

Moving from the BDE to ADO

Migrating data from Paradox/dBASE to SQL Server

by Bob Swart

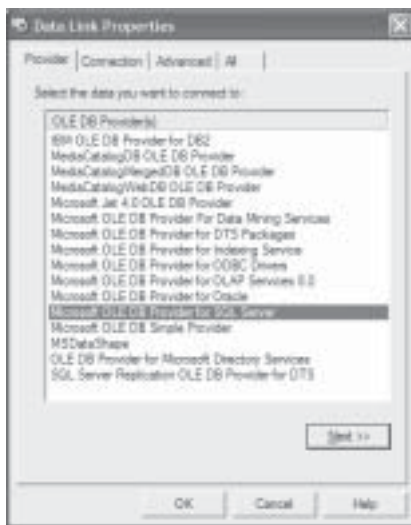
In this brief article, I describe a little application that I wrote some time ago to move data in Paradox and dBASE tables to a SQL Server database. The application is a small Win32 application, and you can use Delphi 7 or Delphi 2005 to play along (I'm not sure about Delphi 6, and Delphi 8 for .NET didn't include dbGO for ADO, so that one is certainly not usable).

dbGO for ADO

I assume you're familiar with the dbGo for ADO components (previously introduced as ADO Express in Delphi 5). If not, here's a very brief summary.

Using dbGO for ADO you can connect to a database using ADO (that much should be obvious, I reckon). The two most important dbGO components that we'll use are TADOConnection and TADODataset. The TADOConnection is used for the connection to the ADO database (which can be MS Access, SQL Server or any other ADO-connectable database), while the TADODataset can be used to retrieve data and work with data from the TADOConnection. The nicest thing about the TADODataset is the fact that it can mimic a Table, Query or Stored Procedure, all controlled by the value of its CommandType property (which in turn defines how to interpret the value of the CommandText property). There are other dbGO for ADO components, but these are less important, especially at this time.

dbGO for ADO and SQL Server



In order to work with SQL Server, we need to use a ADOConnection component and build the ConnectionString. We no longer want to use a Data Link file, but have to click on the Build button in the ConnectionString Editor. The dialog that follows starts with a page where we can select the OLE DB provider. For SQL Server, that should be the Microsoft OLE DB Provider for SQL Server, obviously (see screenshot on the left).

Once we've selected the OLE DB Provider, we can go to the next page and specify the Connection properties. For SQL Server, we



must start with the server name (which can be a . or the name of the server). Then we can either specify a username and password, or use Windows integrated security. As the third step, we have to select the database name on the server. Note that a drop-down combobox will show the available databases.

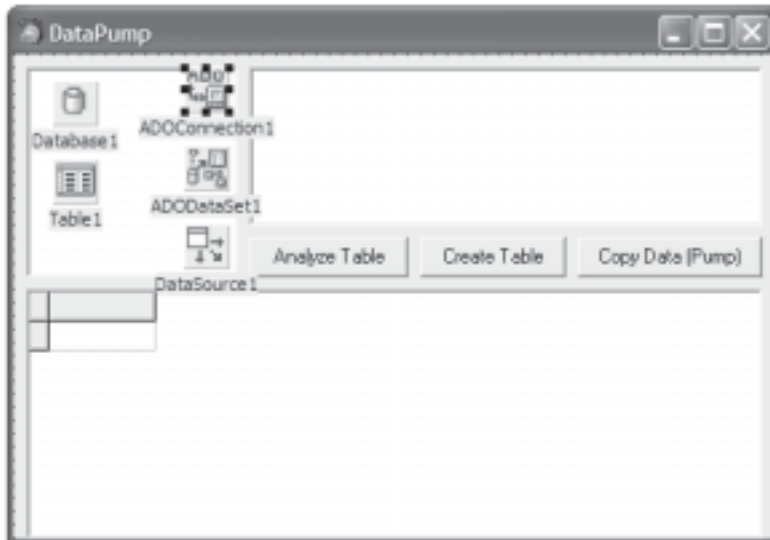
Finally, we can test the connection before closing this dialog. The resulting Connection String for the above example is Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=employee;Data Source=.'. This value will be stored in the DFM file with your Delphi form or data module.

Note that the database name "employee" probably doesn't exist on your system. I created a new SQL Server database using the CREATE DATABASE statement. Alternately you can use the Enterprise Manager to create a new database, or you can use the tempdb name for the temporary database (which can be compared to a scrapbook).

Data Migration

I have written a little application that will migrate BDE tables to SQL Server using dbGO for ADO: At design time, the application consists of a TListBox in the upper-left corner, a TMemo in the upper-right corner, and a TDBGrid at the bottom of the screen. The Listbox is used to display the available BDE database tables for a given alias. That's what the TDatabase and TTable components on the left are for: providing the input tables.

The TADOConnection and TADODataSet components next to them are connected to the target SQL Server database.



When the application starts, the listbox is filled with all the available tablenamees from the database. This is done with the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Database1.Connected := True;
  try
    Database1.GetTableNames(ListBox1.Items, False);
  finally
    Database1.Connected := False
  end
end;
end;
```

The Analyze Table button will open the selected table in the listbox, and walk through the field definitions. Using the selected tablename as well as all fieldnames (and their types), a SQL CREATE TABLE statement is generated and displayed in the memo field. Note that it's not executed yet.

The source code for the Analyze Table button is as follows:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  Str: String;
begin
  if ListBox1.ItemIndex >= 0 then
  begin
    Table1.TableName := ListBox1.Items[ListBox1.ItemIndex];
    Table1.FieldDefs.Update;
    Memo1.Lines.Clear;
    Str := 'CREATE TABLE ' + Table1.TableName + '(';
    for i:=0 to Pred(Table1.FieldDefs.Count) do
    begin
```

```

case Table1.FieldDefs[i].DataType of
    ftString: Str := Str + Table1.FieldDefs[i].Name
                + ' char(' + IntToStr(Table1.FieldDefs[i].Size) + ')';
    ftSmallint,
    ftInteger: Str := Str + Table1.FieldDefs[i].Name + ' int';
    ftFloat: Str := Str + Table1.FieldDefs[i].Name + ' float';
    ftDateTime: Str := Str + Table1.FieldDefs[i].Name + ' datetime';
    ftMemo: Str := Str + Table1.FieldDefs[i].Name +
                ' varchar(' + IntToStr(Table1.FieldDefs[i].Size) + ')';
    else
        if Str[Length(Str)-1] = ',' then Delete(Str,Length(Str)-1,2)
    end;
    if i = 0 then Str := Str + ' NOT NULL';
    if i < Pred(Table1.FieldDefs.Count) then Str := Str + ', ';
end;
Str := Str + ')';
Memol.Lines.Add(Str);
ADODataset1.Active := False;
ADODataset1.CommandType := cmdTable;
ADODataset1.CommandText := Table1.TableName;
ADODataset1.Active := True
end;
end;

```

The Create Table button will execute the SQL CREATE TABLE statement shown in the memo field (note that you can change the SQL statement that was generated by the Analyze Table code). Before a table is created, a DROP TABLE is executed, so you won't get an error regarding duplicate tables.

The source code for the Create Table button is as follows:

```

procedure TForm1.Button2Click(Sender: TObject);
begin
    ADODataset1.Active := False;
    ADOConnection1.Execute('DROP TABLE ' + Table1.TableName);
    ADOConnection1.Execute(Memol.Text);
    ADODataset1.CommandType := cmdTable;
    ADODataset1.CommandText := Table1.TableName;
    ADODataset1.Active := True
end;

```

Finally, we should also be moving the actual data from the BDE / SQL Links table to the dbGO for ADO (SQL Server) table, using the following code:

```

procedure TForm1.Button3Click(Sender: TObject);
var
    i: Integer;
    Str: String;
begin
    Memol.Lines.Clear;
    if ListBox1.ItemIndex >= 0 then
        try
            Table1.TableName := ListBox1.Items[ListBox1.ItemIndex];
            Table1.Open;
            ADODataset1.CommandType := cmdTable;
            ADODataset1.CommandText := Table1.TableName;
            ADODataset1.Open;
            while not Table1.Eof do
                begin
                    Memol.Lines.Text := Memol.Lines.Text + '#';
                    ADODataset1.Append;
                    for i:=0 to Pred(Table1.FieldDefs.Count) do
                        begin
                            case Table1.FieldDefs[i].DataType of
                                ftMemo,
                                ftString: ADODataset1.FieldByName(Table1.FieldDefs[i].Name).AsString :=
                                    Table1.FieldByName(Table1.FieldDefs[i].Name).AsString;
                                ftSmallint,
                                ftInteger: ADODataset1.FieldByName(Table1.FieldDefs[i].Name).AsInteger :=
                                    Table1.FieldByName(Table1.FieldDefs[i].Name).AsInteger;

```

```

        ftFloat: ADODataSet1.FieldByName(Table1.FieldDefs[i].Name).AsFloat :=
            Table1.FieldByName(Table1.FieldDefs[i].Name).AsFloat;
        ftDateTime: ADODataSet1.FieldByName(Table1.FieldDefs[i].Name).AsDateTime :=
            Table1.FieldByName(Table1.FieldDefs[i].Name).AsDateTime;
    else // skip
    end
end;
ADODataSet1.Post;
Table1.Next;
end
finally
    Table1.Close;
    ADODataSet1.First
end
end;

```

Note that the types that are migrated only include the Memo, String, SmallInt and Integer, Float and DateTime types. Other types are skipped at this time (feel free to extend this code yourself).

In my own work, this application has been used to convert several simple Paradox and dBASE tables to SQL Server, allowing me to take my data with me to a more capable database. The next step, of course, involves migrating the BDE data access components to ADO or ADO.NET data access components, but that's a story for another day.



Bob Swart (aka Dr Bob - www.drbob42.net) is a software developer, author, trainer, consultant and webmaster for his own one-man company, Bob Swart Training & Consultancy in Helmond, The Netherlands. He writes for numerous computing magazines as well as his own training material, and is also webmaster to the group.

Tips from the Technical Support Team

Using output parameters with TADODataset

If you are using output parameters with TADODataset objects, you've probably opened the Parameters collection, located the parameter in the object inspector and set its Direction property to pdOutput (or pdInputOutput). Unfortunately, if you then tamper with the CommandText property, the contents of Parameters will be dynamically rebuilt - losing your Direction settings. It's best to programmatically "remind" the object which parameters are meant to be output parameters, by calling

```
<TADODataset object>.Parameters.ParamByName('<param name>').Direction := pdOutput;
```

otherwise you'll get nothing but NULL values when the application runs.