Much ADO About Something

Doug Hennig

The release of the OLE DB provider for VFP means better performance and scalability for applications that need to access VFP data via ADO. However, there are some interesting issues and limitations of this provider you need to be aware of.

We've been able to access VFP data using ADO since ADO's release. However, we had to do it using the OLE DB provider for ODBC, which meant an extra layer in the data access mechanism, causing performance issues under some conditions. Now, VFP 7 includes a native OLE DB provider for VFP (in VFPOLEDB.DLL, installed by default in C:\Program Files\Common Files\System\OLE DB). It has several benefits over the VFP ODBC driver:

- It has a future. Microsoft has announced that the VFP ODBC driver is in "support" mode, meaning no future enhancements will be made to it. That means no support for new features in VFP 7, so the ODBC driver can't access tables belonging to a database container (DBC) that has the new database events feature turned on.
- In addition to better performance, it uses an improved threading model over the VFP ODBC driver for better scalability.
- It provides support for VFP-specific features such as creating rules, triggers, default values, or stored procedures using ADOX.

This article isn't intended to be a primer on using ADO; for that, see John Petersen's ADO white paper in MSDN (http://msdn.microsoft.com/library/default.asp?URL=/library/techart/ADOJump.htm). Instead, we'll look specifically at the VFP OLE DB provider, how it's used, and some issues to be aware of.

Using the OLE DB provider

The OLE DB provider for VFP is very similar to the providers for other database engines. Typically, you instantiate an ADO Connection object, set its ConnectionString property to specify the VFP OLE DB provider and which database you want to access (or directory in the case of free tables), and call its Open() method to establish a connection with the data. Here's an example that accesses the sample TESTDATA database that comes with VFP (taken from ADORECORDSET.PRG, included with this month's Subscriber downloads):

```
local loConn as ADODB.Connection
loConn = createobject('ADODB.Connection')
loConn.ConnectionString = 'Provider=VFPOLEDB;' + ;
  'Data Source=' + addbs(_samples) + 'DATA\TESTDATA.DBC'
loConn.Open()
```

The connection string accepts the following keywords:

- Provider: This should be "VFPOLEDB" (case is unimportant).
- Data Source: Use this to specify the path to the DBC or the directory containing the free tables you want to access.
- Mode: This optional keyword specifies how the DBC and tables are opened. The default mode is Share Deny None, meaning the DBC and tables are opened for sharing. Other acceptable values are Read, ReadWrite, Share Deny Read, Share Deny Write, and Share Exclusive.
- Collating Sequence: Specify the collating sequence to use for character fields in this optional keyword. The default is Machine.

Once you have a connection, you can set the ActiveConnection and other properties of an ADO RecordSet object, and then pass a SQL SELECT statement to its Open() method to fill it with some data.

The following example selects all records from the ORDERS tables into a client-side cursor with optimistic record locking:

```
local loRS as ADODB.RecordSet
loRS = createobject('ADODB.RecordSet')
loRS.ActiveConnection = loConn
loRS.CursorLocation   = 3  && adUseClient
loRS.CursorType       = 3  && adOpenStatic
loRS.LockType         = 3  && adLockOptimistic
loRS.Open('select * from ORDERS')
wait window transform(loRS.RecordCount) + ;
  ' records were returned'
```

When I typed this code, IntelliSense automatically inserted the in-line comments for the CursorLocation, CursorType, and LockType properties because I declared loRS as being an ADODB.RecordSet object. The text shown in the comments are the names of ADO enums (similar to VFP #DEFINE constants); see ADOVFP.H, included with this month's Subscriber downloads, for a complete set of enums defined as VFP constants.

You can instantiate an ADO Command object if you want to execute other types of SQL commands. The following code, for example, will delete all records for the LILAS customer:

```
local loCommand as ADODB.Command
loCommand = createobject('ADODB.Command')
loCommand.ActiveConnection = loConn
loCommand.CommandText = "delete from ORDERS " + ;
  "where CUST_ID = 'LILAS'"
loCommand.Execute()
```

You can use VFP SQL syntax with the VFP OLE DB provider. That means code like the following, which won't work with other providers because they don't support the $ operator, works just fine with VFP's:

```
loRS.Open("select * from CUSTOMER " + ;
  "where 'FUTT' $ upper(COMPANY)")
```

Parameterized views are a big problem—if you try to open them through the OLE DB provider, you get "SQL column *parameter name* not found." Creating an ADO Parameter object with the proper name and value doesn't help. The reason for this problem is that the way VFP handles parameters (using *?parameter name*) is incompatible with the way OLE DB handles them. It's hard to say whether this can even be resolved in a future release or not.

You can specify the behavior of the OLE DB provider by setting provider-specific properties of the Connection object. These properties are accessed through the Properties collection. The following properties are available:

- SET ANSI
- SET BLOCKSIZE
- SET DELETED
- SET EXACT
- SET NULL
- SET PATH
- SET REPROCESS
- SET UNIQUE

Here's an example that prevents deleted records from being processed:

```
loConn.Properties('SET DELETED').Value = .T.
```

The following code (taken from ADOPROPERTIES.PRG) displays all properties in the Properties collection:

```
local loConn as ADODB.Connection, ;
  loProperty as ADODB.Property
loConn = createobject('ADODB.Connection')
loConn.ConnectionString = 'Provider=VFPOLEDB;' + ;
  'Data Source=' + addbs(_samples) + 'DATA\TESTDATA.DBC'
loConn.Open()
for each loProperty in loConn.Properties
  ? loProperty.Name, loProperty.Value
next loProperty
loConn.Close()
```

Updating VFP data

The VFP OLE DB provider doesn't support dynamic ADO cursors (RecordSet.CursorType = 2, the adOpenDynamic enum). However, you can create an updateable RecordSet one of two ways. The first is to use a server-side cursor (RecordSet.CursorLocation = 1, the adUseServer enum, which is the default) and open a table or view with the USE command:

```
loRS.Open('use CUSTOMER')
```

The other way is to use a client-side cursor:

```
loRS.CursorLocation = 3  && adUseClient
loRS.Open('select * from CUSTOMER')
```

In either case, you can then make changes to the RecordSet and use its Update() or UpdateBatch() methods to write the changes back to the source tables. Although you can open a client-side cursor with the USE command, you'll get the following error when you call Update(): "OLE IDispatch exception code 0 from Microsoft Cursor Engine: Insufficient base table information for updating or refreshing." To see an example of updating a VFP table, run ADOUPDATE.PRG.

Stored procedures and database events

In addition to retrieving data or issuing SQL commands with the OLE DB provider, you can call stored procedures of a DBC. This is done through the ADO Command object. Here's an example of a stored procedure that calculates and returns the total sales for the specified customer:

```
procedure GetTotalSalesByCustomer(tcCustID)
select sum(ORDER_AMT) as TOTALSALES ;
  from ORDERS ;
  where CUST_ID = tcCustID ;
  into cursor SALES
return TOTALSALES
```

Here's how this procedure is called using the OLE DB provider. Note that you have to set the CommandType property to 1 (the adCmdText enum) or 8 (adCmdUnknown) rather than 4 (adCmdStoredProc) and that the resulting RecordSet has a single row and a single column called "Return_Value." (This code is taken from ADOSTOREDPROC.PRG).

```
local loConn as ADODB.Connection, ;
  loRS as ADODB.RecordSet, ;
  loCommand as ADODB.Command
loConn    = createobject('ADODB.Connection')
loCommand = createobject('ADODB.Command')
loConn.ConnectionString = 'Provider=VFPOLEDB;' + ;
  'Data Source=' + addbs(_samples) + 'DATA\TESTDATA.DBC'
loConn.Open()
lcCommand = "GetTotalSalesByCustomer('ALFKI')"
loCommand.ActiveConnection = loConn
```

```
loCommand.CommandType      = 8  && adCmdUnknown
loCommand.CommandText      = lcCommand
loRS = loCommand.Execute()
wait window 'Total sales for ALKFI: ' + ;
  transform(loRS.Fields('Return_Value').Value)
loConn.Close()
```

One thing that may not be intuitive is what happens inside the stored procedure. Can it return a VFP cursor as an ADO RecordSet? The thing to remember with a stored procedure is that it's being called from within the VFP engine, so open tables and cursors aren't accessible to the outside world. That means you can't expect to do something like a SQL SELECT into a cursor and expect that cursor to somehow be returned by the OLE DB provider to the client code. The only thing the OLE DB provider can see is the return value of the stored procedure.

There are some serious limitations in what you can do in a stored procedure. Here are the basic rules:

- Potentially destructive commands and functions, including anything that writes to disk (with the exception of updating fields in a table), aren't allowed. Examples of these functions are STRTOFILE(), FOPEN(), DELETE FILE, and SET TEXTMERGE.
- The OLE DB provider doesn't contain VFP's Object Manager, so no OOP features are supported. This means, for example, you can't use CREATEOBJECT() to instantiate an object to do the work.
- ON ERROR isn't supported, so there's essentially no error handling.
- Macro expansion isn't permitted.
- For obvious reasons, there's no user interface support, so commands and functions that request or output values or otherwise display a user interface aren't allowed. Functions in this category include INPUTBOX(), GETFILE(), and MESSAGEBOX().

The "Supported Visual FoxPro Commands and Functions" and "Supported Visual FoxPro SET Commands" topics in the VFP Help file list the commands and functions you can use, while the "Unsupported Visual FoxPro Commands and Functions" topic lists those you can't use. Because of the limited set of commands and functions you can use, creating useful stored procedures can be a challenge, and debugging them a nightmare.

You might want to insert conditional code into your stored procedures that executes different code when they're called from the OLE DB provider. The VERSION() function returns "Microsoft OLE DB Provider for Visual FoxPro *version number* for Windows" when called by the provider, so you can use code like the following:

```
if 'OLE DB' $ version()
* code executed from OLE DB provider
else
* code executed from within VFP
endif
```

DBC events are fired from the OLE DB provider just as they are from within VFP, but since they're stored procedures, they have the same limited range of things you can do with them. For example, popping up a dialog asking the user to log in when they try to open a table won't work, since that requires a user interface.

ADODBCEVENTS.PRG demonstrates DBC events and field default values firing when tables are accessed from the provider. It starts by cloning the VFP TESTDATA database (since it makes changes to the database and I didn't want to mess up your original DBC), and then adds some stored procedures to the DBC. The DBC_AfterOpenTable event, which fires after a table is opened, logs when any tables are opened to the ACCESSLOG table:

```
procedure DBC_AfterOpenTable(cTableName)
local lcDir
lcDir = addbs(justpath(dbc()))
do case
```

```
  case used('AccessLog')
  case file(lcDir + 'AccessLog.dbf')
    use (lcDir + 'AccessLog') again in 0 shared
  otherwise
    create table (lcDir + 'AccessLog') free ;
      (Table C(20), ;
      When T)
endcase
insert into AccessLog values (cTableName, datetime())
use in AccessLog
set null off
```

The ORDER_ID field in the ORDERS table has a DefaultValue property that calls the NEXTID stored procedure to assign the next available order number when a record is added. To test these two stored procedures, the ORDERS table is opened and a record added, both natively and through ADO. The ACCESSLOG and ORDERS tables are then browsed so you can see that any access to the table (native or through the OLE DB provider) is logged and that the NEXTID procedure is called to assign a new ORDER_ID in both cases.

And the winner is…

One question frequently asked is: should I use ODBC or ADO to access my data? The answer is: it depends (this is pretty much the answer to any VFP question <g>). Although it originally had a bad reputation for performance, ODBC is much improved in that regard these days. In some informal testing I did, ODBC was only slightly slower than ADO for accessing VFP data. So, which you should use depends on your requirements. If you need to work with a cursor (for example, to display the results in a grid or a VFP report), ODBC is a better choice since you'd have to convert an ADO RecordSet to a cursor before you can use it. On the other hand, since VFP cursors can't be accessed anywhere but the current datasession of the current VFP session, ADO is obviously the choice when the data has to be passed to something (you can't pass a cursor but you *can* pass a RecordSet object). If access is being done by a data object on a Web server, scalability may be a factor, so ADO might be a better choice as a result.

Of course, this debate ignores XML, which (in my opinion) will ultimately replace ADO as the data exchange mechanism of choice. The ability to easily convert between a cursor and XML using VFP 7's new CURSORTOXML() and XMLTOCURSOR() functions makes XML additionally attractive in new VFP applications.

Conclusion

The OLE DB provider for VFP is a welcome addition to our arsenal of data access tools. ADO is easy to work with, object-oriented, and has a number of benefits over ODBC. If you need to access VFP data other than directly through the VFP data engine, be sure to check out what this new provider can do for you.

Doug Hennig is a partner with Stonefield Systems Group Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT) and co-author (along with Tamar Granor and Kevin McNeish) of "What's New in Visual FoxPro 7.0" from Hentzenwerke Publishing. Doug has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Certified Professional (MCP). Web: www.stonefield.com Email: dhennig@stonefield.com