anguage Enhancements in VFP 7, Part III

Doug Hennig

This is the third in a series of articles on language enhancements in VFP 7. This article covers the last of the improvements to existing commands and functions, and starts discussing new commands and functions.

Sharp-eyed readers may have noticed I missed enhancements to some existing commands and functions in this and the previous two articles; for example, COMARRAY(), DEFINE CLASS, and FUNCTION have new features but haven't been mentioned. The reason I haven't covered them so far is because I want to discuss improvements in OOP, COM, and UI as separate topics. I'll discuss both existing and new commands and functions when we get to those topics.

SET REPROCESS

The new SYSTEM clause for this command allows you to control record locking in the "system" data session, which is used for internal access to tables such as databases, the resource file, SCX and VCX files, etc. To see an example of this, fire up two instances of VFP 7, type SET REPROCESS TO AUTOMATIC SYSTEM in the Command window of each one, then modify a class in a VCX in one instance and try to modify the same class in the same VCX in the second. Notice that VFP will wait to get a lock on the class' records in the VCX until you either close the class in the other instance or press Esc. Press Esc to cancel the attempt. Now type SET REPROCESS TO 2 SECONDS SYSTEM and try again. This time, VFP tries for two seconds to lock the records, and gives up when it can't.

SET TEXTMERGE, TEXT, and SET('TEXTMERGE')

SET TEXTMERGE is a useful command for creating text files where the output has to include the results of VFP expressions; strings between << and >> (you can change the delimiters to something else using SET TEXTMERGE DELIMITERS) are assumed to be expressions to be evaluated if SET TEXTMERGE ON is used. For example, the following code creates an HTML file called TEST.HTML that shows company and contact names in a table (this is a truncated version of TESTTEXTMERGE.PRG included in this month's Subscriber Downloads):

```
use (_samples + 'data\customer')
set textmerge on to TEST.HTML noshow
\\<html>
\<body>
\<table border=1>
\<tr><th>Company</th><th>Contact</th></tr>
scan
  \<tr>
  \<td><<trim(COMPANY)>></td>
  \<td><<trim(CONTACT)>></td>
  \</tr>
endscan
\</table>
\</body>
\</html>
set textmerge to
```

One problem: what if you really want the output in a variable rather than a file (for example, you want to return the HTML as a string from a VFP COM object)? You'd have to use FILETOSTR() to read the contents of the file back into a variable and then delete the file.

Fortunately, VFP 7 adds the ability to send the text merge directly to a variable rather than to a file by using the TO MEMVAR clause. For example, in the above code, changing the SET TEXTMERGE command to:

```
set textmerge on to memvar lcHTML noshow
```

results in lcHTML containing the HTML and no file being created.

When was the last time you used the TEXT command? Back in FoxPro 1.0? Never? That's because this little-used command originally sent a block of hard-coded text between TEXT and ENDTEXT statements to the screen or the printer, and who needs to do that anymore? Although it's been enhanced over the years to support evaluation of expressions (if SET TEXTMERGE ON is used) and output to a file, few developers I know ever used it. In VFP 7, however, this command is much more useful because, like SET TEXTMERGE, you can now send the block of text to a variable. In addition, the new ADDITIVE, TEXTMERGE, and NOSHOW commands control whether existing variable contents are overwritten or not, whether expressions are evaluated or not (so you don't have to use SET TEXTMERGE ON as a separate command), and whether output is echoed to the screen or not (so you don't have to use SET CONSOLE OFF to suppress it). The following code (taken from TESTTEXT.PRG) does the same thing as the previous program:

```
use (_samples + 'data\customer')
text to lcHTML textmerge noshow
<html>
<body>
<table border=1>
<tr><th>Company</th><th>Contact</th></tr>
endtext
scan
  text to lcHTML additive textmerge noshow
  <tr>
  <td><<trim(COMPANY)>></td>
  <td><<trim(CONTACT)>></td>
  </tr>

  endtext
endscan
text to lcHTML additive textmerge noshow
</table>
</body>
</html>
endtext
```

Notice the difference in techniques: SET TEXTMERGE permits lines of code to be interspersed with the output because you have to prefix output with \ or \\, while everything between TEXT and ENDTEXT is considered part of the output so prefixes aren't needed but you can't mix code and output. That means TEXT is more useful when no processing code is needed and SET TEXTMERGE is more useful when code is needed.

SET('TEXTMERGE') has been enhanced to return the output filename (but not the variable name), SHOW or NOSHOW setting, and the nesting level for evaluations in TEXT statements.

SET('CENTURY')

Passing 3 as the second parameter returns the rollover date setting in the Regional Options Control Panel applet of Windows 98, ME, and Windows 2000 (for example, on my system, it returned 2029). In Windows 95 and NT, it returns -1.

STRTRAN()

STRTRAN() now supports a new sixth parameter to permit searches to be case-insensitive and to determine what case to use for the replacement expression. Here's an example of the use of this new parameter (taken from TESTSTRTRAN.PRG):

```
clear
lcString   = 'The quick BROWN fox'
lcOldValue = 'brown'
lcNewValue = 'green'
? strtran(lcString, lcOldValue, lcNewValue)
&& displays The quick BROWN fox
```

```
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 1)
&& displays The quick green fox
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 2)
&& displays The quick BROWN fox
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 3)
&& displays The quick GREEN fox
?
lcOldValue = 'BROWN'
? strtran(lcString, lcOldValue, lcNewValue)
&& displays The quick green fox
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 1)
&& displays The quick green fox
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 2)
&& displays The quick GREEN fox
? strtran(lcString, lcOldValue, lcNewValue, -1, -1, 3)
&& displays The quick GREEN fox
```

Notice the following interesting things:

- Use 1 as the sixth parameter for case-insensitivity. The first STRTRAN() fails to do anything because "brown" isn't found in lcString, but the second succeeds because it's case-insensitive.
- Specifying 2 or 3 as the sixth parameter tells VFP to use the case of the found text for the case of the replacement text. Notice in the fourth, seventh, and eighth examples above that "GREEN" appears in the result even though "green" was specified as the replacement text. That's because the text being replaced ("BROWN") is all upper-cased.

In VFP 6, you might think you could use something like this for case-insensitivity:

```
lcString2 = strtran(upper(lcString1), lcValue1, lcValue2)
```

However, that results in lcString2 being completely upper-cased, which may not be what you want.

SYS(3054)

This indispensable function, which tells you the Rushmore optimization levels for SQL SELECT commands (including those executed when a view is opened), has been made even better in VFP 7. Passing 2 or 12 for the first parameter tell VFP to include the SQL SELECT statement in the output (making it easier to figure out which of several SQL SELECT statements the output applies to). A new third parameter sends the results to a variable rather than _SCREEN, which allows you to control the output (for example, you could log it to a file) or to automate analysis (for example, warning the developer if "none" or "partial" appears in the results but not if "full" does). The following, taken from TESTSYS3054.PRG, shows the use of these new features:

```
close tables all
use (_samples + 'data\customer')
use (_samples + 'data\orders') in 0
sys(3054, 12, 'lcRushmore')
select CUSTOMER.COMPANY, ;
    max(ORDERS.ORDER_DATE) ;
  from CUSTOMER ;
    join ORDERS ;
      on CUSTOMER.CUST_ID = ORDERS.CUST_ID ;
  group by CUSTOMER.CUST_ID ;
  into cursor TEST
clear
? lcRushmore
sys(3054, 0)
```

USE

Whether you use remote views in production applications or not (discussions in a recent thread on the Universal Thread, http://www.universalthread.com, and in documents on the FoxWiki, http://fox.wikis.com, have been just a tad heated!), they are still useful tools. For example, although I normally use SQL

Passthrough or ADO when accessing non-VFP data in production, in a development environment, I sometimes find it handy to create a DBC and some remote views so I can quickly browse the data. One downside of remote views compared to SQL Passthrough, though, is that the connection information is hard-coded (either in a connection stored in the database or in an ODBC datasource). This means this information can't be changed, so you can't use the same remote view to access data on a different server, for example. Also, user names and passwords aren't encrypted in the DBC, so that's a potential security risk.

VFP 7 adds the ability to specify a connection string in the USE command for a remote view. If one is specified, it overrides the defined connection information for the view. If an empty string is specified, VFP displays the Select Data Source dialog.

To see an example of this, do the following steps. First, create an ODBC datasource called "Northwind" for the Northwind database that comes with Access using the Access ODBC driver (if you don't have Access installed, used the SQL Server Northwind database and the SQL Server driver instead). Next, type the following in the Command window (the TEST database referenced in this code is included in this month's Subscriber downloads):

```
open database TEST
use rv_customers connstring 'dsn=Northwind'
```

What's so remarkable about this? Well, the RV_CUSTOMERS view is defined as:

```
create sql view "rv_customers" ;
  remote connect "NorthwindConnection" ;
  as select * from customers customers
```

The NorthwindConnection connection specified here uses the SQL Server driver to access the Northwind database on my server named DHENNIG. Specifying CONNSTRING in the USE command allows you to create a cursor from the CUSTOMERS table in the Access database on your system instead.

The benefit of this enhancement is that you can store the server, user name, and password in a table, build a connection string, and then use it in the USE command for your remote views. This gives you both flexibility (you can easily change the server, user name, and password in the table) and security (the password can be encrypted in the table and decrypted when the connection string is built).

VALIDATE DATABASE RECOVER

Formerly, this command could only be used in "interactive mode" (that is, from the VFP Command window). In VFP 7, this command can now be used in code, allowing you to use it in a runtime environment. Of course, since this is a potentially dangerous command (it can remove tables from a DBC, for example) that displays dialogs that are confusing to the typical end-user ("What *is* a backlink and why would I want to update or remove it?"), you'll want to only use it in "administrator" functions.

New Commands and Functions

We've finished the list of enhancements to existing commands and functions (with the exception, as I pointed out earlier, to those that fall into the OOP, COM, and UI topics, which we'll get to in a future article), so it's time to start in new commands and functions.

ADLLS()

This new function fills an array with declared DLL functions. The array contains the name of the function, the alias name it was given, and the DLL it exists in. TestADLLs.prg shows an example of this, showing the DLL functions loaded by the FoxPro Foundation Classes Registry class:

```
clear
oRegistry = newobject('Registry', ;
  home() + 'FFC\Registry.vcx')
oRegistry.LoadRegFuncs()
adlls(laDLLs)
display memory like laDLLs
```

This function and the new capability of CLEAR DLLS to remove a single DLL function from memory (discussed in last month's article) make unloading only certain sets of functions much easier in VFP 7.

ALANGUAGE()

This function fills an array with VFP language elements. The first parameter is the array to fill and the second is the type of language elements desired: commands, functions, base classes, or DBC events. In the case of functions, the array will have two columns: one for the name of the function and one showing a range of the minimum required and total number of parameters plus a flag indicating if the entire function name is required in code. Run TestALanguage.prg (included with this month's Subscriber Downloads) to see how this function works.

Why did Microsoft add this function to the language? Mostly, I think, because of Intellisense. I suspect that they needed something like this themselves for Intellisense or the Intellisense Manager, so we get it for free. Would you actually use this function yourself? I doubt it, unless you need to parse FoxPro code for some purpose (such as a documenter or Beautify-like utility). I originally imagined using it for a utility to pick field names out of an expression such as UPPER(CUSTOMER.COMPANY) + LEFT(CUSTOMER.ZIPCODE, 5); having an array of language elements would allow me to remove them from the expression, leaving only constants and variable or field names behind. However, it turned out that two other new functions we'll discuss later, GETWORDCOUNT() and GETWORDNUM(), help create a much faster and tighter utility, as we'll see.

ANETRESOURCES()

This cool little function fills an array with the share or printer resources on a specific server. You pass it the array to fill, the name of the server, and a parameter indicating whether you want shares, printers, or both types of resource. You can run TestANetResources.prg to see how it works; pass it the name of the server (for example, "\\myservername") to get resources for.

There's probably a Windows API call that can provide this information for you in VFP 6, but I haven't dug into it. You can get close using the EnumNetworkDrives and EnumPrinterConnections methods of the Windows Script Host Wscript.Network object, but those methods only return drives and printers you're currently connected to rather than the resources that are available but you're not connected to. For details on the Windows Script Host, see http://msdn.microsoft.com/scripting/.

APROCINFO()

Like ALANGUAGE(), I suspect this new function was added because new tools that come with VFP require it rather than because VFP developers requested it. It fills an array with information from a PRG file, including things like PROCEDURE and FUNCTION statements, compiler directives, and class definitions. Run TestAProcInfo.prg to see how it works. As with ALANGUAGE(), it might be useful if you're creating a documenter or some other utility that works with PRGs.

By the way, if you haven't already done so, you might want to download a free utility written by Michael Emmons called Class Navigator that provides a Class Browser-like tool for PRG files. It's available at www.comcodebook.com. I bet if Michael were to write this program again, he'd use this new function because it seems perfectly suited to what his utility does.

Conclusion

Next month, we'll carry on looking at new commands and functions. After we've finished with those, we'll look at OOP, COM, and UI improvements in VFP 7.

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Saskatchewan, Canada. He is the author Stonefield's award-winning Stonefield Database Toolkit. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at the Microsoft FoxPro Developers Conferences (DevCon) since 1997, as well as user groups and regional conferences all over North America. He is a Microsoft Most Valuable Professional (MVP) and Microsoft Certified Professional (MCP). He can be reached at dhennig@stonefield.com.