# Advantage Database Server for Visual FoxPro Developers

*Doug Hennig*

*Stonefield Software Inc.*

*Email: dhennig@stonefield.com*

*Web site: http://www.stonefield.com*

*Web site: http://www.stonefieldquery.com*

*Blog: http://doughennig.blogspot.com*

## Overview

Advantage Database Server is a full-featured, high-performance client/server database engine. Interestingly, it can use Visual FoxPro DBF files as its data store and provides a number of benefits over accessing these files directly. This document introduces Advantage and discusses how to access it from VFP applications.

## Introduction

Visual FoxPro is a wonderful development tool. Its rich object-orientation, powerful language features, integrated report writer, and open and extendible interactive development environment (IDE) make it one of the best tools available for developing desktop applications. However, its built-in data engine is both one of its greatest strengths and greatest weaknesses. Strength because the data engine is tightly integrated into VFP and is one of the fastest on the planet and weakness because the DBF file structure can be subject to corruption, lack of security, and size limitations. Fortunately, VFP developers aren't restricted to only using VFP tables as their data store; VFP makes a great front-end to client/server databases such as SQL Server, Oracle, and Sybase.

This document discusses another product in the client/server database market: Advantage Database Server. It first looks at what Advantage Database Server is and what features it has, then delves into how to access Advantage from VFP applications. For the sake of those who are relatively new to client/server technologies, this document assumes you don't have much experience with accessing backend databases and goes into some detail on how to do that.

## Introducing Advantage Database Server

Advantage Database Server, or ADS, is from Sybase iAnywhere, a subsidiary of Sybase. According to their marketing materials, "Advantage Database Server is a full-featured, high performance client/server data management system specifically designed to meet the needs of business application developers." The more you read about ADS, the more you realize that its features align very nicely with those of the database engine in Visual FoxPro. However, it doesn't replace VFP. Like SQL Server, ADS is a database engine rather than a full-featured programming language, and you can easily access its data in VFP using ODBC or ADO. However, as you will see, ADS has better support for VFP than any other database engine, and its latest incarnation, version 9, greatly extends this support.

Here's an overview of the features of ADS compared to VFP:

- It's a true client/server database engine. With file-based engines like VFP, the server containing the data files is just a file server. All processing, such as selecting records, is performed on the workstation, so the entire table must be brought down from the server. With client/server engines, all processing is done on the server, so only the results are sent to the workstation. This provides several benefits, including reduced network traffic and more database management capabilities. In addition, the engine is multi-threaded and supports multiple processors for better scalability.
    - ADS actually comes with two database engines: local and remote. The local engine isn't a true database server but more like VFP in that it uses file-based access to the data. It uses an in-process DLL that loads into the ODBC driver on the client's machine. The remote engine is a true database server that provides all of the benefits of a

client/server architecture. Advantage Local Server is useful for testing on a single development system or as a low-cost database engine (it's actually free) for commercial applications, but has many significant limitations the Advantage Remote Server doesn't have. The benefit of Advantage Local Server is that it gives you a client/server-like mechanism you can scale up to the full remote server if necessary.

- The remote server can be accessed over the Internet if necessary. It supports encrypted and compressed data transmission for security and performance.
- One of the most interesting things about ADS is that it can use either a proprietary file format (files with an ADT extension) or DBF files for data storage. While there are benefits to using ADT files, including additional data types DBFs don't support, using DBFs makes it easier to migrate an existing VFP application to a client/server model. What's really interesting about this is that you can access your existing DBFs through ADS to take advantage of the features ADS provides while still accessing them directly as VFP tables. This makes a very attractive migration strategy: you can modify your application module by module to use client/server techniques while older modules continue to work unchanged.
- When accessing DBF files, it supports two locking mechanisms: compatible and proprietary. Compatible locking, which uses operating system locks on bytes in the DBF files, allows simultaneous access to the data by ADS and non-ADS (such as VFP) applications. Proprietary locking uses an internal locking mechanism. This provides better stability and control but means that the files are opened exclusively by ADS and cannot be access by another application until they are closed in ADS.
- It provides database security. A valid user account is required to connect to the database so unauthorized users cannot access your data. Different user accounts can have different levels of permissions. For example, it's unlikely normal users need to alter, create, or drop tables, so you can prevent everyone but administrative users from performing these database-altering tasks. Even if you're using ADS with DBF files, you can place these files in a folder on the server that normal users don't have access to, so the only access they have to the data is through ADS. This would be a lot more difficult to implement using a purely VFP solution.
- For additional security, ADS can encrypt the tables using a case-sensitive password. Doing so in purely VFP solution requires a third-party product such as Cryptor by Xitech and managing the access to the encrypted data yourself.
- Like VFP, ADS tables can be free or enhanced with a data dictionary (an ADD file). Similar to the VFP database container (DBC), ADD files don't "contain" the tables but instead provide additional information, or meta data, about them. Advantage's data dictionary maps very closely to the VFP DBC, including things such as long field names, primary keys, referential integrity rules, default field values, field validation rules and custom error messages (although ADS only supports minimum and maximum or null

values rather than expressions which can do any type of validation), table validation rules and custom error messages, views, triggers, and stored procedures. As this document discusses later, ADS comes with a utility that generates an ADD from a DBC, automating most of the effort in creating an Advantage data dictionary.

- Although ADS's documentation uses the term "Advantage optimized filters," the description of the technology that provides the high performance querying capabilities of ADS sounds just like the Rushmore technology that gives VFP its speed. ADS examines an index to determine which records match the filter conditions, only accessing the physical records when an index isn't available. The ADS documentation has terms like "fully optimized" and "partially optimized" just like the VFP documentation. This means VFP developers can use their existing knowledge of optimizing VFP queries with ADS databases.
- ADS has a full-text search engine providing very fast searches of memo files. Many VFP developers use third-party products such as PhDbase for full-text searching in their applications, but some of these tools are no longer available or haven't been upgraded to work with the latest versions of VFP.
- Although ADS can access DBF files, it doesn't have the same limits that VFP does. For example, in VFP, DBF and FPT files are limited to 2 GB. In ADS, there isn't a direct limit on the size of the file; instead, the limit is a maximum of 2 billion (2,147,483,648) records. Of course, if your DBF becomes larger than 2 GB, you'll only be able to access it through ADS since VFP will see it as invalid.
- Since the ADS ODBC driver fully supports VFP 9 data types, you can use it in place of the VFP ODBC driver, which hasn't been updated since VFP 6 and so doesn't support new features like Varchar, Varbinary, and Blob fields.
- ADS supports transactions, complete with commit, rollback, and automatic rollback if the workstation or server crashes during the transaction.
- Replication is a process that distributes changes in records in the tables of one database to the tables of another database, such as changes made in databases in remote offices to a single consolidated database at head office or vice versa. Replication with VFP data is certainly possible but you have to write the code yourself, deal with all types of issues such as conflict resolution, and test it extensively to ensure it works under all conditions. ADS has built-in replication features so they've done all the hard work for you.

- ADS includes online backup capability, meaning you can back up your tables while they're open in an application. It isn't possible to do that using normal backup procedures against VFP tables. You can perform full or incremental backups.

## Installing Advantage Database Server

There are several components that make up Advantage: the database server itself, the Advantage Data Architect, the ODBC driver, and the OLE DB provider.

ADS runs on Windows, Netware, and Linux. For Windows, the name of the installer for the database server is ADSWin_x86_32.EXE. Run this program on the server you want ADS installed on. You can, of course, install it on the same system you do development on rather than a separate server, but you would normally install it on an actual server in a production environment. By default, the engine installs into C:\Program Files\Advantage 9.0 (this is also the default for the other components). After installing the engine files, the installer prompts you for the name of the registered owner, whether the engine's Windows service should be started automatically or manually (the default is automatic), which ANSI character set to use (the default is to use the default setting for the machine), and which OEM/localized character set to use. Once you've answered these questions, the Advantage Configuration Utility opens (see **Figure 1**), allowing you to see statistics about the server, including the number of users and connections, and configure certain properties, such as timeout, ports used, and log file locations.

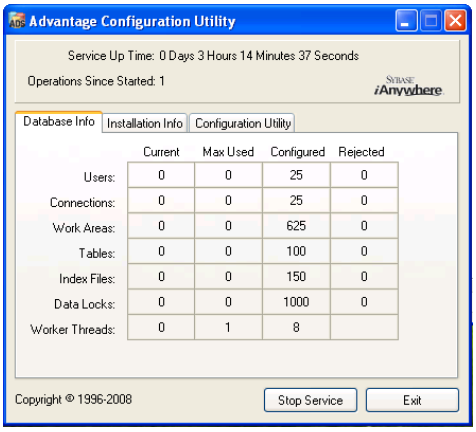



**Figure 1**. After installing ADS, the Configuration Utility appears, allowing you to configure the server properties.

The next thing to install is the Advantage Data Architect, an ADS utility discussed in the next section. Its installer is called Arc32.EXE. Like the server installation, you can specify the install folder name, the ANSI character set, and OEM character set. Next, install the ODBC driver by running ODBC.EXE if you plan on using ODBC to access ADS and install the OLE DB provider by running OLEDB.EXE. You should install the OLE DB provider regardless of whether you plan on using ADO or not because OLEDB.EXE installs a VFP-specific utility discussed later in this document. Both of these prompt for the install folder name, the ANSI character set, and OEM character set.

You're now ready to start using ADS.

## Advantage Data Architect

Advantage Data Architect, also known as ARC, is a connection and database management tool for ADS. If you've used SQL Server Enterprise Manager or Management Studio, or even the VFP Data Explorer, this utility will be somewhat familiar. Interestingly, the complete source code for ARC, which was written in Delphi, is included with the utility. ARC is shown in **Figure 2**.

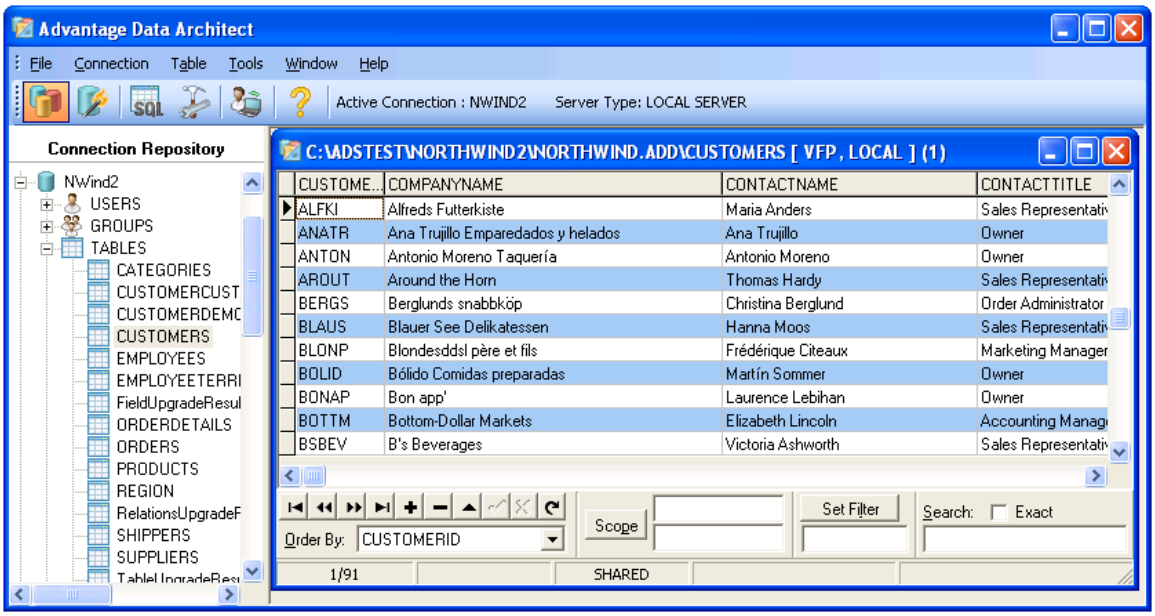



**Figure 2**. Advantage Data Architect provides many of the same features as the VFP Data Explorer or SQL Server Management Studio.

ARC has functions to:

- Create, maintain, and delete databases and tables
    - Browse tables with filtering, searching, sorting, and navigation
    - Import and export data
    - Export table structures as code
    - Manage security settings and user accounts
    - Execute queries in the SQL Utility and Query Builder tools
- Compare data dictionaries

The left pane in ARC is the Connection Repository. This provides easy access to ADS databases you've registered with ARC. (A database doesn't have to be registered with ARC to use it in other applications.) To create a database and add it to the repository, choose Create New Data Dictionary from the File menu; this creates an empty ADD file with the properties you specify in the dialog that appears.

To add an existing database or a directory of free tables to the repository, choose New Connection Wizard from the File menu and follow the steps in the wizard dialog.

I'll discuss various functions in ARC throughout the examples in this document.

## Upsizing a VFP database

Although ADS 9 supports most VFP data features, some of this support relies on using an Advantage database rather than "free" tables. (From an ADS point-of-view, even tables in a VFP DBC are free tables if they aren't included in an ADS database.) This includes support for long field names, primary keys, referential integrity, field and table validation, triggers, and so on; in other words, the same things the VFP database container is used for. In anything but a small database, it would be quite a bit of work to create an Advantage database for an existing VFP database. Fortunately, ADS comes with a utility written in VFP, DBCConvert.PRG, which creates an Advantage database and populates it with information about the tables in a VFP database.

To see how DBCConvert.PRG works, upsize the Northwind sample database that comes with VFP. Start by creating a new folder and copying all the files in the Samples\Northwind folder of the VFP home directory to it; that way, you won't alter your original database when you make some changes described later. Start VFP and run DBCConvert.PRG in the OLEDB subdirectory of the ADS program folder. When prompted for the database, select Northwind.DBC in the folder you copied the files to. After a few seconds, the program completes its tasks. However, note the message displayed: there were 28 errors, but it doesn't indicate what they are. Fortunately, DBCConvert.PRG logs the upsizing process, as discussed later.

Check the directory containing the Northwind database and you'll see some new files:

- Northwind.ADD, AI, and AM: The ADS database.
    - BAK versions of some DBF and FPT files: The upsizing process (not DBCConvert.PRG but ADS itself) backs up these files just in case.
    - FieldUpgradeResults.ADM and ADT, RelationsUpgradeResults.ADM and ADT, TableUpgradeResults.ADM and ADT, and ViewUpgradeResults.ADM and ADT: These ADS tables contain log information about the upsizing process, including any errors that occurred. You'll use these extensively to find and resolve problems in the upsizing process.

Open ARC and choose New Connection Wizard from the File menu or click the New Connection Wizard button in the toolbar. Choose "Create a connection to an existing data dictionary" in the first step of the wizard. In step 2, select the path to Northwind.ADD, the Advantage database created by the upsizing utility, and change Server Type to "remote." Leave the other settings at their default values and click Finish (see **Figure 3**). ARC asks you to login as the AdsSys user (the default administrative user name); since there's no password for that user in this example, click OK.

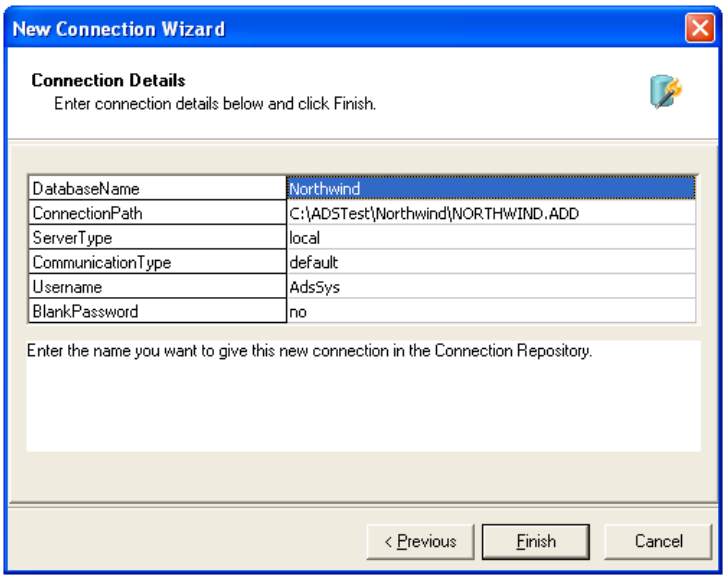



**Figure 3**. Use the New Connection Wizard to create a connection to your upsized VFP database in Advantage Data Architect.

All the tables in Northwind appear under the Tables node but only five of the seventeen views appear under Views. In addition, there are the four new tables mentioned earlier, FieldUpgradeResults, RelationsUpgradeResults, TableUpgradeResults, and ViewUpgradeResults.

Open TableUpgradeResults by double-clicking it. Each upsized table is listed multiple times, once for each operation. The various columns in this table indicate what process was performed in each step. For example, for Customers, there are records for adding the table to the ADS database, specifying long field names, creating indexes, specifying the table validation expression and message, and defining the primary key. Note that one table, Categories, has an error in the step specifying long field names. The error message is (edited for space):

```
The requested operation is not legal for the given field type.   ALTER TABLE CATEGORIES ALTER
"CATEGORYID"  "CATEGORYID" autoinc  NOT NULL ALTER "CATEGORYNA"  "CATEGORYNAME"  char( 15 ) NOT
NULL ALTER "DESCRIPTIO"  "DESCRIPTION"  memo  NULL ALTER "PICTURE"  "PICTURE" blob  NULL
```

Categories contains a General field named Picture but ADS doesn't support General fields so it couldn't process this table. I'll discuss this problem in more detail later.

Open FieldUpgradeResults. This table shows the results of upsizing field comments, validation messages, and default values. Again, one record has an error indicating the upsizing utility couldn't set the comment for Categories.CategoryName; that field doesn't exist in the data dictionary because the error logged in TableUpgradeResults prevented defining the long field names for Categories, so the field is actually named CategoryNa, the 10-character name stored in the DBF header.

RelationsUpgradeResults contains log information for upsized relations. This table contains errors for every record. The error message is (edited for space):

```
The new referential integrity rule could not be added to the Advantage Data Dictionary.  Primary
Key: Visual FoxPro keys used in RI must include a !deleted condition
```

ViewUpgradeResults, which contains log information for upsized views, has lots of errors. In fact, only five views could be upsized. Some of the views failed because they reference Categories.CategoryName, which doesn't exist, but there are several different reasons why others failed.

While upsizing a VFP database takes you a long way to having a complete ADS version of the database, there are a few things to note.

- As indicated by the error message in RelationsUpgradeResults, ADS requires a filter of NOT DELETED() on primary key tags used in referential integrity rules. It's more than a little hassle to add this filter to those tags, because modifying the tag to add the expression deletes persistent relations based on that tag. I've included a program called FixRelations.PRG in the source code accompanying this document that updates all primary keys used in relations to include a NOT DELETED() filter while preserving the relationships.
    - ADS doesn't understand nested JOINs, such as SELECT * FROM Table1 JOIN Table2 JOIN Table3 ON Table2.Field = Table3.Field ON Table1.Field = Table2.Field. You'll need to convert views that use nested JOINs to the more normal sequential JOIN syntax before upsizing them. One way to do that is to open the view in the View Designer and save it again. This works because while in older versions of VFP the View Designer used nested syntax, starting in VFP 8 it uses sequential syntax by default.
    - ADS doesn't support views with ORDER BY clauses or views based on other views.
    - Views using VFP functions won't upsize properly. For example, the Sales_Totals_By_Amount view in the Northwind database uses the VFP BETWEEN() function for one of the WHERE conditions. In that case, changing it to use the SQL BETWEEN clause instead resolves the problem. Other views may not be as easy to fix, however. For example, the Summary_of_Sales_by_Year and Summary_of_Sales_by_Quarter views both use EMPTY() and NVL() in WHERE clauses, so they can't be upsized and must be recreated manually.
    - Tables with General fields won't upsize properly because ADS doesn't support them (yet another reason not to use General fields). That's why the Categories table had an error: Categories.Picture is a General field. In fact, if you try to display the structure of the table in ARC (right-click the table and choose Properties), you'll get an error message and no properties are displayed for that field. You must either remove the Picture field from the table or change it from General to something else, such as Blob.
    - The VFP field-related properties ADS supports are default value, whether nulls are allowed, description (which is upsized from the field comment property), field validation message, and whether codepage translation is performed, so those are all upsized. ADS doesn't support field validation rules, but it does support minimum and

maximum values, so you could manually populate those properties after upsizing is complete.

- Other field properties ADS doesn't support are format, inputmask, field mapping, and field captions, so none of these are upsized. All of these are UI-related properties so they aren't necessary in ADS.
- VFP table-related properties ADS supports are memo block size, table description (upsized from the table comment property), table validation rule, and validation message so those are all upsized. However, rules containing VFP functions are obviously a problem.
- Triggers aren't upsized since they use VFP code which wouldn't be understood by ADS. However, the most common use for triggers is to support referential integrity rules and they are upsized. Note that ADS doesn't support an Ignore RI rule, so those are upsized as Set to NULL. You'll have to recreate triggers used for other purposes.
- Stored procedures aren't upsized for the same reason. You don't have to worry about stored procedures used for RI since RI rules are upsized. However, you'll have to rewrite any other stored procedures in ADS SQL or perhaps move the code into a middle tier component.

- ADS version 9 doesn't support VFP binary indexes, but Sybase plans to support them in version 9.1 or possibly a service pack released before 9.1.

You can fix some of these issues and upsize again. Because DBCConvert.PRG creates an ADS database, that database can't be open in ARC or you'll get an error when you run the PRG again, so close the Northwind connection by right-clicking the connection and choosing Disconnect. Open the Northwind database in VFP and make the following changes:

- Invoices and Product_Sales_For_1997: modify these views, remove the relationships, and recreate them. These views used nested joins so recreating the relationships in VFP 9 converts the views to use sequential join syntax. (Use the View SQL function in the View Designer to confirm that.) While you could do the same for Sales_By_Category, as you'll see later, this view can't be upsized for other reasons.
- Quarterly_Orders and Sales_Totals_By_Amount: modify these views, save, and close without making any changes. These views use the VFP BETWEEN() function for one of the WHERE conditions. Simply saving changes it to use the SQL BETWEEN clause. (Product_Sales_For_1997 also uses BETWEEN() but it was automatically fixed in the previous step when you saved those views. Sales_By_Category also uses BETWEEN() but, as you'll see later, this view can't be upsized for other reasons.)
- Alphabetical_List_of_Products, Current_Product_List, Invoices, and Products_By_Category: modify these views and remove the ORDER BY clause. Unfortunately, you can't do that with Ten_Most_Expensive_Products since it has a TOP

clause and therefore requires an ORDER BY clause. You'll have to recreate that view manually. Also, while you could make this change for Sales_By_Category, Summary_of_Sales_by_Quarter, and Summary_of_Sales_by_Year, it won't help because those views can't be upsized for other reasons.

- Invoices: modify this view, choose View SQL, and change both occurrences of ALLTRIM to TRIM, since ALLTRIM() isn't a supported function in ADS but TRIM() is.
- Run FixRelations.PRG to add a filter of NOT DELETED() to all primary tags involved in persistent relations. (Note: this program has not been extensively tested, so please only run this on a copy of your database!)

- Categories: modify this table and remove the Picture field.

CLOSE TABLES ALL and run DBCConvert.PRG again. This time you get only four errors. Open the Northwind connection in ARC and check the log tables. Categories was upsized properly so now all errors are in views:

- Sales_by_Category can't be upsized because it queries on another view, which isn't supported in ADS.
    - Ten_Most_Expensive_Products has a TOP clause and so requires an ORDER BY clause, which isn't supported in ADS.
- Summary_of_Sales_by_Year and Summary_of_Sales_by_Quarter both use EMPTY() and NVL() in WHERE clauses.

We still have one issue. All the records in the Employees table were deleted! Also, ADS created a new free table, FAIL_EMPLOYEES, which contains all of the deleted records. The reason the records are deleted is that referential integrity is enforced during upsizing. The Employees table has a ReportsTo column that contains the EmployeeID of each person's manager and the ReportsTo value for Andrew Fuller is 0, which doesn't match the EmployeeID of any record. The Insert referential integrity rule for this self-join is set to Ignore, so VFP doesn't raise an error with this record. However, ADS doesn't have an Insert referential integrity rule, just Update and Delete, both of which are set to Restrict. Since Andrew's ReportsTo value is invalid, the RI rule fails and that record is deleted. When that record is deleted, the other records that have Andrew's EmployeeID value in their ReportsTo column are deleted, and so on. As a result, all the records are deleted. You could argue that deleting the record is a bit harsh; perhaps ADS could set ReportsTo to NULL instead. Interestingly, undeleting these records in VFP works.

## Accessing data in VFP or ADS

Upsizing a VFP database doesn't mean it can't be accessed through VFP anymore. It simply means that you now have two ways you can access the database: as native VFP tables or through ADS. You can

modify the data in the tables using VFP or ADS and the other one sees the changes. This includes support for auto-incrementing fields.

For example, after upsizing the Northwind database, open it in ARC and double-click the Employees table to open it. Click the "+" button in the toolbar at the bottom of the table window (see Figure 2) to add a record. Leave EmployeeID blank but fill in the rest of the fields; be sure to fill in a valid value (for example, "2") for the ReportsTo field since the RI rule for the table won't allow a blank or invalid value. Once you've moved off that row, notice the EmployeeID is automatically filled in with the next value.

Now close the table window and disconnect from the database in ARC. Open the database in VFP and open the Employees table. Notice the new record is there. Add a record and notice the next available ID number is automatically filled in for EmployeeID. VFP doesn't require a valid ReportsTo value; you can leave it blank or type something invalid like "99" without an error because the Insert RI rule for the self-join is set to Ignore.

As discussed earlier, being able to access your tables both directly in VFP and through ADS allows you to migrate an existing application to a client/server model one module at a time. For example, in an accounting system, you could modify the Accounts Receivable module to access the data using ADS and deploy that module once the changes are completed and fully tested. The other modules would continue to access the tables directly in VFP. The benefit of this approach is that you don't have to migrate the entire application at once and face the much larger development and testing burden that accompanies such a wholesale change.

If you are starting a new application and have no requirement for backward compatibility of the data, you might consider using ADS native tables (ADT files) rather than DBF files. ADT files have many advantages over DBFs, including no memo file bloat, more data types (such as case-insensitive character fields), longer field names, larger file sizes, more than 255 fields in a table, and automatic reuse of deleted records.

## Full Text Searching

ADS has a fast and powerful full text search (FTS) feature. FTS uses an index on each word in a memo field to provide fast, index-based lookups for desired words. To enable FTS on a table, you have to create an FTS index on one or more memo fields in the table.

If you'd like to test the performance of FTS but don't have a large table with lots of memo content, a program named MakeDemoMemo.PRG included in the source code for this document can help. It goes through your entire hard drive, looking for text files (including PRG, TXT, and HTML) as well as VFP VCX and SCX files and pulls them into the Content memo of a table called DemoMemo.DBF. As you may imagine, this can take some time to run. One test took about ten minutes to create an 81,000 record table with a 545 MB FPT file.

Before adding the table to ADS, a test program looked for all instances of the word "tableupdate" in Content:

```
select * from DemoMemo where atc('tableupdate', Content) > 0 into cursor Temp

select * from DemoMemo where 'tableupdate' $ Content into cursor Temp
```

The first statement, which uses a case-insensitive search, took 305 seconds. The second, which is case-sensitive but faster, took 65 seconds.

Here are the steps to prepare this table for FTS searches:

- Open ARC and choose New Connection Wizard from the File menu. Choose "Create a connection to a directory of existing tables" and click Next. Specify the desired name for the database, select the folder containing DemoMemo.DBF, choose "vfp" for TableType, and click OK. Note that this accesses DemoMemo as a free table rather than through an ADS data dictionary because currently there's a problem accessing FTS indexes on DBF files through a data dictionary. Sybase is aware of the issue and is planning to fix it in a service pack.
    - Right-click the DemoMemo table under the Tables node and choose Properties. Select the Full Text Index Definitions page (see **Figure 4**), click Add Index, and enter the desired name of the index. For Key Field, choose CONTENT (the memo field containing the text to index). You can fine-tune the index by specifying the values of other properties, such as the minimum and maximum word length, "noise" words such as "and" and "the" to exclude from the index, and whether the index is case-sensitive or not. Click OK to create the index.

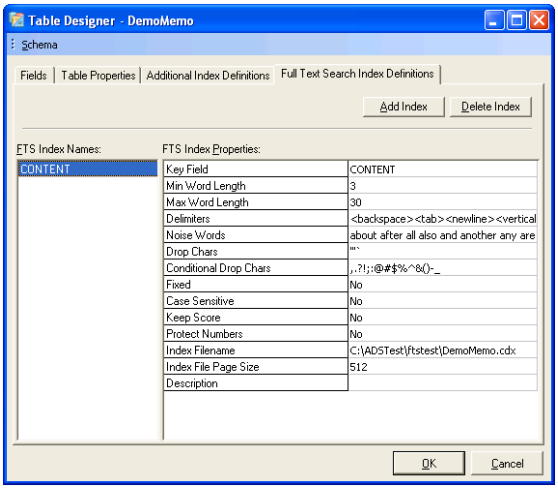



**Figure 4**. The Full Text Search Index Definitions page of the Table Designer dialog in ARC allows you to create indexes that provide fast and powerful full text searching.

It can take some time for ADS to create the FTS index since it creates an index entry for every word in every record. Once you've created the index, though, by default it's automatically updated like other indexes are by record additions, modifications, and deletions. If you wish, you can turn the automatic update off and rebuild the index on demand for better record update performance.

After creating the FTS index, the following statement, executed through the ADS engine, took a mere 0.070 seconds for a case-insensitive search:

```
select * from DemoMemo where contains(Content, 'tableupdate')
```

Full text searches have a lot more power than just searching for the existence of a word, though. For example:

- You can search all fields for a word by specifying "*" as the field name.
- You can search for multiple occurrences of the same word in a given record using the SCORE function. Although not required for this function, turning on the Keep Score property for the index gives better performance since the score value is stored in the index.

```
select * from DemoMemo where contains(Content, 'tableupdate') and
  score(Content, 'tableupdate') > 1
```

- You can look for one word in proximity to another:

```
select * from DemoMemo where contains(Content, 'tableupdate near aerror')
```

One thing to note is that once you've created an FTS index for a table, you can no longer access that table directly in VFP. Trying to do so causes an "operation is invalid for a Memo, Blog, General or Picture field" error because VFP doesn't support indexes on these types of fields.

## Connecting to ADS

There are two ways to access data managed by ADS: using its ODBC driver or its OLE DB provider. (There is actually a third way, using the ADS API functions, but those are low-level functions requiring a lot of coding.) OLE DB requires a connection string while ODBC can use either an ODBC data source name (DSN) or a connection string, sometimes called a DSNless connection.

Here's an example of an OLE DB connection string:

```
provider=Advantage.OLEDB.1;data source=C:\ADSTest\Northwind\Northwind.add;
  User ID=adssys;Password=""
```

An ODBC connection string is similar:

```
driver=Advantage StreamlineSQL ODBC;DataDirectory=C:\ADSTest\Northwind\Northwind.add;
  uid=adssys;pwd=""
```

If you'd rather use an ODBC DSN than a connection string, you can define one using the ODBC Data Source Administrator. Open the Windows Control Panel, open Administrative Tools, and double-click Data Sources (ODBC). Choose the User DSN tab to create a DSN only you can use or System DSN to create a DSN anyone who logs onto your computer can use. Click Add to create a new DSN, select Advantage StreamlineSQL ODBC for the driver, and click Finish. **Figure 5** shows the ADS ODBC driver setup dialog.

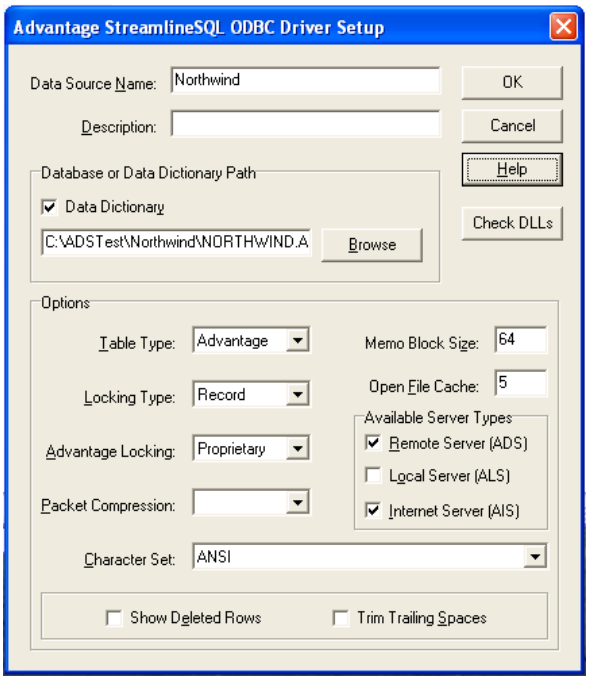



**Figure 5**. You can define a DSN using the ADS ODBC driver.

Specify a name for the data source and fill in the path for the ADS data dictionary. Set the options as desired (the defaults will do for now) and click OK.

## Accessing ADS using remote views

Like a local view, a remote view is simply a pre-defined SQL SELECT statement that's defined in a database container. The difference is that a remote view accesses data via ODBC rather than natively.

You can create a remote view either programmatically using the CREATE SQL VIEW command or visually using the View Designer. In both cases, you need to specify an ODBC connection to use. The connection can either be an ODBC DSN set up on your system or a Connection object that's already defined in the same database. Here's an example that defines a Connection object and creates a remote view to the Customers table of the upsized Northwind database. This was excerpted from code generated by GENDBC; there's actually a lot more code that sets the various properties of the connection, the view, and the fields.

```
CREATE CONNECTION NORTHWINDCONNECTION ;
```

```
   CONNSTRING "DSN=Northwind"
CREATE SQL VIEW "CUSTOMERSVIEW" ;
   REMOTE CONNECT "NorthwindConnection" ;
   AS SELECT * FROM CUSTOMERS Customers
DBSetProp('CUSTOMERSVIEW', 'View', 'UpdateType', 1)
DBSetProp('CUSTOMERSVIEW', 'View', 'WhereType', 3)
DBSetProp('CUSTOMERSVIEW', 'View', 'FetchMemo', .T.)
DBSetProp('CUSTOMERSVIEW', 'View', 'SendUpdates', .T.)
DBSetProp('CUSTOMERSVIEW', 'View', 'Tables', 'CUSTOMERS')
DBSetProp('CUSTOMERSVIEW.customerid', 'Field', 'KeyField', .T.)
DBSetProp('CUSTOMERSVIEW.customerid', 'Field', 'Updatable', .F.)
DBSetProp('CUSTOMERSVIEW.customerid', 'Field', 'UpdateName', 'CUSTOMERS.CUSTOMERID')
DBSetProp('CUSTOMERSVIEW.companyname', 'Field', 'Updatable', .T.)
DBSetProp('CUSTOMERSVIEW.companyname', 'Field', 'UpdateName', 'CUSTOMERS.COMPANYNAME')
```

One of the easiest ways you can upsize an existing application is using the ADS upsizing utility to create an ADS database from your VFP database, then create a new VFP database (for example, REMOTE.DBC) and create remote views in that database with the same names as the tables they're based on. That way, the code to open a remote view is exactly the same as that to open a local table except you'll open a different database first. For example, if you have an application object named oApp and it has an lUseLocalData property to indicate whether local or remote data is used, this code opens the appropriate database and then open either the Customers table or the Customers remote view:

```
if oApp.lUseLocalData
  open database Local
else
  open database Remote
endif
use Customers
```

If you're using cursor objects in the DataEnvironment of forms and reports, you have a little bit of extra work to do because those objects have a reference to the specific database you selected when you dropped the views into the DataEnvironment. To handle this, put code similar to the following into the BeforeOpenTables method of the DataEnvironment:

```
local loObject
for each loObject in This.Objects
```

```
  if upper(loObject.BaseClass) = 'CURSOR' and not empty(loObject.Database)
    loObject.Database = iif(oApp.lUseLocalData, 'local.dbc', 'remote.dbc')
  endif
next
```

### *Advantages*

The advantages of remote views are:

- You can use the View Designer to create a remote view visually. It's great to visually see all the fields in the underlying tables, easily set up the various parts of the SQL SELECT statement using a friendly interface, and quickly set properties of the view using checkboxes or other UI elements.
    - From a language point-of-view, remote views act just like tables. As a result, they can be used anywhere: you can USE them, add them to the DataEnvironment of a form or report, bind them to a grid, process them in a SCAN loop, and so forth.
    - It's easier to convert an existing application to use remote views, especially if it already uses local views, than using other techniques discussed later.
    - Because you can add a remote view to the DataEnvironment of a form or report, you can take advantage of the visual support the DE provides: dragging and dropping fields or the entire cursor to automatically create controls, easily binding a control to a field by selecting it from a combobox in the Properties Window, and so on. Also, depending on the settings of the AutoOpenTables and OpenViews properties, VFP automatically opens the remote views for you.
    - It's easy to update the backend with changes: assuming the properties of the view have been set up properly, you simply call TABLEUPDATE(). Transaction processing and update conflict detection are built-in.
- Remote views are easy to use in a development environment: just USE and then BROWSE.

### *Disadvantages*

The disadvantages of remote views are:

- Remote views live in a DBC, so that's one more set of files you have to maintain and install on the client's system.
    - Since a remote view's SQL SELECT statement is pre-defined, you can't change it on the fly. Although this is fine for a typical data entry form, it can be an issue for queries and reports. You may have to create several views from the same set of data, each varying in the fields selected, structure of the WHERE clause, and so forth.

- You can't call a stored procedure from a remote view, so a remote view needs direct access to the underlying tables.
- When you use TABLEUPDATE() to write changes in the view to the backend database, you have little ability (other than by setting a few properties) to control how VFP does the update.
- As is the case with local views, if you use a SELECT * view to retrieve all the fields from a specific table and the structure of that table on the backend changes, the view is invalid and must be recreated.
- When you open a view, VFP attempts to lock the view's records in the DBC, even if only briefly. This can cause contention in busy applications where several users might try to open a form at the same time. Although there are workarounds (copying the DBC to the local workstation and using that one or, in VFP 7 and later, using SET REPROCESS SYSTEM to increase the timeout for lock contention), it's something you must plan for.
- Until VFP 8, which allows you to specify the connection handle to use when you open a remote view with the USE statement, you had little ability to manage the connections used by your application.

- The connection information used for a remote view is hard-coded in plain text in the DBC. That means that a hacker can easily discover the keys to your backend kingdom (such as the user name and password) using nothing more complicated than Notepad to open the DBC. This isn't much of an issue starting in VFP 7 because it allows you to specify a connection string when you open a remote view with the USE command, meaning that you can dynamically assemble the database path, user name, and password, likely from encrypted information, just before opening the view.

Basically, it comes down to a control issue: remote views make it easy to work with backend data, but at the expense of limiting the control you have over them.

## Accessing ADS using SQL passthrough

VFP provides a number of functions, sometimes referred to as SQL passthrough (or SPT) functions, which allow you to access a backend database. SQLCONNECT() and SQLSTRINGCONNECT() make a connection to the backend database engine. The difference between these two functions is that SQLCONNECT() requires an existing ODBC DSN while SQLSTRINGCONNECT() uses a connection string. SQLDISCONNECT() disconnects from the backend. SQLEXEC() sends a command, such as a SQL SELECT statement, to the database engine, typically (but not necessarily, depending on the command) putting the returned results into a VFP cursor.

Here's an example that connects to the upsized Northwind database, retrieves all customers, and disconnects. (This assumes there's a DSN called "Northwind" that defines how to connect to this database.)

```
lnHandle = sqlconnect('Northwind')
sqlexec(lnHandle, 'select * from Customers')
browse
sqldisconnect(lnHandle)
```

To use a DSNless connection instead, replace the SQLCONNECT() statement with the following:

```
lcConnString = 'driver=Advantage StreamlineSQL ODBC;' + ;
  'DataDirectory=C:\ADSTest\Northwind\Northwind.add;'
lnHandle = sqlstringconnect(lcConnString)
```

**Table 1** lists the keywords you can specify in a connection string; most of these have equivalents in the ODBC dialog shown in Figure 5. All but DataDirectory are optional.

**Table 1**. ADS ODBC driver connection string keywords.

| Keyword | Description |
|---|---|
| DataDirectory | Specify the path and name of the ADD file to use an ADS database. For free tables, specify the directory for the tables. |
| DefaultType | For free tables, specify "FoxPro" for VFP tables or "Advantage" for ADS tables (ADT files). This setting is ignored for databases. |
| ServerTypes | Specify a numeric value that's the sum of the types of ADS server to connect to: Remote (2), Local (1), or Internet (4). For example, use 3 (2 + 1) for Remote and Local. |
| AdvantageLocking | "ON" (the default) to use ADS proprietary locking or "OFF" for VFP-compatible locking. |
| Locking | "Record" (the default) for record locking or "File" to lock the entire file during updates. |
| Rows | Similar to SET DELETED. "True" displays deleted records while "False" (the default) omits them. This setting is ignored for ADS tables because deleted records are never visible in that case. |
| TrimTrailingSpaces | Similar to Varchar fields. "True" removes trailing spaces from character fields returned to the application and "False" (the default) does not. |
| MemoBlockSize | Similar to the SET BLOCKSIZE command. Specify the block size for memo fields for new tables. The default is 64 for VFP tables and 8 for ADS tables. |
| CharSet | The collation setting to use: "ANSI" (the default) or "OEM". If you use "OEM", you must also specify the Language setting. |
| Language | The language to use if CharSet=OEM. |
| MaxTableCloseCache | The number of cursors in the ADS cache; the default is 5. |
| Compression | The type of compression to use. See the ADS help file for a discussion of the types of compression available. |
| CommType | The communication protocol to use. See the ADS help file for details. |

Regardless of whether you use a DSN or a connection string to connect to ADS, you then use the same type of SQL statements you'd use to access, update or delete records in VFP tables, but you use the SQLEXEC() function to execute them. In addition to DML (Data Manipulation Language) functions like SELECT, INSERT, UPDATE, and DELETE, the ODBC driver also supports DDL (Data Definition Language) functions such as CREATE DATABASE | TABLE | INDEX | VIEW | PROCEDURE, DROP INDEX | TABLE | VIEW | PROCEDURE, and ALTER TABLE.

Note that while ADS supports most of the VFP data types, how you specify values for Logical, Date, and DateTime fields is a little different than with VFP syntax. ADS Logical values come back to VFP as Logical fields but you must specify them using .T., True or 1 for true and .F., False or 0 for false. For example, the first of the following statements fails but the rest succeed:

```
sqlexec(lnHandle, "select * from Products where Discontinued")

sqlexec(lnHandle, "select * from Products where Discontinued=.T.")

sqlexec(lnHandle, "select * from Products where Discontinued=True")

sqlexec(lnHandle, "select * from Products where Discontinued=1")
```

Date and DateTime values must be specified using standard ODBC syntax: {d 'YYYY-MM-DD'} for Date and {ts 'YYYY-MM-DD HH:MM:SS'} for DateTime. Here's an example:

```
sqlexec(lnHandle, "select * from orders " + ;
  "where OrderDate between {d '1997-07-01'} and {d '1997-07-31'}")
```

Here's a function called VFP2ODBCDate that converts VFP Date and DateTime values to ODBC syntax:

```
lparameters tuDate
local lcDate, ;
  lcReturn
lcDate = transform(year(tuDate)) + ;
  '-' + padl(month(tuDate), 2, '0') + ;
  '-' + padl(day(tuDate), 2, '0')
if vartype(tuDate) = 'D'
  lcReturn = "{d '" + lcDate + "'}"
else
  lcReturn = "{t '" + lcDate + ;
    ' ' + padl(hour(tuDate), 2, '0') + ;
    ':' + padl(minute(tuDate), 2, '0') + ;
    ':' + padl(sec(tuDate), 2, '0') + "'}"
endif vartype(tuDate) = 'D'
return lcReturn
```

The following example uses this function:

```
ldFrom = {^1997-07-01}
ldTo   = {^1997-07-31}
sqlexec(lnHandle, "select * from orders " + ;
  "where OrderDate between " + VFP2ODBCDate(ldFrom) + " and " + VFP2ODBCDate(ldTo))
```

Instead of converting VFP values into ODBC syntax, you can use a parameterized query, replacing a hard-coded value with a variable name prefixed with "?" (the variable, of course, must be in scope). In that case, VFP takes care of data type conversions for you. This has the additional benefit of avoiding SQL injection attacks (which are beyond the scope of this document). For example:

```
ldFrom = {^1997-07-01}
ldTo   = {^1997-07-31}
sqlexec(lnHandle, 'select * from orders ' + ;
  'where OrderDate between ?ldFrom and ?ldTo')
```

Although VFP can use single quotes, double quotes, and square brackets as string delimiters, pass string values to SPT functions delimited with single quotes only.

Although these examples omit it, be sure to check the return value of SQLEXEC(). If it returns something less than 1, the command failed so use AERROR() to determine what went wrong. Also, you can specify the name of the cursor to create as the third parameter to SQLEXEC(); the cursor is named SQLResult if you omit this parameter.

### *Advantages*

The advantages of using SPT are:

- You have a lot more flexibility in data access than with remote views, such as calling stored procedures using the SQLEXEC() function.
    - You can change the connection information on the fly as needed. For example, you can store the user name and password as encrypted values and only decrypt them just before using them in the SQLCONNECT() or SQLSTRINGCONNECT() functions. As mentioned earlier, this isn't nearly the advantage over remote views that it used to be, since VFP 7 and later allows you to specify the connection string on the USE command.
    - You can change the SQL SELECT statement as needed. For example, you can easily vary the list of the fields, the WHERE clause (such as changing which fields are involved or eliminating it altogether), the tables, and so on.
    - You don't need a DBC to use SPT, so there's nothing to maintain or install, lock contention isn't an issue, and you don't have to worry about a SELECT * statement being made invalid when the structure of the backend tables change.
    - As with remote views, the result set of a SPT call is a VFP cursor, which can be used anywhere in VFP.
    - Although you have to code for it yourself (this is discussed in more detail under Disadvantages), you have greater control over how updates are done. For example, you might use a SQL SELECT statement to create the cursor but call a stored procedure to update the ADS tables.

- You can manage your own connections. For example, you might want to use a connection manager object to manage all the connections used by your application in one place.

### *Disadvantages*

The disadvantages of using SPT are:

- It's more work, since you have to code everything: creating and closing the connection, the SQL SELECT statements to execute, and so on. You don't have a nice visual tool like the View Designer to show you which fields exist in which tables.
    - You can't visually add a cursor created by SPT to the DataEnvironment of a form or report. Instead, you have to code the opening of the cursors (for example, in the BeforeOpenTables method), you have to manually create the controls, and you have to fill in the binding properties (such as ControlSource) by typing them yourself. Don't make a typo when you enter the alias and field names or the form won't work.
    - They're harder to use than remote views in a development environment: instead of just issuing a USE command, you have to create a connection, then use a SQLEXEC() call to get the data you want to look at. You can make things easier on yourself if you create a set of PRGs to do the work for you or you can use the Data Explorer that comes with VFP to examine the structures and contents of the tables. You can even create a DBC and set of remote views used only in the development environment as a quick way to look at the data.
    - Cursors created with SPT can be updatable, but you have to make them so yourself using a series of CURSORSETPROP() calls to set the SendUpdates, Tables, KeyFieldList, UpdatableFieldList, and UpdateNameList properties. Also, you have to manage transaction processing and update conflict detection yourself.
- Since SPT cursors aren't defined like remote views, you can't easily switch between local and remote data using SPT as you can with remote views by simply changing which view you open in a form or report.

## Accessing ADS using ADO

OLE DB providers are similar to ODBC drivers: they provide a standard, consistent way to access data sources. Because OLE DB is a set of low-level COM interfaces, it's not easy to work with in languages like VFP. To overcome this, Microsoft created ActiveX Data Objects (ADO), a set of COM objects that provide an object-oriented front-end to OLE DB.

ADO consists of several objects, including:

- Connection: This is the object responsible for communicating with the data source.

- Recordset: This is the equivalent of a VFP cursor: it has a defined structure, contains the data in the data set, and provides properties and methods to add, remove, or update records, move from one to another, filter or sort the data, and update the data source.

- Command: This object provides the means of doing more advanced queries than a simple SELECT statement, such as parameterized queries and calling stored procedures.

Here's an example (ADOExample.PRG) that gets all Brazilian customers from the upsized Northwind database and displays the customer ID and company name. Notice that the Connection object handles the connection while the Recordset handles the data.

```
local loConn as ADODB.Connection, ;
  loRS as ADODB.Recordset, ;
  lcCustomers

* Connect to the ADS database.

loConn = createobject('ADODB.Connection')
loConn.ConnectionString = 'provider=Advantage.OLEDB.1;' + ;
  'data source=c:\adstest\northwind\northwind.add'
loConn.Open()

* Create a Recordset and set its properties.

loRS = createobject('ADODB.Recordset')
loRS.ActiveConnection = loConn
loRS.LockType         = 3  && adLockOptimistic
loRS.CursorLocation   = 3  && adUseClient
loRS.CursorType       = 3  && adOpenStatic

* Execute a query and display the results.

loRS.Open("select * from customers where country='Brazil'")
lcCustomers = ''
do while not loRS.EOF
  lcCustomers = lcCustomers + ;
```

```
    loRS.Fields('customerid').Value + chr(9) + ;
    loRS.Fields('companyname').Value + chr(13)
  loRS.MoveNext()
enddo while not loRS.EOF
messagebox(lcCustomers)
loRS.Close()
loConn.Close()
```

Notice how this code uses object-oriented code to access the Recordset. The EOF property is the equivalent of the VFP EOF() function and the MoveNext method is like SKIP. To access the value of a field in the current record, use Recordset.Fields('FieldName').Value.

Using parameterized queries with ADO is a little more work than it is with ODBC. In addition to specifying a parameter as "?" (without the variable name), you also have to use ADO Command and Parameter objects to specify the parameter and its value. This code references ADOVFP.H, an include file of constants useful when working with ADO.

```
#include ADOVFP.H

* Connect to the ADS database.

loConn = createobject('ADODB.Connection')
loConn.ConnectionString = 'provider=Advantage.OLEDB.1;' + ;
  'data source=c:\adstest\northwind\northwind.add'
loConn.Open()

* Create a Command object and define the command type and connection.

loCommand = createobject('ADODB.Command')
loCommand.CommandType      = adCmdText
loCommand.ActiveConnection = loConn

* Create a Parameter object, set its properties, and add it to the Command
* object.

loParameter = loCommand.CreateParameter('Country', adChar, adParamInput, 15)
```

```
loParameter.Value = 'UK'

loCommand.Parameters.Append(loParameter)


* Execute a parameterized query and display the results.


loCommand.CommandText = 'select * from customers where country = ?'

loRS = loCommand.Execute()

* same code as above to display the results
```

### *Advantages*

The advantages of using ADO are:

- Many of the advantages are the same as with SPT: you have more flexibility in data access than with remote views, you can change the connection information on the fly as needed, you can change the SQL SELECT statement as needed, you can manage your own connections, and there's no DBC involved.
    - Although performance differences aren't significant in simple scenarios (in fact, in general, ODBC is faster than ADO), ADO is more scalable in heavily-used applications such as Web servers.
    - ADO is object-oriented, so you can deal with the data like objects.
    - Depending on how they're set up, ADO Recordsets are automatically updateable without any additional work other than calling the Update or UpdateBatch methods. Transaction processing and update conflict detection are built-in.
- You can easily persist a Recordset to a local file, then later reload it and carry on working, and finally update the ADS data source. This makes it a much better choice for "road warrior" applications than remote views or SPT.

### *Disadvantages*

The disadvantages of ADO are:

- It's more work, since you have to code everything: creating and closing the connection, the SQL SELECT statements to execute, and so on. You don't have a nice visual tool like the View Designer to show you which fields exist in which tables on the backend.
    - An ADO Recordset is not a VFP cursor, so you can't use it in places that require a cursor, such as grids and reports. There are functions in the VFPCOM utility (available for download from the VFP home page, http://msdn.microsoft.com/vfoxpro) that can convert a Recordset to a cursor and vice versa, but using them can impact performance,

especially with large data sets, and they have known issues with certain data types. If you want to use ADO, CursorAdapter (discussed next) is the way to go.

- There's no visual support for ADO Recordsets, so you have to code their creation and opening, you have to manually create the controls, and you have to fill in the binding properties (such as ControlSource) by typing them yourself. This is even more work than for SPT, because the syntax isn't just CURSOR.FIELD—it's Recordset.Fields('FieldName').Value.
- They're the hardest of the technologies to use in a development environment, since you have to code everything: making a connection, retrieving the data, and moving back and forth between records. You can't even BROWSE to see visually what the result set looks like (unless you use VFPCOM or CursorAdapter to convert the Recordset to a cursor).

- There's a bigger learning curve involved with ADO than using the cursors created by ODBC.

## Accessing ADS using CursorAdapter

One of the things you've likely noted is that each of the mechanisms discussed is totally different from the others. That means you have a new learning curve with each one, and converting an existing application from one mechanism to another is a non-trivial task.

Fortunately, there's a VFP technology that provides a common interface for both ODBC and OLE DB: the CursorAdapter class. CursorAdapter, added in VFP 8, is a great solution because:

- It makes it easy to use ODBC, ADO, or XML, even if you're not very familiar with these technologies.
- It provides a consistent interface to remote data regardless of the mechanism you choose.
- It makes it easy to switch from one mechanism to another.

Here's an example of the last point. Suppose you have an application that uses ODBC with CursorAdapters to access ADS data, and for some reason you want to change to use ADO instead. All you need to do is change the DataSourceType of the CursorAdapters and change the connection to the ADS database, and you're done. The rest of the components in the application neither know nor care about this; they still see the same cursor regardless of the mechanism used to access the data.

Here's an example (CursorAdapterExample.PRG) that gets certain fields for Brazilian customers from the Customers table in the Northwind database. The cursor is updateable, so if you make changes in the browse window, close it, and then run the program again, you'll see that your changes were saved.

```
local lcConnString, ;
  lnHandle, ;
```

```
  loCursor as CursorAdapter, ;
  laErrors[1]
close tables all

* Connect to ADS.

lcConnString = 'driver=Advantage StreamlineSQL ODBC;' + ;
  'DataDirectory=C:\ADSTest\Northwind\Northwind.add;'
lnHandle     = sqlstringconnect(lcConnString)

* Create a CursorAdapter and set its properties.

loCursor = createobject('CursorAdapter')
with loCursor
  .Alias              = 'Customers'
  .DataSourceType     = 'ODBC'
  .DataSource         = lnHandle
  .SelectCmd          = "select CUSTOMERID, COMPANYNAME, CONTACTNAME " + ;
    "from CUSTOMERS where COUNTRY = 'Brazil'"
  .KeyFieldList       = 'CUSTOMERID'
  .Tables             = 'CUSTOMERS'
  .UpdatableFieldList = 'CUSTOMERID, COMPANYNAME, CONTACTNAME'
  .UpdateNameList     = 'CUSTOMERID CUSTOMERS.CUSTOMERID, ' + ;
    'COMPANYNAME CUSTOMERS.COMPANYNAME, CONTACTNAME CUSTOMERS.CONTACTNAME'
  if .CursorFill()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif .CursorFill()
endwith
```

You don't have to create a CursorAdapter programmatically. You can use the Class Designer to create a CursorAdapter subclass and either fill in the properties in the Properties window or use the

CursorAdapter Builder, which provides a nice visual tool for the CursorAdapter (see **Figure 6**). Note that the CursorAdapter Builder doesn't quite work right with ADS when you're using ODBC to connect to the database; see **Appendix A**, "Fixing the VFP CursorAdapter Builder," for details and how to correct the problem.

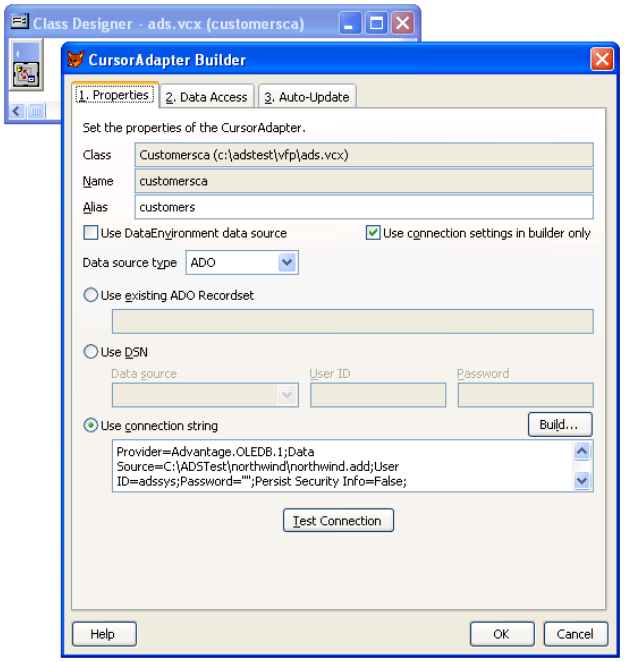



**Figure 6**. The CursorAdapter Builder provides a visual tool to create CursorAdapter subclasses.

Although CursorAdapter has quite a few properties and methods, the most important ones are:

- DataSourceType: specifies how to access the data. The choices are "Native," "XML," "ODBC," and "ADO," although only the latter two are used with ADS.
    - DataSource: the value of this property depends on what DataSourceType is set to. In the case of ODBC, it must be an open ODBC connection handle. For ADO, it's an ADO Recordset object with its ActiveConnection set to an open Connection object. Note that in either case, you're responsible for opening and managing the connection yourself.
    - SelectCmd: the SQL statement to execute to retrieve the data.
    - Alias: the alias of the cursor created.
    - KeyFieldList, Tables, UpdatableFieldList, and UpdateNameList: these fields are the key to making the cursor updateable. Set them to the appropriate values and CursorAdapter automatically writes changes back to the source data. See the VFP documentation for details on these properties.
    - CursorFill: call this method to create the cursor and execute the statement in SelectCmd to populate the cursor.

- CursorDetach: by default, the cursor created by CursorAdapter is "attached" to the CursorAdapter object. When the CursorAdapter is destroyed (such as when it goes out of scope), the cursor is automatically closed. If you want the cursor to remain open, call the CursorDetach method to detach the cursor from the CursorAdapter.

A CursorAdapter can use either ODBC or ADO to connect to ADS. For ODBC, open a connection to the database using SQLCONNECT() or SQLSTRINGCONNECT() and set the CursorAdapter DataSource property to the connection handle and set the DataSourceType property to "ODBC." ADO is a little more work: instantiate and open an ADO Connection object, instantiate a Recordset object, set the Recordset's ActiveConnection property to the Connection object, set the CursorAdapter's DataSource property to the Recordset object, and set DataSourceType to "ADO." For parameterized queries, use the "?VariableName" syntax in your SQL statement, even for ADO. For ADO, though, you must also instantiate an ADO Command object and pass it as the fourth parameter to CursorFill (don't worry about the Parameter object; VFP takes care of that internally).

Instead of doing all that work for ADO manually, the SFCursorAdapterADO subclass included in SFCursorAdapter.VCX in the source code for this document does some of this work for you. Its DataSourceType property is set to "ADO" and its Init method sets up the DataSource property.

```
local loRS as ADODB.Recordset
loRS = createobject('ADODB.RecordSet')
loRS.CursorLocation = 3  && adUseClient
loRS.LockType       = 3  && adLockOptimistic
This.DataSource     = loRS
```

After creating and opening a Connection object, pass it to SetConnection.

```
lparameters toConnection
This.DataSource.ActiveConnection = toConnection
```

You don't have to pass a Command object to CursorFill; SFCursorAdapterADO automatically uses a Command object if there's a "?" in the SQL statement.

```
lparameters tlUseCursorSchema, ;
  tlNoData, ;
  tnOptions, ;
  toSource
local loSource as ADODB.Command, ;
  lnOptions, ;
  llUseCursorSchema, ;
```

```
  llNoData, ;
  llReturn, ;
  laError[1]

* If we have a parameterized query, we need an ADO Command object. Create one
* if it wasn't passed.

if '?' $ This.SelectCmd and vartype(toSource) <> 'O'
  loSource = createobject('ADODB.Command')
  loSource.ActiveConnection = This.DataSource.ActiveConnection
  lnOptions = adCmdText
else
  loSource  = toSource
  lnOptions = tnOptions
endif '?' $ This.SelectCmd ...

* If the first two parameters weren't specified, we don't want to explicitly
* pass .F., so use the default values. If CursorSchema is empty, we'll, of
* course, pass .F. for the first parameter.

do case
  case pcount() >= 2
    llUseCursorSchema = tlUseCursorSchema
    llNoData          = tlNoData
  case pcount() = 1
    llUseCursorSchema = tlUseCursorSchema
    llNoData          = This.NoData
  case pcount() = 0
    llUseCursorSchema = This.UseCursorSchema
    llNoData          = This.NoData
endcase
if empty(This.CursorSchema)
  llUseCursorSchema = .F.
```

```
endif empty(This.CursorSchema)
llReturn = dodefault(llUseCursorSchema, llNoData, lnOptions, loSource) and ;
  used(This.Alias)

* If something went wrong, find out why.

if not llReturn
  aerror(laError)
  This.cErrorMessage = laError[2]
endif not llReturn
return llReturn
```

Here's an example that uses SFCursorAdapterADO, taken from ADOCursorAdapterExample.PRG:

```
local loConnection as ADODB.Connection, ;
  loCA as SFCursorAdapterADO of SFCursorAdapter.vcx
private pcCountry

* Create and open an ADO Connection object.

loConnection = createobject('ADODB.Connection')
loConnection.ConnectionString = 'Provider=Advantage.OLEDB.1;' + ;
  'Data Source=C:\ADSTest\northwind\northwind.add;' + ;
  'User ID=adssys;Password=""'
loConnection.Open()

* Create an SFCursorAdapterADO object and set it up.

loCA = newobject('SFCursorAdapterADO', 'SFCursorAdapter.vcx')
with loCA
  .SetConnection(loConnection)
  .SelectCmd = 'select * from customers where country = ?pcCountry'
  .Alias     = 'customers'
```

```
* Do the query and either show the result set or an error.


  pcCountry = 'Germany'
  if .CursorFill()
    browse
  else
    messagebox(loCa.cErrorMessage)
  endif .CursorFill()
endwith


* Close the connection.


loConnection.Close()
```

### *Advantages*

The advantages of CursorAdapters are essentially the combination of those of all of the other technologies.

- Depending on how it's set up (if it's completely self-contained, for example), opening a cursor from a CursorAdapter subclass can almost be as easy as opening a remote view: you simply instantiate the subclass and call the CursorFill method. You could even call that from Init to make it a single-step operation.
    - It's easier to convert an existing application to use CursorAdapters than to use cursors created with SPT.
    - Like remote views, you can add a CursorAdapter to the DataEnvironment of a form or report and take advantage of the visual support the DE provides: dragging and dropping fields to automatically create controls, easily binding a control to a field by selecting it from a combobox in the Properties Window, and so on.
    - It's easy to update the backend with changes: assuming the properties of the view have been set up properly, you simply call TABLEUPDATE().
    - Because the result set created by a CursorAdapter is a VFP cursor, they can be used anywhere in VFP: in a grid, a report, processed in a SCAN loop, and so forth. This is true even if the data source comes from ADO and XML, because the CursorAdapter automatically takes care of conversion to and from a cursor for you.
    - You have a lot of flexibility in data access, such as calling stored procedures or middle-tier objects.

    - You can change the connection information on the fly as needed, you can change the SQL SELECT statement as needed, you don't need a DBC, and you can manage your own connections.
- Although you have to code for it yourself, you have greater control over how updates are done. For example, you might use a SQL SELECT statement to create the cursor but call a stored procedure to update the backend tables.

## Disadvantages

There aren't a lot of disadvantages for CursorAdapters:

- You can't use a nice visual tool like the View Designer to create CursorAdapters, although the CursorAdapter Builder is a close second.
- Like all new technologies, there's a learning curve that must be mastered.

## Licensing

Advantage Database Server uses a concurrent licensing model; you need one license per connected user. Each workstation can have an unlimited number of database connections. Multiple Advantage-enabled applications running on a single workstation are licensed as a single user. Sybase iAnywhere does not publish a price list for licenses. Although they state that they have flexible pricing options and OEM partner discounts and that their price is competitive to other database servers, you must contact them for a quote based on how many licenses you require. However, I have seen published in one place that a five-user license is $645 and an unlimited user license is $7,870. I have not verified the accuracy of this.

## Resources

The Advantage Developer Zone site, http://devzone.advantagedatabase.com, has numerous resources for learning more about ADS, such as newsgroups (including a VFP-specific newsgroup), online documentation, white papers, tutorials, and sample code. Also, a book by Cary Jensen and Loy Anderson, *Advantage Database Server: A Developer's Guide* (ISBN 978-1-4259-7726-9), provides a great introduction to ADS. Although none of the examples are in VFP, VFP developers will have no trouble understanding and translating the code.

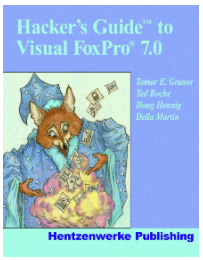
Andrew MacNeill interviewed J.D. Mullin, R & D Manager for ADS, in The FoxShow #49, a podcast available for download at http://akselsoft.libsyn.com/index.php?post_id=302994. This interview provides some background to ADS and VFP and discusses some of the design features of ADS. J.D. has a blog (http://jdmullin.blogspot.com) and has posted some videos, including one on using the Advantage SQL Debugger (http://devzone.advantagedatabase.com/jeremym/sqldebug/sqldebug.html).

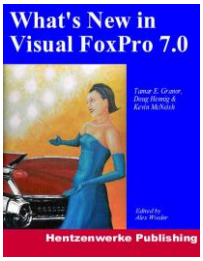
## Summary

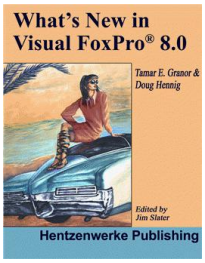
Advantage Database Server is an exciting database engine that provides better support for VFP application developers than any other client/server database engine. It can form the basis of a migration strategy to move your applications from file-based data access to true client/server technology.

## Biography

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna. Doug is co-author of the "What's New in Visual FoxPro" series, "The Hacker's Guide to Visual FoxPro 7.0," and the soon-to-be-released "Making Sense of Sedna and VFP 9 SP2." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). Doug wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor and Advisor Guide. He currently writes for FoxRockX (http://www.foxrockx.com). He spoke at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://www.codeplex.com/VFPX). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://fox.wikis.com/wc.dll?Wiki~FoxProCommunityLifetimeAchievementAward).







# Appendix A. Fixing the VFP CursorAdapter Builder

The CursorAdapter Builder has a problem with the ADS ODBC driver (it works fine with the ADS OLE DB provider). The Select Command Builder dialog displays when you click the Build button for the Select command in page 2 of the CursorAdapter Builder. This dialog gives an error with the ADS ODBC driver because the driver returns a different result set to the SQLTABLES() and SQLCOLUMNS() functions than SQL Server and many other drivers do. Rather than having fields named TABLE_NAME and COLUMN_NAME, ADS names them TABLENAME and COLUMNNAME.

Fortunately, Microsoft provides the source code for the CursorAdapter Builder and the fix was easy, so the source code accompanying this document includes a replacement DEBuilder.APP that takes care of these issues. Copy that file to the Wizards folder of the VFP home directory, overwriting the existing file. The source code for the fix is in the included DECABuilder.VCX, which you normally find in the Tools\XSource\VFPSource\Wizards\DEBuilder folder of the VFP home directory after extracting XSource.ZIP in Tools\XSource.