## Column: Visual Basic for Dataheads
## Figures: NONE
## File for Companion Disk: 06VBDH.ZIP

Data Data Everywhere

Whil Hentzen

What good is a programming language if you can't get to data somehow? Here is probably the biggest difference between Visual Basic and Visual FoxPro – VB doesn't have a native data engine. Thus, the way you talk to a data set is significantly different. However, you'll find that once you've made the connection to a data set, what you do next is more familiar than you would have thought. In this article, I'll discuss the fundamental concept of how you go about accessing data in VB, and describe a couple of ways of making it happen. For those of you interested in the buzzwords of the month, I'll discuss the various TLAs involved in data access: ADO, DAO, RDS, RDO, UDS, ABC, XYZ and do-ray-mi; and why you want to use ADO. Then I'll show you how to get started with ADO quickly and easily.

*\\\ July header
Now that you understand why you want to use ADO and how to get a connection started, I'll describe how to do real-world tasks with ADO.

Visual Basic is a general purpose programming language that has more and more been used as a front end to database applications. As such, it does not have a native data manipulation language – instead, data access is a feature that's been cobbled on to it, somewhat as an afterthought. Before you disparage this approach, you should know a little about how this state of affairs came about.

A brief history of data access in Visual Basic

If you look back a dozen years or so, you had three general choices when writing a database application. You could use one of those "Xbase" tools like dBASE, Clipper, or Fox. You could use a non-Xbase tool like Paradox or Rbase. Or you could use BASIC and manipulate text or binary files by hand. Many programmers chose the third route – not only hand-coding screens like Xbasers did in the early days, but also hand-coding data access. Instead of the Xbase "USE" command to open and gain access to a file, you had to go through a series of commands to identify the file, grab a file handle, open the file, write bytes to the file, reposition the pointer in the file, and so on – much like you can do with low level file functions in Fox. Imagine if you were limited to LLFFs in Fox for all your data access!

When Visual Basic first arrived in the early 90's, hand-coding of screens went away – you simply drew objects with a Form Designer tool. Each subsequent version of VB added more power and capability – so no one noticed that they were still hard-coding the data access portion of the application.

In the mid-90's, more powerful data access mechanisms were added to Visual Basic, to hide the hand-coding of data access much like the Form Designer hid the hand-coding of screens. These mechanisms have gone through a number of iterations, and Microsoft is not done yet.

DAO, ADO, 123, Hike!

It's not worth spending a lot of time going back to the beginning of data access in VB, but the last two mechanisms are worthy of discussion. Many VB applications in use today were written using a mechanism called Data Access Objects, or "DAO". DAO provided access to relational databases like Microsoft Access and SQL Server. It's been superceded by ActiveX Data Objects, or ADO. ADO is actually a front end for a broader technology called OLE DB.

OLE DB has come about due to the explosion of non-relational data sources, such as email, images, movies, sound files, HTML documents, and so on. Just as ODBC is a generic API (application programming interface) for all sorts of relational databases, OLE DB is a generic API for all sorts of data. However, as has been said over and over again, writing to the OLE DB API is "just too hard" so ADO was put together to provide an easy to use interface to OLE DB.

While you can still use DAO, ADO is the current technology of choice, and unless you are faced with having to support old VB apps, I'd suggest you ignore DAO and learn ADO. That's what I'm going to do here.

What is ADO?

ADO is just a DLL that sits in your WINDOWS\SYSTEM or WINNT\SYSTEM32 directory. It provides services just like any other DLL provides functionality. You can drop an ADO control on a form, just like you can drop a Calendar OCX or a DLL that provides serial communications. No real magic there – but it's not terribly clear to those just getting started – or to those of us who are used to having data access built into the product like we are with Fox.

So, like Jet, the data access engine used in Microsoft Access, ADO is just an engine – but it one that provides the ability to talk to a variety of data sources, as opposed to just an MDB file like Access. This data engine, fortunately, was not written by the same people who wrote Jet – instead, it was written by the folks who built and enhanced the Fox data engine. Yes, an ADO recordset is, in many ways, an abstraction of a Visual FoxPro cursor!

Connections

When you issue a "USE" command in Fox, an awful lot happens behind the scenes, but eventually you're attached to a DBF file. Implicit in this scenario is that the file you're "USEing" is a DBF with a specific file structure. Other than that, a DBF is essentially a dumb file just sitting on disk. If you're using a DBF that's part of a DBC, there's a bit of additional complexity in that the DBC inserts a layer of information in between you and your data, but the Fox engine handles all that transparently. Thus, you can USE any kind of database as long as it's got a .DBF file structure.

With ADO, you can connect to a variety of data sources that can be structured a whole bunch of different ways, and in fact, can be housed in an entirely different environment, such as a SQL Server database or an email program. Not only that, but it is theoretically possible to write a data access layer in an application, and then attach to it from a different front end. For example, as the story goes, you could write a middle tier in Visual FoxPro using ADO to talk to any old back end, and then use a front end written in, oh, say, HTML or Visual Basic. I don't know of anyone who has done this successfully (in other words, had a legitimate business need for this architecture, successfully implemented it with a bare minimum of compromises, and finally, actually money at it), but it's an interesting intellectual exercise, and somewhere down the road I imagine it will become useful.

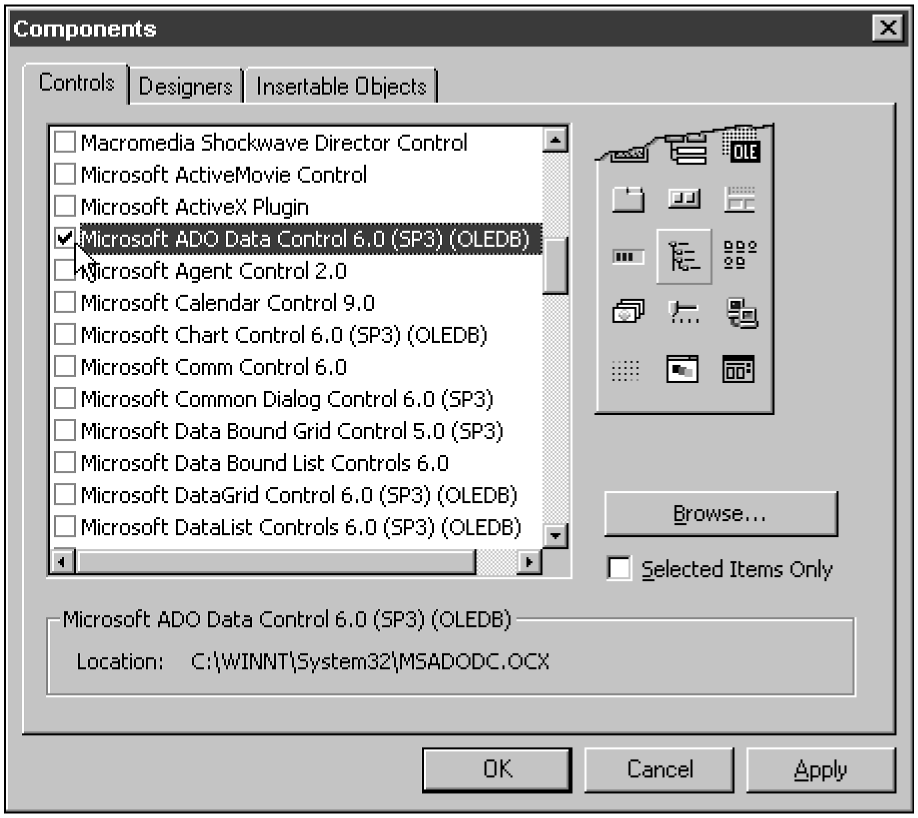
First, I'm going to create a simple data entry and navigation form using the ADO Data Control, and in doing so, create a connection so that you can see where it fits in with the larger scheme of things. The ADO Data Control acts much like a wizard – set a few properties and you're done. However, just like a wizard, it's easy to use it without understanding what's underneath, and thus your usage is superficial and limited. For this reason, I'll then take apart the connection, and build one programmatically so that you can see how each piece works.

Just like every introductory demonstration, I'll use Microsoft Access 2000 as the back end. Oh, get that gloomy look off your face. First of all, it's more likely that you've got a copy of Access lying around than SQL Server, and second, there's less overhead and infrastructure to worry about. Once you've got the basics down, you can upgrade your choice of back end and deal with the added complexities then.

Setting up a form with an ADO Data Control

This example uses the MG.MDB database provided in this month's subscriber downloads. It contains about six tables having to do with an automobile car club, including tblCars, which I'll use in this first example.

First, after opening Visual Basic, you'll need to add the ADO Data Control to your Toolbox. This control comes with Visual Basic, but is not part of the intrinsic Toolbox. Right-click on the Toolbox, select the Components menu option, and select the Microsoft ADO Data Control 6.0 (OLE DB) item in the list box in the Controls tab as shown in Figure 1. After you check the check box and select OK, you'll have another control on your Toolbox.


Components

Controls | Designers | Insertable Objects

- [ ] Macromedia Shockwave Director Control
- [ ] Microsoft ActiveMovie Control
- [ ] Microsoft ActiveX Plugin
- [✓] Microsoft ADO Data Control 6.0 (SP3) (OLEDB)
- [ ] Microsoft Agent Control 2.0
- [ ] Microsoft Calendar Control 9.0
- [ ] Microsoft Chart Control 6.0 (SP3) (OLEDB)
- [ ] Microsoft Comm Control 6.0
- [ ] Microsoft Common Dialog Control 6.0 (SP3)
- [ ] Microsoft Data Bound Grid Control 5.0 (SP3)
- [ ] Microsoft Data Bound List Controls 6.0
- [ ] Microsoft DataGrid Control 6.0 (SP3) (OLEDB)
- [ ] Microsoft DataList Controls 6.0 (SP3) (OLEDB)

Browse...

[ ] Selected Items Only

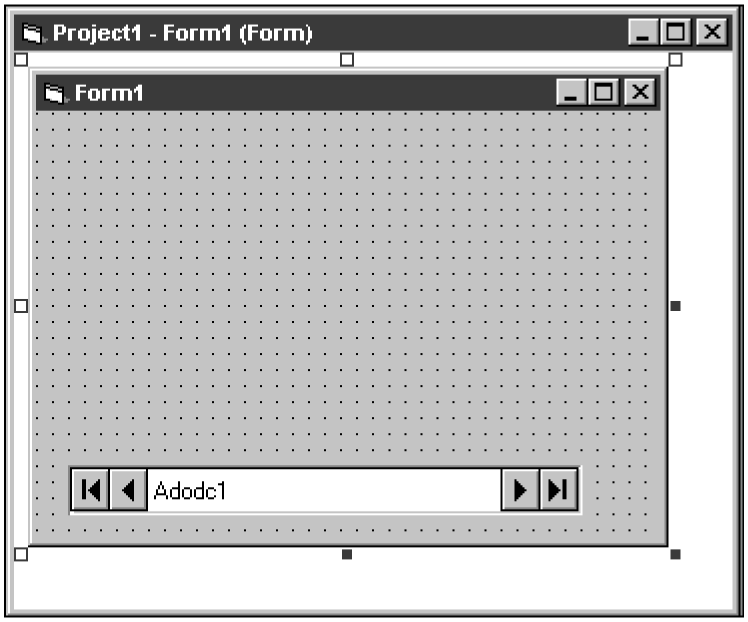
Microsoft ADO Data Control 6.0 (SP3) (OLEDB)

Location: C:\WINNT\System32\MSADODC.OCX

OK | Cancel | Apply

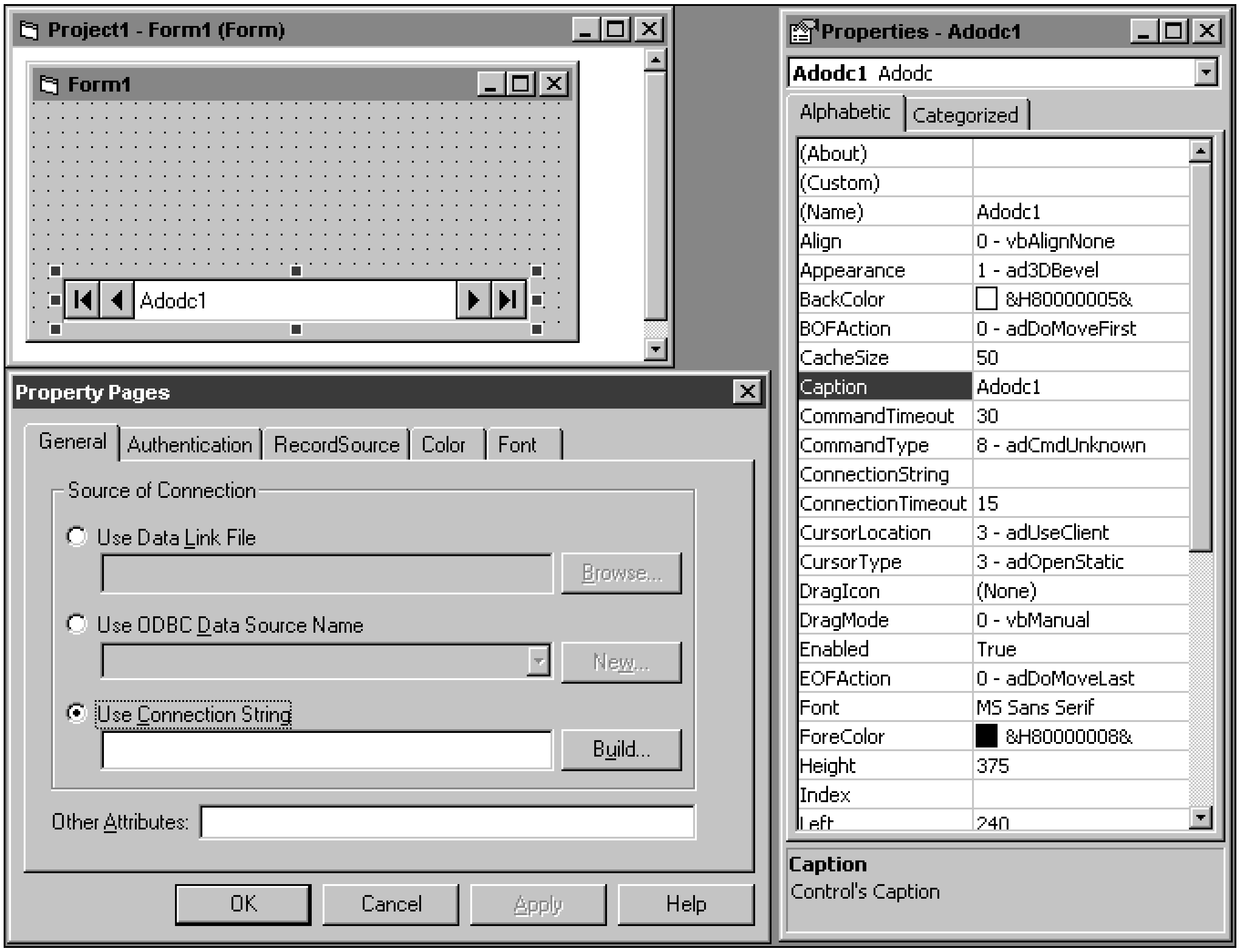
## 06VBDH01.TIF
Figure 1. Add the Microsoft ADO Data Control 6.0 to your Toolbox before starting out.

Next, open a new project (Standard EXE) and a blank form. Drop the ADO Data Control on the form. You'll get a navigation bar as the visible part of the control, as shown in Figure 2.



## 06VBDH02.TIF
Figure 2. Dropping the ADO Data Control 6.0 on your form will display a navigation bar.

Next, it's time to connect this control to a data source. First, right-click on the ADO Data Control and select the ADODC Properties menu command. The Property Pages dialog is opened. Note that this dialog is not the same as the Properties Window that is also displayed, as shown in Figure 3. (You can also open the Property Pages dialog by double-clicking on the Custom property in the Property Window, or clicking on the ellipses button to the right of the Custom property.)



Source of Connection

○ Use Data Link File

○ Use ODBC Data Source Name

● Use Connection String

Other Attributes:

Adodc1 Adodc

(About)	
(Custom)	
(Name)	Adodc1
Align	0 - vbAlignNone
Appearance	1 - ad3DBevel
BackColor	□ &H80000005&
BOFAction	0 - adDoMoveFirst
CacheSize	50
Caption	Adodc1
CommandTimeout	30
CommandType	8 - adCmdUnknown
ConnectionString	
ConnectionTimeout	15
CursorLocation	3 - adUseClient
CursorType	3 - adOpenStatic
DragIcon	(None)
DragMode	0 - vbManual
Enabled	True
EOFAction	0 - adDoMoveLast
Font	MS Sans Serif
ForeColor	■ &H80000008&
Height	375
Index	
Left	240

Caption
Control's Caption

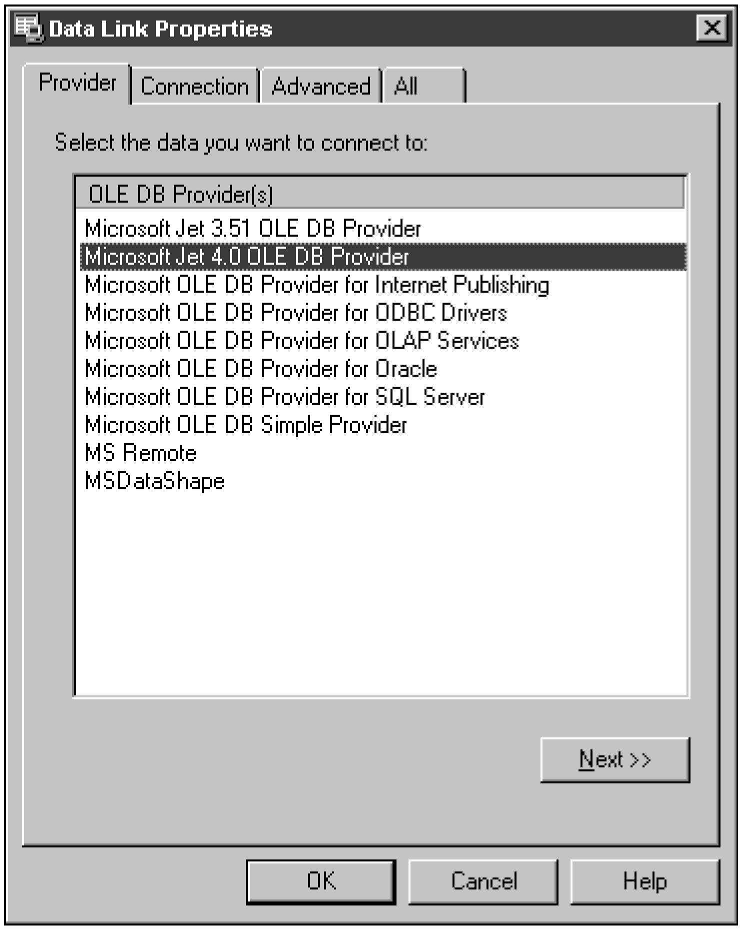
## 06VBDH03.TIF
Figure 3. Open the Property Pages dialog by right-clicking on the ADO Data Control and selecting ADODC Properties.

The Property Pages dialog acts like a “mini-wizard” where you are guided through each piece of building the connection. (I’ll explain what all of these options are for later.) Select the Use Connection String option button, and then click on the Build… button. The Data Link Properties dialog opens, and a list of OLE DB Providers is displayed as shown in Figure 4.

Remember that OLE DB is an API to a variety of data sources, just like ODBC is. The only difference is that ODBC basically attaches to relationally formatted data while OLE DB is more robust. Pick the OLE DB provider you want – in this case, it will be Jet 4.0.

You may be wondering where these OLE DB Providers come from. They just sort of come along for the ride. Some are installed with Windows, and others with a specific application. For example, when you install Office 2000 Pro (or, at least, Access 2000), you get the Jet 4.0 OLE DB Provider automatically.

Then, press the Next button in order to identify which Jet 4.0 database you want to work with.



Select the data you want to connect to:

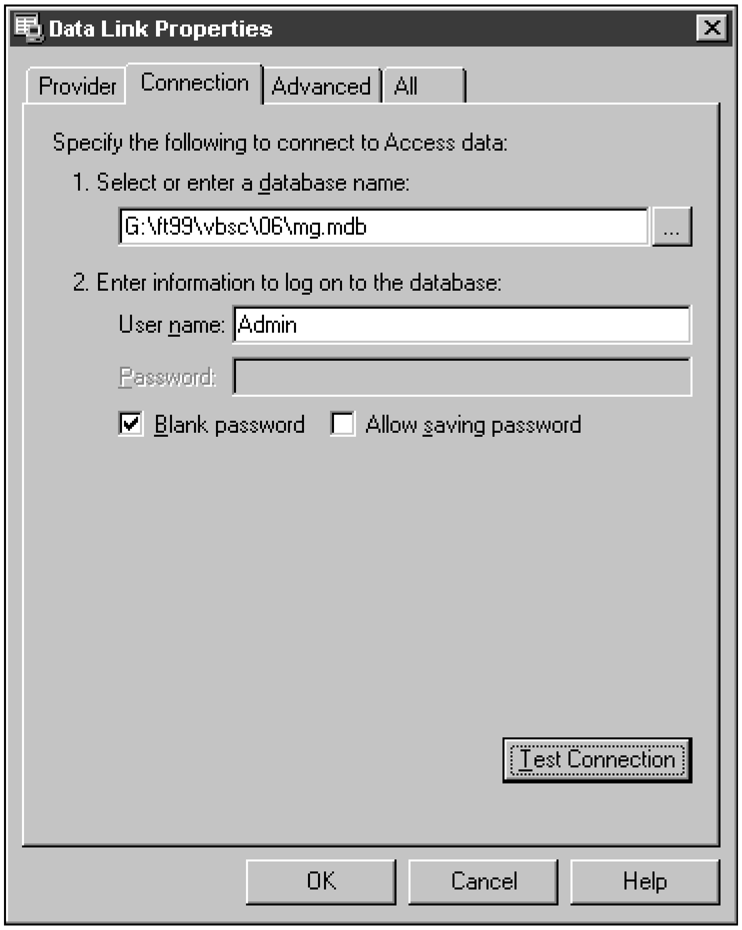
OLE DB Provider(s)

- Microsoft Jet 3.51 OLE DB Provider
- Microsoft Jet 4.0 OLE DB Provider
- Microsoft OLE DB Provider for Internet Publishing
- Microsoft OLE DB Provider for ODBC Drivers
- Microsoft OLE DB Provider for OLAP Services
- Microsoft OLE DB Provider for Oracle
- Microsoft OLE DB Provider for SQL Server
- Microsoft OLE DB Simple Provider
- MS Remote
- MSDataShape

## 06VBDH04.TIF
Figure 4. The Data Link Properties dialog displays all available OLE DB Providers.

The contents of the Connection tab of the Data Link Properties dialog will vary according to which kind of OLE DB provider you selected in the previous tab. Note that in Figure 5, I've already selected the MG.MDB Access 2000 database that's residing somewhere deep in the bowels of drive G. Also note that the OLE DB Provider for Jet 4.0 knew to ask for a User name, a password, and to allow a couple of password-related options.

Once you've entered or selected a database, you may want to make sure that ADO can talk to it by pressing the Test Connection button.



Specify the following to connect to Access data:

1. Select or enter a database name:

G:\ft99\vbsc\06\mg.mdb

2. Enter information to log on to the database:

User name: Admin

Password:

[✓] Blank password [ ] Allow saving password

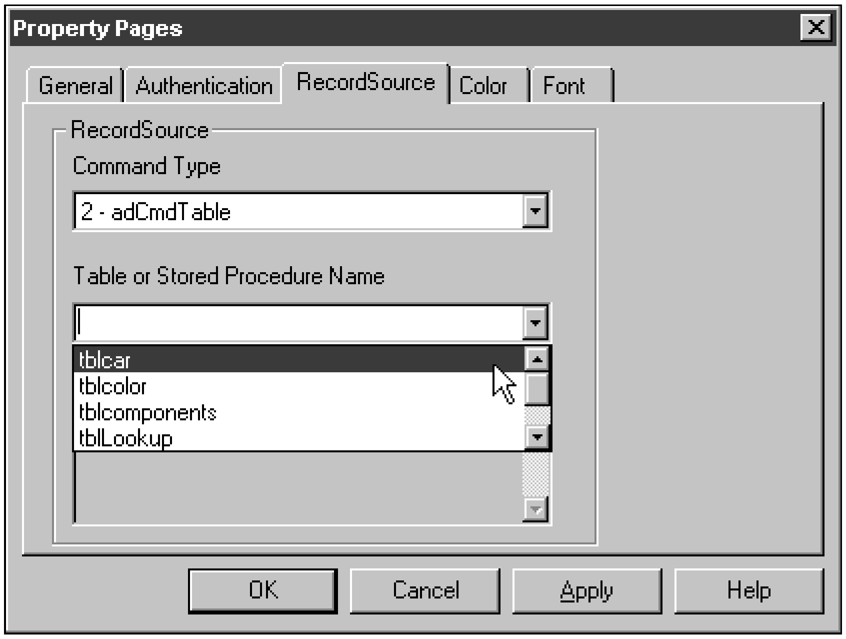
## 06VBDH05.TIF
Figure 5. Use the Connection tab to select the specific data source once you've selected an OLE DB Providers.

Finally, press the OK button, and you're all set – you'll be retruend to the Property Pages dialog, and you'll see a string of text entered into the text box below the option button. The whole string will read something like this (all on one line in the text box, of course):

```
Provider=Microsoft.Jet.OLEDB.4.0;
  Data Source=G:\ft99\vbsc\06\mg.mdb;
  Persist Security Info=False
```

You'll see that the various items you selected are identified in the Connection String, including a hard-coded Data Source. That's why I didn't provide a sample form with this application – you most likely wouldn't have a drive G with the same directories I've got on my machine.

Finally, you'll need to identify a RecordSource. Select the third tab in the Property Pages dialog, change the Command Type to adCmdTable, and when you open up the Table or Stored Procedure Name combo, you'll see all of the tables in the MG.MDB database displayed. Pick the table of interest – in Figure 6, I've picked tblCar.



RecordSource

Command Type: 2 - adCmdTable

Table or Stored Procedure Name:

tblcar
tblcolor
tblcomponents
tblLookup

## 06VBDH06.TIF
Figure 6. Use the RecordSource tab to select the actual data in the database you picked in the previous step.

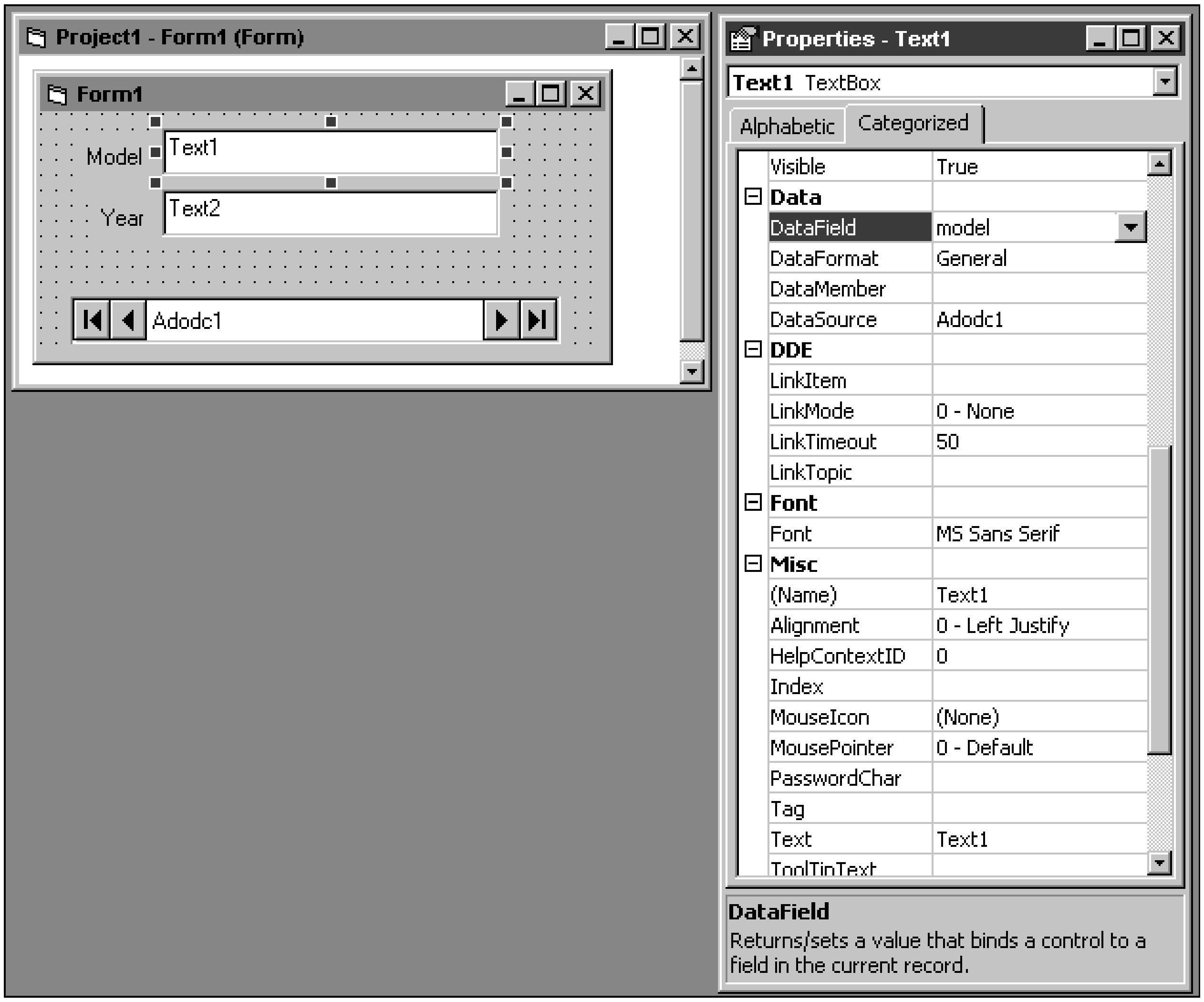
That's it for setting up the connection. Now it's time to put something more interesting on the form than just a data control. How about a text box or two???

Connecting controls to an ADO record set

Drop a couple of labels and text boxes on your form. In order to bind a text box to a field in a record source, you need to specify two properties. The first is to identify which ADO record set contains the field you want. This may seem a bit odd at first, until you realize that an ADO record set is much like a View in a VFP Data Environment. You can have several Views in a DE, and you can have several ADO record sets in a VB form. You'd just drop several ADO Data Controls on the form (and they'd be named, by default, ADODC1, ADODC2, and so on.)

In Figure 7, I've sorted the Properties window with the Categorized tab, so that I can see the Data properties all together and segregated from the other properties.

Clicking on the DataSource property will open a combo box that lists all of the available ADO record sets – in this case, the combo box only had one entry, ADODC1. Then, click on Data Field. The combo box will be populated with all of the fields that are in the record source you specified. In this example, all of the fields in the tblCar table are available. If you created a different record source, say, by a SQL Select command, that resulted in a four field result set, only those four fields would be displayed in the DataField property combo.



Text1 TextBox

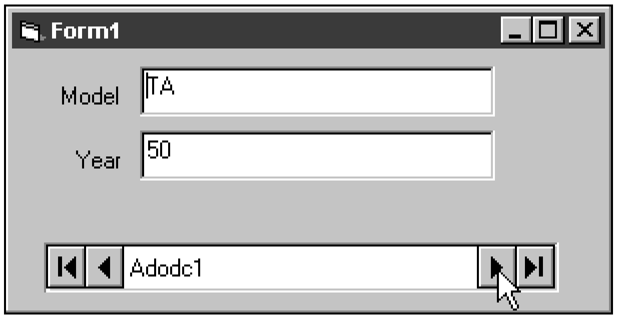
Visible	True
⊟ Data	
DataField	model
DataFormat	General
DataMember	
DataSource	Adodc1
⊟ DDE	
LinkItem	
LinkMode	0 - None
LinkTimeout	50
LinkTopic	
⊟ Font	
Font	MS Sans Serif
⊟ Misc	
(Name)	Text1
Alignment	0 - Left Justify
HelpContextID	0
Index	
MouseIcon	(None)
MousePointer	0 - Default
PasswordChar	
Tag	
Text	Text1
ToolTipText	

DataField
Returns/sets a value that binds a control to a field in the current record.

## 06VBDH07.TIF
Figure 7. Set the DataSource and DataField properties of a text box in order to attach it to a specific field in a database.

Do the same for the Year text box, and save your form.

In order to see the Data Control in action, run the form by clicking on the square button in the toolbar. You'll see the form as shown in Figure 8.



Model	TA
Year	50

## 06VBDH08.TIF
Figure 8. You can use the intrinsic controls in the ADO Data Control on the form in order to navigate to the first, next, previous and last records in the table.

Admittedly, this isn't very fancy, but it gives you two essential abilities. The first is to navigate through a record set. You can press the four buttons on the data control to move to the first, previous, next, and last records in the record set. The other is to edit a value. This isn’t immediately obvious, since there is no Edit or Save button, but the text boxes are indeed live. Change a value, move to another record, and voila!, your change has been committed.

Revisiting the record set idea

OK, you’re probably not going to pull this code out of this article and use it in an application. But it demonstrates the basic idea of an ADO record set – the new way for Visual Basic applications to address data. A record set, then, is just a cursor in Visual Basic clothing. It doesn’t have to be the whole table, and it can contain data from more than one table as well.

But an ADO record set is better than a cursor – it’s actually an object, and as such has properties that you can manipulate and methods that you can call in order to work with the record set. For example, the AbsolutePosition property is much like the Recno() function in Fox – it tells you which record in the record set you’re on. And the MoveFirst and MoveNext methods, for example, move the record pointer to the first record and the next record in the record set, respectively. The syntax looks like this for a record set named adodcNewCustomers (as opposed to the unhelpful nomenclature of “adodc1”).

```
adodcNewCustomers.Recordset.AbsolutePosition
adodcNewCustomers.Recordset.MoveFirst
adodcNewCustomers.Recordset.MoveNext
```

If you’re thinking this all looks awful familiar, well, right you are. You already *know* all this, don’t you?

Conclusion

In summary, data is data is data. Connecting to it in Visual Basic using ADO is more involved than the USE command you’re used to in Fox, but once you’ve got the connection made, you can manipulate the record set just as you work with a View in Fox. Next month, I’ll show you how to create and manipulate a record set in code – and how to connect it to other controls like list boxes, combo boxes and grids.

<< none this month>
## make this a sidebar

Stupid VB Tricks

Differences between VB and VFP, and inconsistencies in the VB environment that can "gotcha" if you're not careful or observant.

Stupid VB trick #6: