ADO Support In Delphi 5

by Guy Smith-Ferrier

In Issue 46 I wrote an article describing how to make use of ActiveX Data Objects in Delphi 4 using the ADO type library. I mentioned then that Delphi 5 was very likely to include direct support for ADO and I am very pleased to say that that is exactly what happened. This article investigates the new ADO support in Delphi 5.

First, the bad news. The ADO components are only available in the Enterprise version of Delphi 5 (the new name for the Client/Server Edition). Fortunately, Delphi Professional developers can buy an option pack called ADO Express, which includes the ADO components, at a cost of £129. *[Editor's comment: I am very disappointed indeed that Inprise have taken this route, against specific advice to the contrary, especially as the BDE seems clearly to be on the way out. Nevertheless, there are cheaper ways to get ADO support, including the freeware TADODataSet from www.alohaoi.com/ADODB/.]*

Assuming you have the Enterprise version then there is a new `ADO` page on the palette containing the ADO classes listed in Table 1.

Getting started with ADO is as simple as getting started with the BDE. Start a new application. Drop a `TADOTable` on to a form. Set the `ConnectionString` to:

TADOConnection	A connection to an ADO data store
TADOCommand	An ADO command object
TADODataSet	A dataset retrieved from an ADO data store
TADOTable	A dataset that encapsulates a table accessed through an ADO data store
TADOQuery	Provides the means for issuing SQL against an ADO data store
TADOStoredProc	Encapsulates a stored procedure in an ADO database
TRDSConnection	A Remote Data Services connection

➤ *Table 1*

```
Driver=
  Microsoft Access Driver (*.mdb);
DBQ=C:\Program Files\
  Microsoft Office\Office\
  Samples\Northwind.MDB
```

Set the `TableName` to `Customers`, set `Active` to `True`, add a `TDataSource` and a `TBGrid` and connect them all together and run the program. *Voila!* An ADO application in seconds.

Mmm. Well, this might fool your boss but your average Delphi programmer is probably a bit more sceptical (paranoid?) and right now you should be thinking 'I'm sure there's a bit more to it than that' and there is. So let's start at the beginning.

The Beginning

Delphi 5 ships with Microsoft Data Access Components (MDAC) 2.1. MDAC 2.1 includes ADO 2.1. Unfortunately, Delphi 5 ships with purely MDAC and not the MDAC SDK. Although the Delphi 5 help describes the Delphi ADO components adequately, you are almost certain to need the additional documentation which comes with the MDAC SDK. Thankfully you can download this from the Microsoft website, at www.microsoft.com/data/download.htm, where you will need to download both the MDAC SDK 2.0 (37.8Mb) and MDAC SDK 2.1 Update (5.6Mb).

Table 2 shows the history of ADO and which products include which versions.

➤ *Table 2*

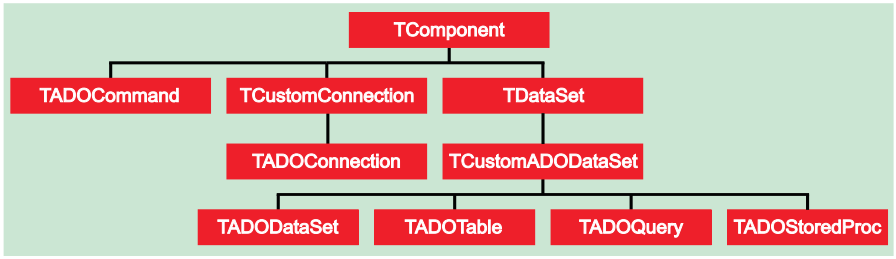
ADO Version	Released	Included With
1.1	Winter 1996	Windows 98, Internet Explorer
1.5	Autumn 1997	Windows NT4 Option Pack, Visual InterDev, many version 5 MS programming languages
2.0	Summer 1998	Visual Studio 6, many MS programming languages, a cut down version of SQL Server
2.1	March 1999	Delphi 5, Office 2000, Internet Explorer 5, SQL Server 7.0 and SQL Server 6.5 SP5
2.5	Later in 1999	Windows 2000

Sadly, Microsoft played a rather horrific trick when naming ADO versions. Instead of calling ADO v2.0 simply 'ADO v2.0' as anyone else would have, they decided to call it 'ADO v1.5 version 2.0'. The idea was that version 2.0 is sort of compatible with version 1.5 but in reality it is just simply confusing. The simplest way of determining which version of ADO you have is to drop a `TADOConnection` object onto a form and add a button with the following code:

```
Caption :=
  ADOConnection1.Version;
```

BDE Versus ADO

There are several ways to explain the new ADO support in Delphi 5. The approach that I want to take is to explain its features by comparing it with something that nearly all Delphi programmers know: the



➤ *Figure 1*

BDE Component	ADO Component
TSession	N/A
TDatabase	TADOConnection
TTable	TADOTable
TQuery	TADOQuery
TStoredProc	TADOStoredProc
TClientDataSet	Any ADO DataSet component with MSPersist OLE DB Provider

➤ *Table 3*

BDE. I take this approach because most Delphi programmers will be thinking about their current project (probably based on the BDE) and wondering how much work would be involved in converting it over to ADO. So let's start with the ADO components.

Figure 1 shows the ADO class hierarchy. Table 3 shows the BDE components and their ADO equivalents.

`TADOConnection` broadly performs the same roles that `TDatabase` does: it handles login prompts, connection strings (the ADO equivalent of alias names), transaction processing and transaction isolation levels. However, whereas `TADOConnection` handles most of the same features that `TDatabase` does it handles them all in an ADO way instead of a BDE way. This reveals the philosophy behind all of the Delphi ADO components. Whereas the ADO components do use Delphi concepts they do not use BDE concepts. So, for example, `TADOConnection.IsolationLevel` is a `TIsolationLevel` enumerated type instead of the `TOleENum` (that is, an integer) which you would get if you used ADO's Connection interface directly and therefore `TADOConnection.IsolationLevel` is implemented in a Delphi way instead of a pure COM way. But `TADOQuery` has a `Parameters` property instead of `TQuery`'s `Params` property and `TADOQuery.Parameters` is of type `TParameters` instead of the BDE equivalent `TParams` and therefore `TADOQuery` has been implemented in an ADO way instead of in a BDE way. The Delphi developers have tried to keep as much compatibility with original Delphi BDE components as possible but where they had a choice between keeping compatibility with the BDE or keeping compatibility with ADO they have chosen the latter. For my money, they have made the right choice.

Staying on the theme of 'making the right choice', I'll digress for a short moment. I am delighted to say that the ADO components *do not* hide the ADO interfaces upon which they are based. Each ADO component freely exposes the interface which it wraps around. For example, `TCustomADODataSet` (upon which all of the ADO dataset components are based) exposes a `RecordSet` property which is the ADO `RecordSet` interface it uses and `TADOConnection` exposes a `ConnectionObject` property which is the ADO Connection interface which it uses. Encapsulation purists will say that by exposing the internal workings of a class you allow programmers to subvert a class. This is true. However, the advantages and disadvantages of making the interface available are the same as making the BDE handle available in the BDE components. Although I am sure that I will receive mail on this one, I urge you to make the same decision when designing your own components.

OLE DB Providers

Figure 2 shows the ADO structure.

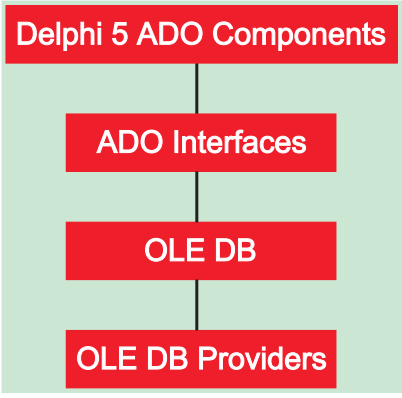
Delphi ADO components encapsulate ADO interfaces. ADO interfaces encapsulate OLE DB interfaces. OLE DB Providers (the ADO equivalent of BDE drivers) do the actual work of communicating with the data either directly or by using the RDBMS's API. Table 4 shows the OLE DB Providers which come with ADO.

Clearly, this means that Access, MS SQL Server and Oracle all have direct support and all other RDBMSs have support only via ODBC or via an OLE DB Provider supplied by the relevant vendor. As I mentioned in the last article the Oracle OLE DB Provider is written by Microsoft and Oracle have no intention of writing their own OLE DB Provider as they believe the correct solution is to use Oracle Objects For OLE (OO4O). You can draw your own conclusions from that. InterBase fans will notice that there is no OLE DB Provider for InterBase. Fortunately, one is promised for InterBase 6.

Connection Strings

Connection strings are the life blood of ADO. Connection strings

➤ *Figure 2*



Provider	Driver	Description
ODBC Drivers	MSDASQL	Existing ODBC drivers
Jet 4.0	Microsoft.Jet.SQLOLEDB.4.0	MS Access databases
SQL Server	SQLOLEDB	MS SQL Server databases
Oracle	MSDAORA	Oracle databases
Directory Services	ADSDSOObject	Resource data such as Active Directory (useful in NT5)
Index Server	MSIDXS	MS Index Server (indexed data on web sites)
Site Server Search		MS Site Server (data on websites)
Data Shape	MSDataShape	Hierarchical record sets (master/detail record sets)
Persisted Recordset	MSPersist	Locally saved record sets (briefcase applications)
Simple Provider	MSDAOSP	For creating your own providers for simple text data

➤ *Table 4*

are to ADO what alias names are to the BDE. All ADO datasets have a connection string and it can be as simple as:

```
DSN=Northwind DSN
```

where `Northwind DSN` is an ODBC Datasource Name which I have set up for the Northwind.MDB Access database supplied with Access, Visual Basic and Microsoft Office.

However, unless you are using ODBC it is unlikely that you will get away with such a simple connection string. ADO looks for four arguments in the connection string (arguments are separated by semi colons): provider, file name, remote provider and remote server. The last two are used for RDS which I am not covering in this article. Provider is the name of the OLE DB Provider and it defaults to `MSDASQL` (the ODBC OLE DB Provider). File name is a Data Link file which I will come back to later. All other arguments (such as the DSN argument in the previous example) are passed on from ADO to the OLE DB Provider. Different providers look for different arguments, but Table 5 shows the arguments which the ODBC OLE DB provider looks for.

The SQL Server OLE DB Provider simply looks for `Server`, `Database`, `UID` and `PWD` arguments.

Delphi's ADO components all have a `ConnectionString` property which is a string into which you can just type your connection string. Alternatively, `ConnectionString` can be built using the `ConnectionString` editor which ultimately links to Microsoft's own Data Link editor. In real world development, though, you are just as unlikely to set `ConnectionString` properties on individual dataset components as you are to set the `DatabaseName` property of a `TTable` or a `TQuery` to an alias name for BDE application development. Instead you would create a `TADOConnection` component just as you would have created a `TDatabase` component. You set the `TADOConnection` component's `ConnectionString` property and then set each ADO component's `Connection` property to the `TADOConnection` component. `TCustomADODataSet`'s `Connection` and `ConnectionString` properties are mutually exclusive, so setting one property clears the other. `TADOConnection` also has a `Provider` property which is the name of the OLE DB Provider used. This is automatically set after the `ConnectionString` is changed.

➤ *Table 5*

DSN or Data Source	The Data Source Name
Driver	The ODBC Driver Name
DefaultDir	The directory for dBase files
DBQ	The path and name of the database
Server	The server name
Database	The database name
UID	The user ID
PWD	The password for the user ID

Data Link Files

Data Link files are simply connection strings stored in a file. The file is in INI file format and has the .UDL extension. As such you can see UDL files as small pieces of IDAPI32.CFG. In BDE terms IDAPI32.CFG is the equivalent of all of the UDL files on a hard disk (plus additional global configuration information). However, whereas IDAPI32.CFG is considered by most programmers to be a private file which is rather definitely the domain of the application programmer or, perhaps, the DBA, Microsoft is not presenting UDL files in the same light. Microsoft appears to positively encourage users to dive into UDL files and start fiddling with their contents. Maybe this is just a case of getting used to a new philosophy, but right now this has to be my number one reason for not using UDL files.

TADOConnection

As has already been mentioned, `TADOConnection` plays the same role as `TDatabase`. `TDatabase`, however, is a BDE component and does not

feature in an ADO application. Table 6 shows the TADOConnection properties.

Although TADOConnection and TDatabase both inherit directly from TCustomConnection, TCustomConnection has very few properties and methods and so TADOConnection and TDatabase share little in common syntactically. Specifically, TADOConnection does not support the following TDatabase properties: AliasName, DatabaseName, Directory, DriverName, Exclusive, Handle, HandleShared, IsSQLBased, KeepConnection, Locale, Params, Session, SessionAlias, SessionName, ReadOnly, Temporary, TraceFlags and TransIsolation.

This isn't a criticism of the ADO components, instead it is intended to provide an indication of the fact that we're not in Kansas anymore. Of course, to present TADOConnection simply in terms of its compatibility with TDatabase is to deny the features it has which TDatabase does not. Inherent in all ADO components is the Properties property which is a direct link to ADO's Properties collection of dynamic properties. This property gives you access to a mile of additional properties which are dependant not only on the ADO component to which they refer but also to the OLE DB Provider in use. Listing 1 shows the code which reveals all of the dynamic properties in a TADOTable.

Alternatively, you can access dynamic properties directly by their name instead of by their ordinal position. The following piece of code allows you to see whether the ADO recordset can fetch backwards:

```
Memo1.Lines.Add(
  ADOTable1.Properties[
  'Fetch Backwards'].Value);
```

➤ *Listing 1*

```
var
  intProperty: integer;
  strValue: string;
begin
  Memo1.Clear;
  for intProperty:=0 to ADOTable1.Properties.Count - 1 do begin
    strValue:=ADOTable1.Properties[intProperty].Value;
    Memo1.Lines.Add(
    ADOTable1.Properties[intProperty].Name + ' = ' + strValue);
  end;
end;
```

Attributes	Specifies automated transaction behavior
CommandCount	Number of associated command components
Commands	Array of associated TADOCommand components
CommandTimeout	Specifies number of seconds to attempt execution of a command (30)
Connected	Specifies whether or not a connection is active
ConnectionObject	The ADO Connection interface
ConnectionString	Connection string
ConnectionTimeout	Specifies number of seconds to attempt opening of a connection (15)
ConnectOptions	Specifies whether a connection is synchronous or asynchronous
CursorLocation	Specifies whether to use client-side or server-side cursor library
DataSetCount	Number of active datasets associated with the connection component
DataSets	Array of active datasets associated with the connection component
DefaultDatabase	The database used if the database in ConnectionString is unavailable or not specified
Errors	The ADO Errors collection from the most recent error
InTransaction	Indicates whether a transaction is in progress
IsolationLevel	Specifies the transaction isolation level for transactions
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection
Mode	Indicates the permissions available to a connection
Properties	The ADO Properties collection of dynamic properties
Provider	Specifies the provider for the ADO connection
State	Indicates the current state of the ADO connection
Version	Indicates the version of ADO used

➤ *Table 6*

Another very useful TADOConnection feature is the ability to access schema information easily. TADOConnection has a method called OpenSchema which accepts four parameters. The first parameter is the type of schema information to retrieve. There are over 40 possible values to enter for this first parameter but, to give you some ideas, the values allow you to specify a list of tables, a list of columns for a given table, a list of primary keys, a list of indexes, a list of views and a list of SQL features supported. The second and third parameters qualify the information being retrieved and the fourth parameter is the TADODataSet into which the resulting information is placed. So if you execute the following code:

```
ADOConnection1.OpenSchema(
  siTables,
  EmptyParam, EmptyParam,
  ADODataSet1);
```

and connect ADODataSet1 to a TDataSource and connect a grid to

the TDataSource you will see a list of tables for the current connection.

As mentioned earlier TADOConnection has properties and methods for transaction processing. These serve to illustrate another fundamental difference between ADO and the BDE.

TADOConnection has BeginTrans, CommitTrans and RollbackTrans methods and IsolationLevel and InTransaction properties. If you are familiar with transaction processing with TDatabase you can guess how these might work. One difference, however, is that TADOConnection.BeginTrans returns the transaction nesting level of the new transaction and thus allows you to nest transactions. But the difference between ADO and the BDE that I want to highlight is that ADO makes no attempt to add transaction processing where the driver does not already support it.

Whereas the BDE attempts to add transaction processing (of a sort) for Paradox and dBase databases, ADO makes no such attempt and simply returns '*The operation requested by the application is not supported by the provider*'.

The same is true for bidirectional cursors. In general this is true for all features in ADO and therefore not only is the level of portability of your application from one database engine to another potentially lower than an application based on BDE but also ADO places a greater responsibility on the programmer for understanding the different databases and their drivers than the BDE does.

ADO DataSets

Delphi 5's ADO data set components all inherit from TCustomADODataSet, which does nearly all of the work for TADOTable, TADOQuery and TADOStoredProc.

These three components actually do very little and simply add one or two properties or methods which are specific to TTable, TQuery or TStoredProc. As such it is simpler to look at TCustomADODataSet than at each of the three components in turn. Table 7 shows the properties of TCustomADODataSet.

BlockReadSize	Determines how many record buffers are read in each block
CacheSize	Determines how many rows the recordset keeps in its buffer (1)
CanModify	Indicates whether the underlying recordset permits write access
CommandText	Specifies a command to be executed
CommandTimeout	Specifies number of seconds to attempt execution of a command (30)
CommandType	Specifies the type of command to execute
Connection	Specifies the ADO connection component to use
ConnectionString	Specifies the connection information for the data store
CursorLocation	Specifies whether to use client-side or server-side cursor library
CursorType	Type of cursor to use for a recordset
DataSource	Alternative to MasterSource property
ExecuteOptions	Set of TExecuteOptions
Filter	Textual filter condition
FilterBookmarks	Filters a data set to just rows with pre-defined bookmarks
Filtered	Activates the filter condition
FilterGroup	Filters a recordset based on row update status
IndexDefs	Collection of index definitions
IndexFieldCount	Number of fields that comprise the current key
IndexFields	Array of fields of the current index
IndexName	Name of currently active index
LockType	The lock type used when opening a dataset (ltOptimistic)
MarshalOptions	Specifies which records are marshaled back to the server
MaxRecords	Maximum rows to return in a result set (0, i.e. no limit)
ParamCheck	Should the Parameters list be regenerated when the SQL changes
Parameters	Collection of parameters for SQL statement
Prepared	Should the command be prepared before execution (False)
Properties	The ADO Properties Collection object
Recno	The ordinal position of the record within the result set
RecordCount	Number of rows in the result set
Recordset	The ADO Recordset object
RecordsetState	The current state of the ADO dataset component
RecordSize	The size of a record in the dataset
RecordStatus	The status of the current record
Sort	Specifies the sort order of the recordset
StoreDefs	Indicates whether the table's field and index definitions persist with the data module or form

➤ *Table 7*

Clearly, TCustomADODataSet and TDBDataSet (on which TTable, TQuery and TStoredProc are based) inherit from TDataSet and so a very large part of your regular usage of these components will require no change in moving from the BDE to ADO. Methods and properties such as Open, First, Next, Prior, Last and EOF are common to all datasets. However, TADOTable is not compatible with TTable and vice-versa.

To give you an idea of the level of portability you can expect, the

BDE Tool	Works With ODBC Drivers via BDE	Comments / Equivalents
Database Desktop	No	Database Desktop cannot be used with ADO as it uses BDE aliases only
Database Explorer	Yes	Database Explorer can only be used with ADO ODBC data sources
SQL Monitor	No	SQL Monitor is fundamentally based on SQL Link drivers so will not work with ADO. Use Visual Studio Analyzer instead
DataPump	Yes	DataPump can only be used with ADO ODBC data sources. TBatchMove cannot be used with ADO because its Source property is TBDEDataSet and its Destination property is TTable.
SQL Builder	Yes	SQL Builder can only be used with ADO ODBC data sources. However, SQL Builder is not on the context menu for TADOQuery.
InstallShield Express For Delphi 5	N/A	InstallShield can install Delphi 5 ADO applications but does not have any specific provision for installing ADO itself
BDE Administrator	N/A	ODBC Data Source Administrator

➤ *Table 8*

following `TDBDataSet` properties are not supported by `TCustomADODataSet`:

`AutoRefresh`, `CacheBlobs`, `CachedUpdates`, `Database`, `DatabaseName`, `DBHandle`, `DBLocale`, `DBSession`, `ExpIndex`, `KeySize`, `Locale`, `SessionName`, `UpdateMode`, `UpdateObject`, `UpdateRecordTypes`, `UpdatesPending`.

Furthermore the following `TTable` properties are not supported by `TADOTable`:

`DefaultIndex`, `Exclusive`, `Exists`, `Handle`, `IndexFiles`, `KeyExclusive`, `KeyFieldCount`, `TableLevel`, `TableType`.

One of the differences which could have a great impact on portability is the loss of cached updates. Clearly cached updates are implemented by the BDE and so you couldn't expected to see them implemented in ADO components. However, all is not lost. ADO supports 'batch updates' which are very similar to cached updates. Instead of having a `CachedUpdates` property which is set to `True` ADO's batch updates are initiated by opening a data set which has `LockType` set to `ltBatchOptimistic`. Instead of calling `ApplyUpdates` or `CancelUpdates` you would call `UpdateBatch` or `CancelBatch`.

BDE Tools

One of the downsides of switching to ADO is that you will have to throw away many of your favourite BDE tools. Table 8 shows a list of BDE tools and how applicable they are to ADO.

The general rule of thumb is that if the tool supports ODBC and you have an ODBC driver for your data source then you can still use the tool.

ADOX, ADOMD And JRO

Sometimes I feel that there is nothing that Uncle Bill likes more than a good acronym. Like any good Microsoft technology ADO is full of them. ADOX, ADOMD and JRO are separate but related object models to ADO. ADOX is *ADO Extensions For DDL And Security*. It includes Catalog, Tables, Indexes, Keys, Columns, Views, Procedures, Users, Groups and more and is a database independent solution to the myriad of different SQL DDL (and to some extent DCL) dialects. ADOMD is *ADO Multi-Dimensional* and is designed to integrate with On Line Analytical Processing servers. ADOMD fulfils the same purpose as Delphi's own decision cubes and Excel's pivot tables. JRO is *Jet and Replication Objects* and it allows you to manipulate replication features and other features of an Access database (no other database is supported).

Unfortunately, Delphi 5 has no direct support for any of these technologies. There are no `TADOView` components and no `TADOUser` components, but I would put good money on them arriving in a future version. Until then you can solve the problem of lack of direct support using the same method as I did in my previous article by importing the appropriate type libraries. Alternatively some Delphi third party vendors support ADOX directly.

Summary

It is great to see Borland supporting ADO in Delphi 5. It is a testament to the design of the VCL's data access components that the user interface can so easily switch from existing BDE components to the new ADO components.

However, my verdict on the portability of code from BDE components to ADO components is not as good. Obviously the level of portability can only be assessed on a case by case basis but I would be surprised if any real world application survived without any changes. In addition the transition from BDE to ADO requires you not only to learn new tools but eliminates the experience you have gained from years of using the BDE. This isn't to say that you shouldn't use ADO but simply to point out how much work is involved in learning a new database access layer.

In this article I hope to have given you an idea of what you can expect from Delphi 5's ADO support but I have not covered ADO in depth and you should expect more articles on the myriad of ADO topics in future issues.

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