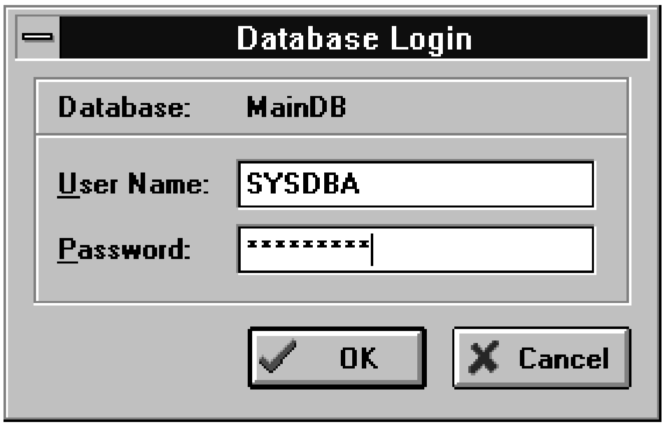
Customised Logins

Taking over Delphi's default login for database acess

by Xavier Pacheco

When designing database applications, it's often necessary to limit access to vital data. Access to this data can be obtained by using the proper user name and password. Delphi database applications inherit default security measures when using a database server that requires users to login to the server. Such measures include displaying a default login dialog box, see Figure 1, to obtain a valid user name and password in order to establish a connection to the database.

Under certain circumstances you might want to override Delphi's default and give your users access to the data without having to login every time they launch your application. Or, perhaps you want to provide your own custom login dialog box. You might also want to allow one user to logoff and log back in on the same machine without closing the application. This article will show how to perform these tasks.

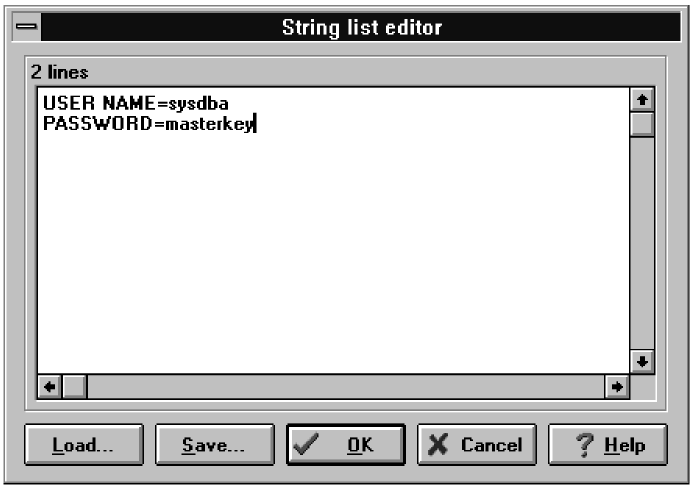


Database: MainDB

User Name: SYSDBA

Password: *********

➤ *Figure 1 Delphi's default login dialog*



```
USER NAME=sysdba
PASSWORD=masterkey
```

➤ *Figure 2 Initializing the Params property with a user name and password*

No Login Dialog

Typically, you connect data-aware components through a TDataSource component to a TTable component which is directly linked to a table in the database. The TTable component connects to the database server by its Alias property. You define aliases by using the BDECFG.EXE utility that ships with Delphi.

Note: When I refer to the TTable component, I am also referring to the TQuery and TStoredProc components since all are TDataset descendants.

For database servers which enforce security, a login dialog box will automatically be displayed to prompt the user for a valid user name and password. To prevent the login dialog box from displaying, you must use the TDataBase component to establish the initial connection to the database server. The TDatabase component provides additional functionality that gives you greater control over your database connection in addition to allowing you to customize server logins. In this article, I won't go into all the features of TDatabase, instead, I'll focus specifically on the login process.

When using a TDatabase component for preventing the login dialog from displaying, the following TDatabase properties must be set accordingly:

➢ AliasName Set to an existing BDE alias which has already been defined with the BDECFG.EXE utility. This is the same value typically used as the Alias property for TTable and TQuery components when a TDatabase component is not used.

➢ DatabaseName Set to an application specific alias, known only by the application using the TDatabase component. TTable, TQuery and TStoredProc components will use this value for their Alias property.

➢ Login Prompt Set to false in order to force the application to look in the TDatabase's Params property for the user name and password.

➢ Params Initialize to contain the user name and password. This is done by invoking the String List Editor for this property and specifying the strings as shown in Figure 2.

Note: Make sure that there are no spaces in the strings that specify the user name and password, otherwise the connection will fail. Once you've established the

TDatabase property settings, set your TTable.Alias property to reflect the value of the TDataBase.DatabaseName property. This value will appear in the drop down list for the TTable's Alias property in the Object Inspector.

Now you can connect to the server by setting the TDatabase component's Connected property to true at design-time in the Object Inspector, or at run-time with:

```
Database1.Connected := true;
```

This will prevent users from having to type in a user name and password every time they run your application.

Custom Login Dialogs

Instead of using the default login dialog provided by Delphi, you can design and use your own login for your database applications. When doing so, you must pass the values obtained from your login dialog box to the user name and password values of the TDatabase component. This is done in the TDatabase component's OnLogin event handler, which is called whenever a TDatabase component is connected and when its LoginPrompt property is set to true. A typical OnLogin event handler that accomplishes this is shown in Listing 1.

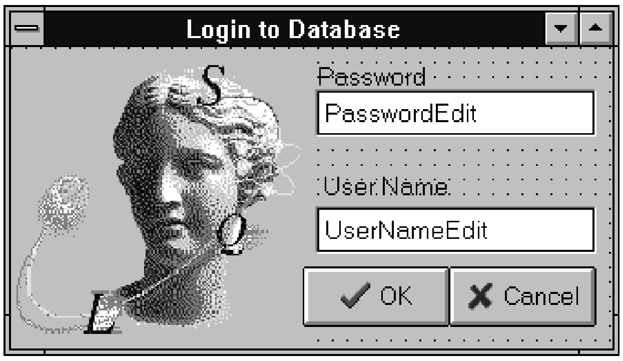
The LoginDlg is first created and the values from it are used to set the user name and password values for the TDatabase component by setting the appropriate items in the LoginParams parameter. LoginDlg can be any form from which the user name and password can be obtained. Therefore, your login dialog can have a much different look, like that shown in Figure 3.

Allowing Users To Logoff And Login

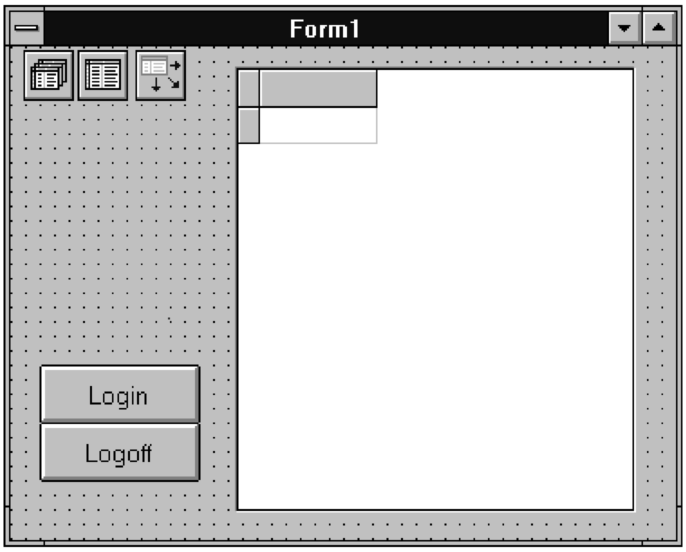
You can give your users the capability of logging off and logging back in, perhaps as a different user, in the same application without closing down the application. To do this, you must provide a method for logging off and logging back in. The example I've created for this article accomplishes this by providing login and logoff buttons on the main form, shown in Figure 4.

DataBase1's Alias property is set to IBLOCAL. This alias was preset when you installed Delphi and points to Interbase demo tables. DataBase1's DatabaseName property is set to MainDB. Table1's Alias property is therefore set to MainDB so that its connection goes through DataBase1. I've selected the COUNTRY table as the table to view. The TDatasource and TDBGrid components are used to display the data once the connection is made and the user has successfully logged in. Initially, Database1's Connected property is set to false as is Table1's Active property. Listing 2 shows the complete source code for the main form.

TForm1 contains three public variables UserName, Password and LoginSuccess. The Button1Click method is the login button's OnClick event handler. Here, I invoke the LoginForm and initialize UserName and Password with the values from LoginForm. Then I set Database1's Connected property to false and back to true again. This



➤ *Figure 3 A Custom Login Dialog*



➤ *Figure 4 Main form for the login/logoff example project*

```
procedure TForm1.Database1Login(Database: TDatabase; LoginParams: TStrings);
var
  LoginDlg: TLoginForm;
begin
  LoginDlg := TLoginForm.Create(Application);
  try
    if LoginDlg.ShowModal = mrOk then begin
    { Get the user name and password from the form's variables }
      LoginParams.Values['USER NAME'] := LoginDlg.UserNameEdit.Text;
      LoginParams.Values['PASSWORD'] := LoginDlg.PassWordEdit.Text;
    end;
  finally
    LoginDlg.Free;
  end;
end;
```

➤ *Listing 1 OnLogin event handler for TDatabase component*

```
unit Login0;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics, Controls, Forms, Dialogs, Grids, DBGrids,
  DB, DBTables, StdCtrls;
type
  TForm1 = class(TForm)
    Database1: TDatabase;
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Button2: TButton;
    procedure Database1Login(Database: TDatabase;
LoginParams: TStrings);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private   { Private declarations }
  public    { Public declarations }
    UserName,
    PassWord: String;
    LoginSuccess: Boolean;
  end;
var  Form1: TForm1;

implementation
uses Login1; { Make sure to add this to the uses clause }
{$R *.DFM}

procedure TForm1.Database1Login(Database: TDatabase;
  LoginParams: TStrings);
begin
  { Get user name and password from form's variables }
  LoginParams.Values['USER NAME'] := UserName;
  LoginParams.Values['PASSWORD'] := PassWord;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if LoginForm.ShowModal = mrOk then begin
    { Set the form's UserName and Password variables to
      that specified by the LoginForm edit controls }
    UserName := LoginForm.UserNameEdit.Text;
    PassWord := LoginForm.PasswordEdit.Text;
    try
      try
        { Disconnect the database }
        DataBase1.Connected := false;
        { Re-connect the database }
        DataBase1.Connected := true;
        { Set the table to active }
        Table1.Active := true;
        LoginSuccess := true;
      except
        on E:EDBEngineError do begin
          MessageDlg(E.Message, mtError, [mbok], 0);
          LoginSuccess := false;
        end;
      end;
    finally
      { Here you might handle application level security
        such as hiding or displaying controls based on
        the value of LoginSuccess }
      if LoginSuccess then
        ShowMessage('YES!')
      else
        ShowMessage('Nope');
    end;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DataBase1.Connected := false;
  UserName := '';
  Password := '';
end;
end.
```

➤ *Listing 2 LOGIN0.PAS – The main form's unit*

causes the Database1Login method to execute before the connection is actually made. Database1Login is Database1's OnLogin event handler. Since UserName and Password have already been set, their values are used to set the LoginParams values in the Database1Login method.

Note: The LoginForm is defined in the unit LOGIN1.PAS included with the source on the accompanying disk. It is simply the form shown in Figure 3.

If the login was successful (the user typed in a value user name and password) LoginSuccess is set to true, otherwise the except block is executed, which displays a message and sets LoginSuccess to false. Since the finally block is always executed, whether the login attempt is successful or not, you can use LoginSuccess to determine what, if any, application level security measures must be taken. Such a measure may be displaying or hiding various controls on the main form. In this example, I only display a message.

The Logoff button's OnClick event handler Button2Click simply sets Database1's Connected property to false and sets UserName and Password to empty strings. This effectively logs the current user out.

You are not restricted to Delphi's default behaviour for logging on to database servers. By using and building on the techniques I've shown here, you can design the security requirements for your Delphi application around the security requirements of both the users of your application and the tasks that your application aims to solve.

Xavier Pacheco is a Delphi Developer with TurboPower Software and co-author of "Delphi Developer's Guide" by Sams Publishing. You can reach Xavier on CompuServe at 76711,666