserSession(RWebApplication.Data).SessionDataModule; // **renamed**
end;
```

```
[...]
```

```
{/geshibot}{/moshide}
```

When you reach this point, please make sure that your renaming went well and is exactly as described above by compiling the application. You should not continue as long as you encounter errors. Please check that your changes are identical to the marked passages above.

As the IW application serves the clients' requests in threads, we will use a global datamodule for the common components we need in the server-wide context, and a client datamodule to create instances from for each user session. The client module is the auto-generated unit we adapted above (SessionDataModule). So now we need another datamodule. We call it GlobalDataModule and save it as uGlobalDataModule.pas. Here we must drop some of the RO/DA components. In our single-tier application we will use the RO LocalServer - it was designed for exactly this purpose. It allows access to the RODA services in the same application. So this component belongs in the global datamodule, accompanied by some other required components we drop here: From the DataAbstract component tab:

- TDAConnectionManager

- TDADriverManager
- TDAIBXDriver From the Remobjects SDK tab:
- TROBinMessage
- TROLocalServer

There are a few settings to be entered for some of the components. In the object inspector, click on the TROLocalServer's property Dispatchers to link the server component to the message component. Then click on Add and select the existing messageServer from the Message dropdown List:

Now you can close the dialog. Of course you need to link the connection manager with the driver manager via the connection manager's DriverManager property now, as usual.

We add a initialization and finalization section to create /destroy the global datamodule, and that's all we have to do in this unit.

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot: language="delphi"}
unit uGlobalDataModule;
interface

uses
  SysUtils,
  Classes,
  uROClient,
  uROBinMessage,
  uDAEngine,
  uDAIBXDriver,
  uDADriverManager,
  uDAClasses,
  uROServer,
  uROLocalServer;

type
  TGlobalDataModule = class(TDataModule)
    serverLocal: TROLocalServer;
    mgrConnections: TDAConnectionManager;
    mgrDrivers: TDADriverManager;
    driverIBX: TDAIBXDriver;
    messageServer: TROBinMessage;
  end;

var
  GlobalDataModule: TGlobalDataModule;

implementation

{$R *.dfm}

initialization /**addedd**
  GlobalDataModule := TGlobalDataModule.Create(nil); /**added**

finalization /**addedd**
  FreeAndNil(GlobalDataModule); /**added**

end.
```

{/geshibot}{/moshide}Now that we defined the global (in a 2-tier scenario: server-side) part of the RO communication layer, we complete the session (client-side) part: In the SessionDataModule, drop the following components:

From the Remobjects SDK tab:

- TROLocalChannel
- TROBinMessage
- TRORemoteService Set the TROLocalChannel's ServerChannel property to its counterpart on the global datamodule, GlobalDataModule.serverLocal, and link the TRORemoteService's properties Message and Channel to messageSession and channelLocal. Enter DataService in the RemoteService property.

Next we select **Turn IWRODA into a Remobjects SDK server** from the Delphi Remobjects menu. This will adapt the project file to generate required files after we defined our services. Now we are asked if we want to launch the ServiceBuilder to define a service, and that's exactly what we will do. Let's rename the service library to **IWRODALibrary** and add 4 services (**CalcService**, **DataService**, **LoginService** and **TimeService**). The next step is to include **DataAbstract RODL** because we want to define one of our services as **DataAbstract** service (we can use DA features for local data access in non-DA services too, as we will see later on).

The services **TimeService** and **CalcService** will implement methods we all know from the standard RO examples (**Sum** and **GetTime**), so you add the simple code yourself in the **CalcService** and **TimeService**. We will look at the **LoginService** and **DataService** in detail, as they deal with database access via DA. First the **LoginService** - simply add a method called **Login** with two input parameters (**iUsername: string**, **iPassword: string**) and a boolean result. Now the **DataService** - because we included the existing **DataAbstract RODL**, we can now use the **DataAbstractService** as ancestor for our **DataService**.

There is nothing more to do now, as everything is implemented in the included definition. So after closing the ServiceBuilder, we compile our application and select the Remobjects SDK Remote Datamodule three times when suggested. The 4th module will be automatically created as **DataAbstract Datamodule** because of its inheritance as defined in the ServiceBuilder.

After the 4 new units have been added to our project, we first complete the **DataService_Impl** unit. Drop a **TDABinDataStreamer** and a **TDASchema** on the module and give some meaningful names again (**schemaUsers** and **streamerUsers**, as we will access the users table in our database). After setting the service module's **ServiceSchema** and **ServiceDataStreamer** properties to the components we just added, now include the **uGlobalDataModule** unit in the uses clause and link the **schemaUser's ConnectionManager** property to **GlobalDataModule.mgrConnections**.

It's time to define our schema. Double click on the **schemaUsers** component to launch the **DA Schema Modeller**. Add a **IBX** connection to the database we initially created. Let's call it **IBX Connection**. For this example it's sufficient to simply drag & drop the table **USERS** from the **Data Explorer** pane to the **Data Tables** pane. After closing the **Schema Modeller**, enter **IBX Connection** in the **DataService's Connection** property. That's it for now in the **DataService** module.

Don't forget to complete the simple methods **GetTime** in **TimeService_Impl.pas** and **Sum** in **CalcService_Impl.pas** as usual, as we will use them later.

Now we will complete the **LoginService** implementation by dynamically creating a dataset & connection to verify a username and password from the web page. But first let's drop a **TDASchema** to define the table we want to use for verification. In our simple example this will be the **USERS** table again. Rename the new schema to **schemaLogin**, and after adding the **uGlobalDataModule** to the uses clause, hook it to the required connection manager **GlobalDataModule.mgrConnections** again. Now double-click the **schemaLogin** and pull the **USERS** table from the **Data Explorer** pane to the **Data Tables** pane. Here we will expand the **USERS** statement to receive username and password as parameters:

After you changed the SQL statement, don't forget to recreate the parameters in the **SchemaModeller!**

Now close the **Schema Modeller** and complete the code for the validation. This is what the completed **LoginService_Impl.pas** looks like (don't forget to include **uDAInterfaces.pas** in the uses clause):

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot language="delphi"}
unit LoginService_Impl;

{-----}
{ This unit was automatically generated by the RemObjects SDK after reading }
{ the RODL file associated with this project . }
{ }
{ }
```

```
{ This is where you are supposed to code the implementation of your objects.  }
{-----}
```

```
interface
```

```
uses
```

```
{vcl:}Classes,
SysUtils,
{RemObjects:}uROClientIntf,
uROTypes,
uROServer,
uROServerIntf,
uROSessions,
{Required:}uRORemoteDataModule,
{Used RODLs:}DataAbstract4_Intf,
{Generated:}IWRODALibrary_Intf,
uDAClasses;
```

```
type
```

```
{ TLoginService }
TLoginService = class(TRORemoteDataModule, ILoginService)
  schemaLogin: TDASchema;
protected
  { ILoginService methods }
  function Login(const iUsername: string; const iPassword: string): Boolean;
end;
```

```
implementation
```

```
{$R *.dfm}
```

```
uses
```

```
uDAInterfaces,
{Generated:}IWRODALibrary_Invk,
uGlobalDataModule;
```

```
procedure Create_LoginService(out anInstance: IUnknown);
```

```
begin
```

```
  anInstance := TLoginService.Create(nil);
```

```
end;
```

```
{ LoginService }
```

```
function TLoginService.Login(const iUsername: string; const iPassword: string): Boolean;
```

```
var
```

```
  dsLogin: IDADataset;
  con: IDAConnection;
```

```
begin
```

```
  Result := False;
```

```
  with schemaLogin.ConnectionManager do
```

```
    con := NewConnection(GetDefaultConnectionName);
```

```
  dsLogin := schemaLogin.NewDataset(con, 'USERS');
```

```
  if not con.InTransaction then
```

```
    con.BeginTransaction;
```

```
  try
```

```
    with dsLogin do
```

```
      begin
```

```
        ParamByName('username').AsString := iUsername;
```

```
        ParamByName('userpass').AsString := iPassword;
```

```
        Open;
```

```
        Result := not IsEmpty;
```

```
        Close;
```

```
      end;
```

```

if con.InTransaction then
  con.CommitTransaction;

except
  if Assigned(con) then
    if con.InTransaction then
      con.RollbackTransaction;
    end;
  end;

initialization
  TROClassFactory.Create('LoginService', Create_LoginService, TLoginService_Invoker);

end.

{/geshibot}{/moshide}

```

Now that we completed the background processing, we have to create the web forms to receive the parameters and show the results. First we drop some IW components on the Login form:

- TIWEdit (2)
- TIWLabel (2)
- TIWButton

We rename the components: labelName, labelPassword, editName, editPassword and buttonLogin .

Now create a new IW application form with the Delphi wizard. This will become our content form to be shown after the user logged in.

please correct the auto-generated uses clause immediately!

change

```

uses
  ServerController;    to

```

```

uses
  uServerController;

```

Again, drop some IW components:

- TIWLabel (2)
- TIWButton
- TIWDBGrid
- TIWLink

The new names: labelHello, labelTime, buttonShow, gridUsers and linkQuit .

Additionally, this form is the place where we need our DA "client-side" components from the DataAbstract tab:

- TDABinDataStreamer
 - TDARemoteDataAdapter
 - TDACDSDDataTable
 - TDADataSource
- We rename the components: dataStreamer, remoteAdapter, tableUsers and dsUsers .

Linking the components is business as usual:

```

gridUsers.DataSource:      dsUsers
dsUsers.DataTable:        tableUsers

```

```
tableUser.Logicalname:      USERS
tableUser.RemoteDataAdapter: remoteAdapter
remoteAdapter.DataStreamer: dataStreamer
```

Set gridUsers ' property Visible to False .
Now include the units uSessionDataModule , uServerController and SysUtils in the uses clause.

In our frmContent's OnCreate event, link the remoteAdapter's property RemoteService to the UserSession's SessionDataModule.serviceRemote . (see the following code box). The SessionDataModule is available at runtime after a client connected. Because we encapsulate our client-related stuff in a per-session manner, the application remains thread-safe.

This is the complete unit for our content form. Please implement all events to have a working example.

```
{moshide [click to expand source code]][click to collapse source code]}{geshibot language="delphi"}
unit uFrmContent;
```

```
{PUBDIST}
```

```
interface
```

```
uses
  IAppForm,
  IApplication,
  ITypes,
  Classes,
  Controls,
  IControl,
  ICompLabel,
  uDADataStreamer,
  uDABinAdapter,
  DB,
  uDADataTable,
  uDAScriptingProvider,
  uDACDSDDataTable,
  IGrids,
  IWDBGrids,
  ICompButton,
  uDARemoteDataAdapter,
  IWHTMLControls;
```

```
type
```

```
TfrmContent = class(TIAppForm)
  labelHello: TIWLabel;
  labelTime: TIWLabel;
  buttonShow: TIWButton;
  gridUsers: TIWDBGrid;
  linkQuit: TIWLink;
  dataStreamer: TDABinDataStreamer;
  remoteAdapter: TDARemoteDataAdapter;
  tableUsers: TDACDSDDataTable;
  dsUsers: TDADatasource;
  procedure IAppFormRender(Sender: TObject);
  procedure IAppFormCreate(Sender: TObject);
  procedure buttonShowClick(Sender: TObject);
  procedure linkQuitClick(Sender: TObject);
end;
```

```
implementation
```

```
{ $R *.dfm }
```

```
uses
  SysUtils,
  uServerController,
  uSessionDataModule,
  IForm;
```

```

procedure TfrmContent.IWAppFormRender(Sender: TObject);
begin
  labelHello.Caption := 'Hello ' + UserSession.Username;
  labelTime.Caption := 'Your login time was ' + DateTimeToStr(UserSession.LoginTime);
end;

procedure TfrmContent.IWAppFormCreate(Sender: TObject);
begin
  remoteAdapter.RemoteService := UserSession.SessionDataModule.serviceRemote;
end;

procedure TfrmContent.buttonShowClick(Sender: TObject);
begin
  tableUsers.Open;
  gridUsers.Visible := True;
end;

procedure TfrmContent.linkQuitClick(Sender: TObject);
begin
  WebApplication.Terminate('That's it!');
end;

end.

```

{/geshibot}{/moshide} Note that the two labels on the content form are filled in the OnRender event with values from the UserSession - values we filled with our RO services before.

What is left over, is the first form in our application - the Login form. Make sure to include our library interface file IWRODALibrary_Intf.pas in the uses clause, because we will access the LoginService from this login form.

We also need the uSessionDataModule unit included to connect to the services.

Insert the unit uFrmContent in the upper, interface uses clause, because we need it in our type declaration. Note the public field frmContent in the TfrmLogin declaration. Remember, we can't use global variables.

Our login unit's OnCreate event is also a good place to create the content form above - the user will see next after authentication.

Now fill the OnClick event for the Login button. There is a basic check for the username and password length before our LoginService is called. Last but not least we also fetch the login time from our TimeService to display it on the next web page and store it in the user's session. Because we are displaying information on web pages, we have to convert our services' results to strings. This could have been implemented in the services themselves. But to have a logical border, and to be able to use the services in a non-IW scenario, we do it in the web forms of our IW application.

```

{moshide [click to expand source code]][click to collapse source code]}{geshibot language="delphi"}
unit uFrmLogin;

```

```
{PUBDIST}
```

```
interface
```

```

uses
  IWAppForm,
  IWApplication,
  IWTypes,
  IWCompLabel,
  IWCompButton,
  Classes,
  Controls,
  IWControl,
  IWCompEdit,

```

```

uFrmContent;

type
TfrmLogin = class(TIWAppForm)
  editName: TIWEdit;
  editPassword: TIWEdit;
  buttonLogin: TIWButton;
  IWLabel1: TIWLabel;
  IWLabel2: TIWLabel;
  procedure buttonLoginClick(Sender: TObject);
  procedure IWAppFormCreate(Sender: TObject);
public
  FfrmContent: TfrmContent;
end;

implementation
{$R *.dfm}

uses
  uServerController,
  IWRODALibrary_Intf,
  uSessionDataModule;

procedure TfrmLogin.buttonLoginClick(Sender: TObject);
begin
  with UserSession do
  begin
    if ((Length(editName.Text) < 3) or (Length(editPassword.Text) < 3)) then
    begin
      WebApplication.ShowMessage('Username / password too short');
      exit;
    end;
    if CoLoginService.Create(SessionDataModule.messageSession,
      SessionDataModule.channelLocal).Login(editName.Text, editPassword.Text) then
    begin
      Username := editName.Text;
      LoginTime := CoTimeService.Create(SessionDataModule.messageSession,
        SessionDataModule.channelLocal).GetTime;
      FfrmContent.Show;
    end;
  end;
end;

procedure TfrmLogin.IWAppFormCreate(Sender: TObject);
begin
  FfrmContent := TfrmContent.Create(WebApplication);
end;

end.

{/geshibot}{/moshide}

```

We are ready now - let's compile and run the application. This is what the server will show you in your browser:

After you could be authenticated, your username and login time is shown on a new page.

Additionally, you can view the table of users from the database.

You can download the source code and SQL file of this example here.

This is a very basic example to demonstrate the usage of Remobjects SDK and DataAbstract in combination with IntraWeb in a single-tier application. As you could see, it fits together perfectly in a rather straightforward manner. Of course one can think of different scenarios - when moving an existing RODA server to an IW server, you can use the RO

DLLChannel, if minimum effort for the conversion and keeping both technologies a bit separated is the main aspect. You can implement more complex session scenarios when additionally using the RO session management in the service's business logic. Or you can create 2-or-more-tier applications where required, with a lot of comfort. Your imagination is the limit.

Web links:

Delphi: CodeGear

DataAbstract and the Remobjects SDK: Remobjects Software IntraWeb: Atozed Software

Firebird RDBMS: Firebird

{moscomment}