ActiveX Data Objects

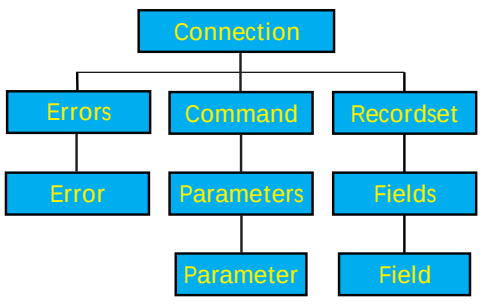
by Guy Smith-Ferrier

ActiveX Data Objects (ADO) is a set of ActiveX classes from Microsoft for accessing data. As such, ADO is a direct replacement for the VCL data retrieval classes and the BDE. In this article we'll look at how to get started with ADO and how it compares with the BDE.

Since the 1980s vendors have been offering solutions using a single API to access multiple data formats and engines. Microsoft's first try was ODBC. Borland were adamant that ODBC wasn't the answer and IDAPI provided more seamless access to both navigational and set-oriented database engines. Both solutions used the technology of the day, which was function-based APIs in a DLL.

The world then moved upwards and onwards into the world of object oriented programming and Borland placed class wrappers around IDAPI in the form of `TSession`, `TDatabase`, `TTable`, `TQuery` and `TStoredProc` classes.

With the whole world going to class-based solutions, and Microsoft digging in their COM heels, it was inevitable that Microsoft would upgrade their ODBC solution: OLE DB was born. OLE DB is a set of over fifty COM classes which are intended to solve the same problems as ODBC and more. However, as Microsoft say in their own literature, OLE DB is for 'system programmers' and not 'application programmers'. In other words, OLE DB is so big, its class hierarchy so rigid and so full featured, that it is not for the faint hearted. Enter ADO: Microsoft's replacement for DAO (Data Access Objects). ADO is a layer of eleven COM classes which sit on top of OLE DB, simplifying the application programmer's task.



➤ *Figure 1*

So why should you be interested in ADO ? Perhaps the first reason is that Inprise are taking it seriously enough to include support for ADO in Delphi 5. There is no official statement (and therefore no guarantee) but Josh Dahlby (one of the development team who worked on MIDAS) has already let it slip in newsgroups, and John Kaster, Inprise's Developer Relations Manager (formerly the BDE Product Manager), has already demonstrated the ADO classes in Australia and is expected to demonstrate them in Holland in June. The second reason is that ADO is a free replacement for the BDE and, if your application is COM based, this is a much more intuitive approach to database application development. The third reason is that RDA (Remote Data Access), another Microsoft technology, is an extension to ADO and is a free alternative to MIDAS.

Getting ADO

So how do you get ADO? Well, there's a good probability you've already got it. ADO 2.1 is included with Internet Explorer 5, SQL Server 7.0, SQL Server 6.5 SP5 and Office 2000, and will be in Windows 2000. All future versions of Windows will include the most recent version of ADO. Previous versions of ADO were included with Visual Studio 6.0, Windows 98, Internet Explorer 4.0, Windows NT4 Option Pack, Visual InterDev and many version 5 Microsoft programming languages. Search for MSADO15.DLL on your hard disk to check.

Alternatively you can download ADO from Microsoft's website for free (www.microsoft.com/data/mdac2.htm). The redistributable version is 6.2Mb but you should really download the SDK instead at 37.8Mb, as it is the best source of ADO and OLE DB information.

ADO Classes

Figure 1 shows the ADO classes and how they relate to each other.

The `Connection` class represents a connection to a data source and is analogous to a cross between a `TSession` and a `TDatabase`. The `Recordset` class is used for non-parameterised queries and is analogous to a `TTable` or a `TQuery`. The `Command` class is used for executing commands and stored procedures, and opening parameterised queries, and is analogous to `TTable`, `TQuery.ExecSQL` and `TStoredProc`.

Getting Started

Like the equivalent Delphi classes, many ADO features can be learned on a 'nice to know' basis instead of a 'need to know' basis. For example, in Delphi programmers can get started using `TTable` and `TQuery` straight away without ever considering `TDatabase` or `TSession`. Delphi creates `TDatabase` and `TSession` classes as needed behind the scenes when programmers don't provide them themselves. So you can start using ADO `Recordsets` immediately without having to understand `Connections`.

As ADO is a set of automation objects, we can use early vtable binding, dispinterfaces or late `IDispatch` binding. The arguments for and against each approach are the same for ADO as for any automation object, that is, early binding gives performance, compile-time checking and the nuisance of having to specify all parameters even when you want the defaults, and late binding gives ease of development.

For all of the examples in this article early vtable binding is used, so the type library needs to be imported. To import the ADO type library, use `Project | Import Type Library | Microsoft ActiveX Data Library` and add the resulting ADODB_TLB.PAS to your uses clause.

Listing 1 shows a piece of code which opens the `Suppliers` table in

Northwind.MDB and displays the second field of every record (fields are numbered from 0) in a memo.

The code needs a little explanation. `CoRecordset.Create` returns an interface to a `_Recordset` COM object using the `CoRecordset` proxy class created when the ADO type library was imported. `Recordset.Open` takes five parameters and, as we are using vtable binding via the interface, we suffer the problems of having to specify all five parameters even when we want the defaults. The first parameter is the name of the table. An alternative is to specify an SQL string, for example `SELECT * FROM SUPPLIERS`. The second parameter specifies how to connect to the data source. In this case it is a string indicating the ODBC driver and the name of the Access MDB file. The third parameter is the cursor type. The fourth parameter is the lock type and the fifth specifies how the source parameter (the first parameter) should be interpreted (as a table name, a command text or a stored procedure).

The remainder of the code should be self-evident. Although the method names are slightly different to what we are used to, the intention is all very clear. Table 1 shows a complete listing of `Recordset` methods with their Delphi `TDataSet` equivalents.

Connections

In Listing 1 the second parameter passed to `Recordset.Open` is a connection string. When a `Recordset` is used with a connection string it automatically creates a `Connection` object and assigns the string to the `Connection`'s `ConnectionString` property. ADO uses this technique in several places to give a less rigid approach to ADO than OLE DB allows. However, you can still pass a complete `Connection` object yourself if you need greater control. Again, this is similar to the way Delphi allows programmers to create their own `TDatabase` objects.

Listing 2 shows an example of creating a `Connection` object directly. The `Connection` object used here is the simplest and simply sets the data source name to `GuysDSN`. The first parameter to `Connection.Open` is the connection string, the second is the username, the third is the password and the fourth specifies additional connection information (such as whether the connection should be made asynchronously).

```
var RS: _RecordSet;
begin
  RS:=CoRecordSet.Create;
  RS.Open('Suppliers', 'Driver=Microsoft Access Driver (*.mdb);'+
    'DBQ=C:\Program Files\Microsoft Office\Office\Samples\Northwind.MDB',
    adOpenForwardOnly, adLockReadOnly, adCmdTable);
  while not RS.EOF do begin
    Memo1.Lines.Add(RS.Fields.Item[1].Value);
    RS.MoveNext;
  end;
  RS.Close;
end;
```

➤ *Above: Listing 1*

➤ *Below: Table 1*

Name	Description	Delphi Equivalent
AddNew	Add a new record	TDataset.Insert
Cancel	Cancels a pending asynchronous Open	
CancelBatch	Cancels a pending batch update	
CancelUpdate	Cancels pending updates to the current record	TDataset.Cancel
Clone	Creates a duplicate Recordset	
Close	Closes the Recordset	TDataset.Close
CompareBookmarks	Compares two bookmarks	
Delete	Deletes the current record or group of records	TDataset.Delete
Find	Search for a record	TDataset.Locate
GetRows	Retrieve multiple records into an array	
GetString	Retrieve a Recordset as a string	
Move	Moves the record pointer	TDataset.MoveBy
MoveFirst	Moves to the first record	TDataset.First
MoveLast	Moves to the last record	TDataset.Last
MoveNext	Moves to the next record	TDataset.Next
MovePrevious	Moves to the previous record	TDataset.Prior
NextRecordset	Moves to the next recordset in a series of commands	
Open	Opens the recordset	TDataset.Open
Requery	Re-executes the query	TDataset.Refresh
Resync	Refreshes the data	TDataset.Refresh
Save	Saves recordset to a file	
Supports	Determine whether the record set supports a particular functionality	
Update	Saves changes	TDataset.Post
UpdateBatch	Writes all pending changes	

```
var
  Conn: _Connection;
  RS: _RecordSet;
begin
  Conn:=CoConnection.Create;
  Conn.Open('DSN=GuysDSN', '', '', -1);
  RS:=CoRecordSet.Create;
  RS.Open('SELECT * FROM Animations', Conn, adOpenForwardOnly, adLockReadOnly,
    adCmdText);
  Memo1.Lines.Add(RS.Fields.Item[1].Value);
  RS.Close;
  Conn.Close;
end;
```

➤ *Above: Listing 2*

➤ *Below: Listing 3*

```
try
  RS.Open('SELECT * FROM DoesNotExist', Conn, adOpenForwardOnly, adLockReadOnly,
    adCmdText);
  RS.Close;
except
  on E: EOLEException do
    for intError:=0 to Conn.Errors.Count - 1 do
      ShowMessage(Conn.Errors.Item[intError].Description);
end;
```

```
for intProp:=0 to Conn.Properties.Count - 1 do
begin
  Prop:=Conn.Properties.Item[intProp];
  strValue:=Prop.Value;
  Memo1.Lines.Add(Prop.Name+' = '+strValue);
end;
```

➤ *Listing 4*

As mentioned earlier, `Connection` objects have a lot in common with `TDatabase`. Table 2 shows `Connection` methods and their Delphi equivalents. As you can see, converting from `TDatabase`'s transaction processing to ADO's doesn't take too much head scratching.

Exception Handling

Delphi traps ADO exceptions as regular `EOLEException` objects. Unfortunately, these exceptions are not very forthcoming, so a little more effort is required. Listing 3 shows a snippet of code which handles an ADO exception using ADO's `Errors` and `Error` objects.

Properties Objects

Apart from the 'hard coded' properties found in all objects, ADO supports 'dynamic' properties in the form of `Properties` objects and `Property` objects. `Properties` objects are a collection of `Property` objects. `Command`, `Connection`, `Field`, `Parameter` and `Recordset` all have a `Properties` property. The `Properties` collection contains a very large amount of information about the object in question. The actual properties and their names depend on the OLE DB Provider in use, as different OLE DB Providers offer different services. However, typically there can be more than eighty properties in all for a `Connection` object. As such `Properties` represents a way of bundling together a whole host of information without having to create named properties in the interface definition. Listing 4 shows a simple `FOR` loop which adds all of the properties from a `Connection` object to a `TMemo`.

However, a more typical approach is to access `Properties` by their name:

```
Label1.Text:=
  Conn.Properties.Item[
  'Stored Procedures'].Value;
```

This is possible because the parameter to `Item` is an `OLEVariant` and can be an integer or a string.

Schema Information

The `Connection` class has a method called `OpenSchema` which returns a `Recordset` containing schema information. The first parameter to `Connection.OpenSchema` declares the information type you want. There are many schema information types available including tables, columns, indexes, check constraints, primary keys and stored procedures. The code in Listing 5 shows how `Recordset.OpenSchema` can be used to retrieve the same information as `TSession.GetTableNames`.

Asynchronous Processing

ADO 2.0 added support for asynchronous processing for several potentially lengthy operations. The opening of a dataset is a typical example. By default, `Open` is synchronous, but it is simply a matter of passing `adASyncFetch` in the `Options` parameter to open a table asynchronously. As a result, execution of the program continues immediately and the table is actually opened in the background. Asynchronous operations have two events: a `Will` event and a `Complete` event. The `Will` event is fired as soon as the processing starts and the `Complete` event is fired when the processing is complete.

At first sight this seems like an excellent opportunity for better performance as other tasks can be

➤ *Table 2*

Name	Description	Delphi Equivalent
BeginTrans	Begins a transaction	TDatabase.StartTransaction
Cancel	Cancels an asynchronous Execute or Open	
Close	Closes a connection	TDatabase.Close
CommitTrans	Commits a transaction	TDatabase.Commit
Execute	Executes an SQL statement	TQuery.Open, TQuery.ExecSQL, TStoredProc.ExecProc
Open	Opens a connection	TDatabase.Open
OpenSchema	Gets schema information	TQuery on system tables
RollbackTrans	Rolls back a transaction	TDatabase.Rollback

performed whilst the operation is taking place. Certainly better performance can be obtained in this way, particularly by opening many tables in parallel instead of in serial. However, a little thought should reveal that this very useful feature doesn't need to be a feature of the database system as exactly the same result can be achieved using `TThread` and generating an `OnTerminate` event when the process is complete. It is much neater having the process available as a parameter passed to `Open` but this just makes it more convenient.

Connection Pooling

ADO supports connection pooling (without the need for MTS). The default is that connection pooling is on, so you have to explicitly turn it off using a `Connection` if you don't want it. The connections can only be reused when a compatible connection is requested. In other words, a connection made by one user cannot be reused by a connection made by another user (this makes sense because otherwise you could circumvent security).

Using Command Objects

Command objects fulfil a number of roles, including opening datasets, executing SQL which doesn't return result sets, plus executing parameterised queries and stored procedures. In Listing 6 a `Command` is being used to execute a regular SQL `SELECT` statement.

What seems strange is that `Recordset` has no `Parameter` objects and therefore cannot execute parameterised queries. Listing 7 shows a variation on Listing 6 which uses a parameterised query.

Remember that the convention of using a full colon followed by a name (for example `:CUSTID`) to identify a parameter in an SQL string is a BDE convention not an SQL one.

Briefcase Model

In Delphi the briefcase model, or Recordset Persistence as ADO calls it, is achieved using `TClientDataSet.SaveToFile` and `TClientDataSet.LoadFromFile`. It is all very neat and concise and it works well, albeit using a Borland proprietary format. The ADO equivalents are `Recordset.Save` and `Recordset.Open`. Saved records can be loaded again using the `Recordset.Open` method and specifying `Provider=MSPersist`. In ADO 2.0 records are saved in a proprietary format called Advanced Data TableGram (ADTG). In ADO 2.1 there is a choice of ADTG and XML formats. In a future release HTML will also be an option.

Third Party Products

There are two third party products in the Delphi ADO market: Adonis from Cybermagic Productions (who are at www.cybermagic.co.nz/adonis) and ADOSolutio (yes, *without* an 'n' at the end) from Lectum Information Technology (at www.lectum.com). The products have fundamentally different approaches and your choice is more likely to be based on what you think the best approach is to ADO than their feature lists.

Adonis takes the approach that as a Delphi programmer you want a `TDataSet` descendant that looks and acts like something you're already familiar with. As such Adonis's class list includes `TADODatabase`, `TADODataset`, `TADOQuery`, `TADORDataSet`, `TADOSchema`, `TADOStoredProc`, `TADOTable`, `TADOUpdateSQL` and `TRDSRemoteObject`.

ADOSolutio takes the approach that if you're going to use ADO then you want your classes to look and act like ADO. As such ADO's class list includes `IMADOConnection`, `IMADODataSet`, `IMADOCommand`, `IMRDSClientDataSet` and `IMRDSDataSpace`.

For my money if Delphi 5 really does include ADO support then a product bought today is likely to be bought as a stop gap until the built-in support arrives, or to support code in older versions of Delphi. So you will probably want your stop gap to be as much like the Delphi 5 classes as possible.

Pros And Cons

So, should you make the leap from the BDE to ADO ? Well, let's examine the pros and cons.

On the up side ADO is a Microsoft technology. As such it is likely to be used by a much wider audience than the BDE and have a much greater impetus. Also as it is a Microsoft technology it is not too surprising that it is a COM technology and COM is built into the operating system and is a widely used and stable technology. In addition ADO and RDS are free and, with their introduction into every new release from Microsoft, they will

```
var
  Conn: _Connection;
  RS: _RecordSet;
begin
  Conn:=CoConnection.Create;
  Conn.Open('DSN=GuysDSN', '', '', -1);
  RS:=Conn.OpenSchema(adSchemaTables, EmptyParam, EmptyParam);
  while not RS.EOF do begin
    Memo1.Lines.Add(RS.Fields.Item['TABLE_NAME'].Value);
    RS.MoveNext;
  end;
  RS.Close;
  Conn.Close;
end;
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
var
  Conn: _Connection;
  Command: _Command;
  RS: _RecordSet;
  RecordsAffected: OLEVariant;
begin
  Conn:=CoConnection.Create;
  Conn.Open('DSN=GuysDSN', '', '', -1);
  Command:=CoCommand.Create;
  Command.Set_ActiveConnection(Conn);
  Command.CommandText:='SELECT * FROM Suppliers';
  Command.CommandType:=adCmdText;
  RS:=Command.Execute(RecordsAffected, EmptyParam, -1);
  while not RS.EOF do begin
    Memo1.Lines.Add(RS.Fields.Item[1].Value);
    RS.MoveNext;
  end;
  RS.Close;
  Conn.Close;
end;
```

soon be installed on every Windows PC. As mentioned earlier ADO is based on OLE DB which provides access to a range of data sources including databases, spreadsheets and HTML. Finally, as RDS is free and represents a way of creating n-tier database applications, it is an alternative to MIDAS.

On the down side ADO is a Microsoft technology... To some this means 'Microsoft lock-in'. Also, as it is based on COM, it is a bitter pill to swallow if your company has already committed to CORBA. You can have both COM and CORBA in the same application but it is a little contradictory and doesn't sit well with developers, managers or customers. Furthermore ADO must be installed on the client machine (even if you use RDS for n-tier applications). ADO certainly isn't small, but you could argue that if it's already on the user's PC there is no additional space requirement.

Another cause for concern is ADO's lack of support for all things non-Microsoft. The get out clause is that ADO is based on OLE DB which uses its own ODBC provider by default, so if you have an ODBC driver then you have support. However, if your database is InterBase and you're used to the level of control given by FreeIB-Components, then the ODBC driver just doesn't cut it. More worryingly, consider the Oracle OLE DB provider, written by Microsoft. It seems unlikely that Oracle will write their own OLE DB provider as they offer Oracle Objects For OLE (OO4O) and they say OO4O has features which are unique to it and are 'cumbersome or inefficient to use from other ODBC or OLE DB-based components such as ADO'. So if you are using Oracle and ADO you need to consider how responsive Microsoft will be when Oracle 9 comes out with features that OLE DB hasn't anticipated.

```
Command:=CoCommand.Create;
Command.Set_ActiveConnection(Conn);
Parameter:=Command.CreateParameter('PCOUNTRY', adBStr, adParamInput, 2, 'UK');
Command.Parameters.Append(Parameter);
Command.CommandText:='SELECT * FROM Suppliers WHERE COUNTRY=?';
Command.CommandType:=adCmdText;
RS:=Command.Execute(RecordsAffected, EmptyParam, -1);
```

➤ *Listing 7*

Conclusion

Although ADO is a relatively new technology it is based on a more mature foundation which is ultimately based on COM. As a BDE replacement it has advantages and pitfalls. However, Microsoft's attempts to ensure that ADO is on every PC in the near future, and Borland's possible support for ADO in Delphi 5, make it a technology worth watching.

Guy Smith-Ferrier is Technical Director of Enterprise Logistics Ltd., a training company specialising in Delphi, Visual Basic and Visual FoxPro. He can be contacted at gsmithferrier@EnterpriseL.com