dapt Your Cursors to CursorAdapters

Doug Hennig

The CursorAdapter class is one of the most important new features in VFP 8 because it provides an easy-to-use, consistent interface to remote data. In this month's article, Doug examines CursorAdapters and how they work. Next month, we'll take a look at some advanced uses of them.

More and more VFP developers are storing their data in something other than VFP tables, such as SQL Server or Oracle. There are a lot of reasons for this, including fragility of VFP tables (both perceived and actual), security, database size, and corporate standards. Microsoft has made access to non-VFP data easier with every release, and has even been encouraging it with the inclusion of MSDE (Microsoft Data Engine, a free, stripped-down version of SQL Server) on the VFP 7 CD.

However, accessing a backend database has never been quite as easy as using VFP tables. In addition, there are a variety of mechanisms you can use to do this:

- Remote views, which are based on ODBC connections.
- SQL passthrough (SPT) functions, such as SQLCONNECT(), SQLEXEC(), and SQLDISCONNECT(), which are also based on ODBC connections.
- ActiveX Data Objects, or ADO, which provide an object-oriented front end to OLE DB providers for database engines.
- XML, which is a lightweight, platform-independent, data transport mechanism.

If you've spent any time working with these mechanisms, one of the things you've likely noted is that each of them 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.

VFP 8 makes accessing remote data much easier than in earlier releases because of a new base class: CursorAdapter. In my opinion, CursorAdapter is one of the biggest new features in VFP 8. The reasons I think it's so cool are:

- 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 SQL Server data, and for some reason you want to change to use ADO instead. All you need to do is change the DataSourceType property of the CursorAdapters and change the connection to the backend 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.

PEMs

Let's start our examination of CursorAdapter by looking at its properties, events, and methods (PEMs). We won't discuss all of the PEMs here, just the more important ones. See the VFP 8 documentation for the complete list.

DataSourceType

This property is a biggie: it determines the behavior of the class and what kinds of values to put into some of the other properties. The valid choices are "Native", which indicates that you're using native tables, or "ODBC", "ADO", or "XML", which means you're using the appropriate mechanism to access the data.

DataSource

This is the means to access the data. VFP ignores this property when DataSourceType is set to "Native" or "XML". For ODBC, set DataSource to a valid ODBC connection handle (meaning you have to manage the connection yourself). In the case of ADO, DataSource must be an ADO RecordSet that has its ActiveConnection object set to an open ADO Connection object (again, you have to manage this yourself).

UseDEDataSource

If this property is set to .T. (the default is .F.), you can leave the DataSourceType and DataSource properties alone since the CursorAdapter will use the DataEnvironment's properties instead (VFP 8 adds DataSourceType and DataSource to the DataEnvironment class as well). An example of when you'd set this to .T. is when you want all the CursorAdapters in a DataEnvironment to use the same ODBC connection.

SelectCmd

In the case of everything but XML, this is the SQL SELECT command used to retrieve the data. In the case of XML, this can either be a valid XML string that can be converted into a cursor (using an internal XMLTOCURSOR() call) or an expression (such as a UDF) that returns a valid XML string.

CursorSchema

This property holds the structure of the cursor in the same format as you'd use in a CREATE CURSOR command (everything between the parentheses in such a command). Here's an example: CUST_ID C(6), COMPANY C(30), CONTACT C(30), CITY C(25). Although it's possible to leave this blank and tell the CursorAdapter to determine the structure when it creates the cursor, it works better if you fill CursorSchema in. For one thing, if CursorSchema is blank or incorrect, you'll either get errors when you open the DataEnvironment of a form or you won't be able to drag and drop fields from the CursorAdapter to the form to create controls. Fortunately, the CursorAdapter Builder that comes with VFP can automatically fill this in for you.

AllowDelete, AllowInsert, AllowUpdate, and SendUpdates

These properties, which default to .T., determine if deletes, inserts, and updates can be done and if changes are sent to the data source.

KeyFieldList, Tables, UpdatableFieldList, and UpdateNameList

These properties, which serve the same purpose as the same-named CURSORSETPROP() properties for views, are required if you want VFP to automatically update the data source with changes made in the cursor. KeyFieldList is a comma-delimited list of fields (without aliases) that make up the primary key for the cursor. Tables is a comma-delimited list of tables. UpdatableFieldList is a comma-delimited list of fields (without aliases) that can be updated. UpdateNameList is a comma-delimited list that matches field names in the cursor to field names in the table. The format for UpdateNameList is as follows: CURSORFIELDNAME1 TABLE.FIELDNAME1, CURSORFIELDNAME2 TABLE.FIELDNAME2, ... Note that even if UpdatableFieldList doesn't contain the name of the primary key of the table (because you don't want that field updated), it must still exist in UpdateNameList or updates won't work.

**Cmd, *CmdDataSource, *CmdDataSourceType*

If you want to specifically control how VFP deletes, inserts, or updates records in the data source, you can assign the appropriate values to these sets of properties (replace the * above with Delete, Insert, and Update).

CursorFill(UseCursorSchema, NoData, Options, Source)

This method creates the cursor and fills it with data from the data source (although you can pass .T. for the NoData parameter to create an empty cursor). Pass .T. for the first parameter to use the schema defined in CursorSchema or .F. to create an appropriate structure from the data source (in my opinion, this behavior is reversed). MULTILOCKS must be set on or this method will fail. If CursorFill fails for any reason, it returns .F. rather than raising an error; use AERROR() to determine what went wrong (although be prepared for some digging, since the error messages you get often aren't specific enough to tell you exactly what the problem is).

CursorRefresh()

This method is similar to the REQUERY() function: it refreshes the cursor's contents.

Before() and After*()*

Pretty much every method and event has before and after "hook" events that allow you to customize the behavior of the CursorAdapter. For example, in AfterCursorFill, you can create indexes for the cursor so they're always available. In the case of the Before events, you can return .F. to prevent the action that trigger it from occurring (this is similar to database events).

Example

Here's an example (CursorAdapterExample.prg, included in this month's Subscriber Downloads) that retrieves certain fields for Brazilian customers from the Customers table in the Northwind database that comes with SQL Server. The cursor is updateable, so if you make changes in the cursor, close it, and then run the program again, you'll see that your changes were saved to the backend.

```
local lcConnString, ;
  loCursor as CursorAdapter, ;
  laErrors[1]
lcConnString = 'driver=SQL Server;server=(local);' + ;
  'database=Northwind;uid=sa;pwd=;trusted_connection=no'
* change password to appropriate value for your database
loCursor = createobject('CursorAdapter')
with loCursor
  .Alias              = 'Customers'
  .DataSourceType     = 'ODBC'
  .DataSource         = sqlstringconnect(lcConnString)
  .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
```

DataEnvironment and Form Changes

To support the new CursorAdapter class, several changes have been made to the DataEnvironment and Form classes and their designers.

First, as I mentioned earlier, the DataEnvironment class now has DataSource and DataSourceType properties. It doesn't use these properties itself but they're used by any CursorAdapter member that has UseDEDataSource set to .T. Second, you can now create DataEnvironment subclasses visually using the Class Designer (woo hoo!).

As for forms, you can now specify a DataEnvironment subclass to use by setting the new DEClass and DEClassLibrary properties. If you do this, anything you've done with the existing DataEnvironment (cursors, code, etc.) will be lost, but at least you're warned first. A cool new feature of forms that's somewhat related is the BindControls property; setting this to .F. in the Property Window means that VFP won't try to data bind the controls at init time, only when you set BindControls to .T. What's this good for? Well, how many times have you cursed that parameters are passed to Init, which fires after all the controls have been initialized and bound to their ControlSources? What if you want to pass a parameter to a form that tells it which table to open or other things that affect the ControlSources? This new property makes this issue a snap.

Advantages

There are lots of good reasons to use CursorAdapters instead of remote views, SPT, ADO, or XML.

- Each mechanism has a different interface. With remote views, you open them with a USE command. With SPT, you use SQLCONNECT() and SQLEXEC() to create a cursor. With ADO, you instantiate ADO Connection and RecordSet objects (and possibly a Command object, depending on how you're retrieving the data) and call their Open methods. With XML, you first obtain an XML string from somewhere (such as a COM component in a n-tier application or SQL Server via SQLXML), then use XMLTOCURSOR() to convert it to a VFP cursor. Writing updates to the backend is also different with each mechanism. So, as you move from one mechanism to another, you have new techniques to learn, plus you may have to rewrite a lot of existing code.

  CursorAdapters have a consistent interface regardless of which mechanism you use for data retrieval. You set various properties of the object and call the CursorFill method to retrieve the data, work with the cursor as you would any other VFP cursor, and then call TABLEUPDATE() (or let VFP handle it automatically) to write changes back.

- In a development environment, you often want to open a cursor from the Command Window to review its contents. This is easy for remote views (just USE the view), but a lot more work for SPT, ADO, and XML.

  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.

- Upsizing an existing application can be relatively easy with remote views: you replace the existing Cursor objects that point to local tables or views in a DataEnvironment with those that reference remote views instead. With SPT, ADO, and XML, you have to retool your entire data access schemes.

  It's as easy to upsize an existing application to use CursorAdapters as it is with remote views; you simply replace Cursor objects with CursorAdapter objects instead.

- Remote views are easy to work with in the Form and Report Designers. You can add one to the DataEnvironment and then take advantage of the visual support the DataEnvironment 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. SPT, ADO, and XML don't have such visual support.

  CursorAdapters have the same support in a DataEnvironment that remote views do.

- Remote views are easy to create using the View Designer. Although it had serious limitations in the past, especially when dealing with views joining more than two tables, VFP 8 fixes most of these problems and adds a lot more capabilities, such as two-way editing: you can change the code in the SQL window and see those changes reflected in the visual part of the View Designer. With SPT, ADO, and XML, it's more work, since you have to code everything: creating and closing the connection, the SQL SELECT statements to execute, and so on.

  VFP 8 includes a CursorAdapter Builder that makes short work of setting the properties necessary to retrieve and update data. It even includes a SelectCmd builder that, while not as visual as the View Designer, allows you to select which fields should be retrieved from the remote tables using a "mover" control.

- It's easy to update the backend database with changes made in remote views and ADO RecordSets. Assuming the properties of the view have been set up properly, you simply call

TABLEUPDATE(). In the case of ADO, you call RecordSet.Update() or UpdateBatch(). With SPT and XML, you have a bunch of manual work to do to pass updates back.

As we saw earlier, performing updates with CursorAdapter can be as easy as setting a few properties to make VFP do all the work, or you can have more flexibility by specifying how deletes, inserts, and updates are each handled.

- Because the result set created by remote views and SPT are VFP cursors, they can be used anywhere in VFP: in a grid, a report, processed in a SCAN loop, and so forth. ADO and XML, on the other hand, must be converted to cursors before they can be used in these ways, which adds complexity and additional processing time to your application.

  The result set of a CursorAdapter is a VFP cursor, so it has the same benefits as remote views and SPT. Even better, you get a VFP cursor even if the data source is ADO and XML, because the CursorAdapter automatically takes care of conversion to and from a cursor for you.

- 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. This isn't a problem with SPT, ADO, or XML.

  CursorAdapters don't suffer from this problem: you can easily change the SelectCmd property to change what data is retrieved and how.

- You can't call a stored procedure from a remote view, so a remote view needs direct access to the underlying tables. This may be an issue with your application's database administrators; some DBAs believe that data access should only be via stored procedures for security and other reasons. Also, because they are precompiled on the backend, stored procedures often perform significantly faster than SQL SELECT statements. With SPT, ADO, and XML, you can call stored procedures if necessary.

  CursorAdapters can also use stored procedures by simply setting SelectCmd as necessary.

- Remote views live in a database container, so that's one more set of files you have to maintain and install on the client's system. Also, when you open a view, VFP attempts to lock the view's records in the DBC, even if it's 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'll need to plan for. One further problem: 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. Because they have nothing to do with DBCs, none of these are an issue for SPT, ADO, and XML.

  Since they don't live in a DBC, CursorAdapters don't have these issues either.

- Because they work with ODBC only, remote views and SPT are stuck in the "client-server" model of direct data connections. ADO and XML are the mechanisms of choice for passing data between the layers in an n-tier application.

  Because CursorAdapters can create VFP cursors from ADO or XML, they're ideal for use in the UI layer of an n-tier application.

Summary

I think CursorAdapter is one of the biggest and most exciting enhancements in VFP 8 because it provides a consistent and easy-to-use interface to remote data, plus, as we'll see in future articles, it allows us to create reusable data classes. Next month, we'll look at specifics of accessing native or remote data using ODBC,

ADO, and XML. The following month, we'll look at creating reusable data classes, and discuss how to use CursorAdapters in reports.

Doug Hennig is a partner with Stonefield Systems Group Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT) and Stonefield Query, co-author of "What's New in Visual FoxPro 7.0" and "The Hacker's Guide to Visual FoxPro 7.0", both from Hentzenwerke Publishing, and author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's Pros Talk Visual FoxPro series. He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals", both 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 and www.stonefieldquery.com Email: dhennig@stonefield.com