# anguage Enhancements in VFP 7, Part V

*Doug Hennig*

**This fifth in a series of articles on language enhancements in VFP 7 covers the remaining new general-purpose commands and functions.**

### GETWORDCOUNT() and GETWORDNUM()

As was done with several filename processing functions (JUSTPATH(), JUSTSTEM(), JUSTFNAME(), etc.) in VFP 6, these two functions are being taken from FOXTOOLS.FLL and added as native functions in VFP 7. GETWORDCOUNT() returns the number of words found in a string and GETWORDNUM() returns the specified word (by index) from a string. Although the default delimiters are space, tab, and carriage return, you can specify different delimiters for processing different types of strings (for example, you could specify a comma and carriage return to process comma-delimited files).

My first thought when reading about these functions was "so what?"; since I never used these functions in FOXTOOLS, why would I need them in the language? Then one use hit me: finding fields in VFP expressions. Because of the types of tools I develop, I frequently need to determine which fields are involved in an expression. For example, I might want to know if a particular field has an index on it so if someone wants to remove that field, the index needs to be removed as well. This is more complicated than it seems. Just using the $ or AT() functions is prone to being fooled; for example, "TAX" $ "UPPER(CUSTOMER.TAXSTATUS)" returns .T. so it incorrectly appears that the TAX field is used in the expression. So, to do this previously, I had to do a lot of careful (and ugly) string parsing to check exactly how a field name was used in an expression. Once I started thinking about GETWORDCOUNT() and GETWORDNUM(), a much simpler approach made itself clear.

FIELDSINEXPR.PRG is the new version of this code. The first parameter is an array (passed by reference using @) that it'll fill with all the fields involved in the expression passed as the second parameter. The optional third parameter is the alias to add to each field name if the field isn't aliased in the expression; if it isn't passed, unaliased fields are left unaliased. This function uses GETWORDCOUNT() to determine how many "words" there are in the expression; the delimiter for these "words" are punctuation characters that can appear in VFP expressions, such as #, <, >, (, ), etc. It then uses GETWORDNUM() in a loop to get each "word". Numbers are ignored, as are functions (the character following the word is an opening parenthesis) and strings (the character at the start of the word is a quote or opening square bracket). The "word" is then aliased if it isn't already and an alias was passed, and then if it isn't already in the array, it's added. Here's the code for this routine:

```
lparameters taFields, ;
  tcExpression, ;
  tcAlias
local lcExpression, ;
  lnFields, ;
  llAddAlias, ;
  lcWords, ;
  lnWords, ;
  lnI, ;
  lcWord, ;
  lnWord, ;
  lcChar, ;
  lcStart

* Define function and field delimiters in expressions.

#define ccDELIMITERS ' !#%^*()-+/,<>[]{}'

* Ensure the necessary parameters were passed.

assert type('taFields[1]') <> 'U' ;
  message 'FieldsInExpr: array not passed'
```

```
assert vartype(tcExpression) = 'C' and ;
  not empty(tcExpression) ;
  message 'FieldsInExpr: invalid expression passed'

* Set up the variables we need. Because a space before an
* open paren can fool us, let's remove any we find (note
* that we can't use NORMALIZE() here because a field list
* like COMPANY, CITY is reduced to just COMPANY).

lcExpression = strtran(tcExpression, ' (', '(')
lnFields     = 0
llAddAlias   = vartype(tcAlias) = 'C' and ;
  not empty(tcAlias)
lcWords      = ''

* Determine how many "words" there are in the expression,
* and then check each one. Count how many times each word
* occurs.

lnWords = getwordcount(lcExpression, ccDELIMITERS)
for lnI = 1 to lnWords
  lcWord  = getwordnum(lcExpression, lnI, ccDELIMITERS)
  lnWord  = occurs(lcWord + ',', lcWords) + 1
  lcWords = lcWords + lcWord + ','

* If this isn't a digit, see if it's a function (it's
* followed by an open paren; we need to check this
* specific occurrence of the word since it may appear
* more than once) or a string. If not, add an alias if
* necessary and if it isn't already in the array, add it.

  if not isdigit(lcWord)
    lcChar  = substr(lcExpression, ;
      at(lcWord, lcExpression, lnWord) + len(lcWord), 1)
    lcStart = left(lcWord, 1)
    if lcChar <> '(' and not lcStart $ ['"] and ;
      lcStart <> '['
      if llAddAlias and not '.' $ lcWord
        lcWord = tcAlias + '.' + lcWord
      endif llAddAlias ...
      if ascan(taFields, lcWord) = 0
        lnFields = lnFields + 1
        dimension taFields[lnFields]
        taFields[lnFields] = lcWord
      endif ascan(taFields, lcWord) = 0
    endif lcChar <> '(' ...
  endif not isdigit(lcWord)
next lnI
return lnFields
```

TESTFIELDSINEXPR.PRG tests this using several different expressions. Some of them are slightly different from others to show that FIELDSINEXPR.PRG can handle differently formatted expressions.

Another place where these functions can be used is processing HTML. The FoxWiki (an excellent VFP resource developed by Steven Black; fox.wikis.com), for example, automatically hyperlinks a word in "camel notation" (an upper-cased letter in the middle of a word, such as "FoxWiki") to another topic with that word as the title. This allows someone to enter the text of a document and automatically link it to other documents without having to enter <A HREF> tags. GETWORDCOUNT() and GETWORDNUM() would make the text parsing this kind of ability requires simpler.

While you could write functions that imitate GETWORDCOUNT() and GETWORDNUM() in VFP 6, why bother? Simply SET LIBRARY TO FOXTOOLS and use these functions as if they're built into the language.

### INPUTBOX()

This function displays a dialog in which the user can enter a single string. Interestingly, this dialog has existed in VFP since the beginning but wasn't accessible before: it's the same dialog displayed to get values for parameterized views. You can specify the prompt, dialog caption, a default value (returned if the user

chooses Cancel, presses Esc, or the dialog times out), and a timeout value. The following code (TESTINPUTBOX.PRG in the sample files) shows an example; it opens the VFP sample CUSTOMER table, asks you for a city (the default is San Francisco), and shows how many customers were found in that city:

```
use _samples + 'data\customer'
lcCity = inputbox('Enter the city to search for:', ;
  'Find City', 'San Francisco', 5000)
select count(*) from customer ;
  where city = lcCity ;
  into array laFind
wait window transform(laFind[1]) + ;
  ' customers were found in ' + lcCity
```

Because we have no control over the appearance of the dialog other than the prompt and caption, and we can't tell if the user chose Cancel if a default value was provided, it's hard to say whether this function will be used much in real applications. However, for developer tools or simple applications, it saves you having to create a form just so the user can enter a single string.

### QUARTER()

This new function returns the quarter the specified Date or DateTime expression falls in. You can specify the starting month for the year (the default is 1) so it can be used for either calendar or fiscal years. TESTQUARTER.PRG demonstrates this function.

The VFP 6 equivalent of this function, QUARTER.PRG, is pretty simple:

```
lparameters tdDate, ;
  tnStart
local lnStart, ;
  lnMonth, ;
  lnReturn
lnStart  = iif(pcount() = 1, 1, tnStart)
lnMonth  = month(tdDate) - lnStart
lnMonth  = lnMonth + iif(lnMonth < 0, 12, 0)
lnReturn = int(lnMonth/3) + 1
return lnReturn
```

### STREXTRACT()

Another useful string processing function! STREXTRACT() returns the text between delimiters. Hmm, let me think, what type of text do we know of that's littered with delimiters? Can you spell HTML and XML?

STREXTRACT() accepts up to five parameters: the string to search, the beginning and ending delimiters, the occurrence number to retrieve, and a case-sensitivity flag. You can leave out the beginning delimiters to return text from to the start of the string to the ending delimiter or the ending delimiter to return text from the beginning delimiter to the end of the string.

Here's an example: TESTSTREXTRACT.PRG reads FOXWEB.HTML into a variable, then finds and prints all hyperlinks in the document. STREXTRACT() is used for three tasks in this code. First, it returns the text between "<A HREF" and "</A>" (in the FindLink subroutine), which is the URL and text associated with it. Then, in the first lcHRef assignment statement, it returns the text between "=" and ">", which is the URL to jump to. Finally, in the lcText assignment, it retrieves the string starting from ">", which is the text associated with the URL.

```
clear
lcHTML  = filetostr('FoxWeb.html')
lcLink  = FindLink(lcHTML, 1)
lnOccur = 1
do while not empty(lcLink)
  lcHRef = strextract(lcLink, '=', '>')
  lcHRef = alltrim(strtran(lcHRef, '"'))
  lcText = alltrim(strextract(lcLink, '>'))
  ? 'Link ' + transform(lnOccur) + ': ' + lcText + ;
    ' (' + lcHRef + ')'
  lnOccur = lnOccur + 1
```

```
  lcLink  = FindLink(lcHTML, lnOccur)
enddo while not empty(lcLink)

function FindLink(tcHTML, tnOccur)
return alltrim(strextract(tcHTML, '<a href', '</a>', ;
  tnOccur, 1))
```

For the first lcHRef assignment statement, here's the code you'd have to use without STREXTRACT():

```
lnPos1 = at('=', lcLink)
lnPos2 = at('>', lcLink)
lcHRef = substr(lcLink, lnPos1 + 1, lnPos2 - lnPos1 - 1)
```

I don't know about you, but I always have to spend time figuring out what that last parameter in SUBSTR() should be ("do I subtract 1 here or not?"). So, not only will STREXTRACT save two lines of code, it'll also help me avoid the infamous "boundary" problem.

### SYS Functions

What would a new release of VFP be without a few new SYS() functions? Half of them provide features to COM servers, so I'll discuss those next month.

VFP 6 Service Pack 4 added a new setting configurable only in CONFIG.FPW: BITMAP. Normally, VFP updates the screen by drawing a bitmap in memory, then moving the entire bitmap to the screen at once. While this provides better video performance for normal applications, it slows down applications running under Citrix or Windows Terminal Server. Those environments look for changes to the screen and send them down the wire to the client system. Since every change causes the entire screen to be sent to the client, rather than just the changes made, this results in a lot of data being sent. Adding BITMAP=OFF turns off this screen drawing behavior. OK, that's the background. What's new to VFP 7 is SYS(602), which allows you to determine the value of this setting.

Technically, SYS(1104) isn't new (I'm not sure which release it was added in), but in VFP 7, it's finally documented. This function purges memory cached by programs and data. According to the help file, "you can improve performance by calling SYS(1104) after executing commands that make extensive use of memory buffers." I couldn't confirm this in informal testing I did, but I'm sure there are circumstances where this is useful.

Normally, VFP won't let you use the debugger in "system component" code, such as the Class Browser. According to the help file, SYS(2030) allows you to enable or disable debugging features within this code. Again, I was unable to confirm this. I was really hoping this would allow us to use the debugger to trace code in or called from a report, but that doesn't appear to be the case.

SYS(2600) isn't currently documented, but I ran across this function while spelunking in the FOXCODE table that provides Intellisense to VFP 7, and Microsoft was kind enough to forward some information about it to me. SYS(2600) either copies the contents of a specified range of memory addresses to a string or copies the contents of a string to memory. Readers familiar with BASIC will recognize these features as the equivalent of the PEEK() and POKE() functions (honestly, I didn't make those names up!). To see an example of this function, type GETENV and an open parenthesis in the Command window and notice the list of environment variables that appears. That list is read from memory using SYS(2600). The following code (TEST2600.PRG in the sample files) is similar, showing not only the variable names but their current values as well. This code uses HEX2DECIMAL.PRG to convert a hex string into a numeric value.

```
clear
declare integer GetModuleHandle in Win32API ;
  string
declare integer GetProcAddress in Win32API ;
  integer, string
lnHandle  = GetModuleHandle('msvcrt.dll')
lnPointer = GetProcAddress(lnHandle, '_environ')
lnPointer = Peek(lnPointer)
lnI       = 0
lnAddress = Peek(lnPointer)
do while lnAddress <> 0
```

```
  lnI = lnI + 1
  ? GetString(lnAddress)
  lnAddress = Peek(lnPointer + lnI * 4)
enddo while lnAddress <> 0

function GetString(tnAddress)
local lcReturn, ;
  lcChar, ;
  lnI
lcReturn = ''
lcChar   = sys(2600, tnAddress, 1)
lnI      = 0
do while asc(lcChar) <> 0
  lcReturn = lcReturn + lcChar
  lnI      = lnI + 1
  lcChar   = sys(2600, tnAddress + lnI, 1)
enddo while asc(lcChar) <> 0
return lcReturn

function Peek(tnAddress)
return Hex2Decimal(sys(2600, tnAddress, 4))
```

SYS(2800) enables or disables support for Microsoft Active Accessibility. I don't have Active Accessibility set up on my system, so I didn't test this function.

SYS(2801) extends event tracking to include Windows mouse and keyboard events. You can configure it to track VFP events only (the current behavior and the default setting), Windows events only, or both.

### TEXTMERGE()

The new TEXTMERGE() accepts a string and returns that string with any expressions evaluated. I discussed this function in some detail and presented a VFP 6 version in my December 2000 column ("Have Your Cake and Eat it Too").

### WDOCKABLE()

VFP 7 supports "dockable" system windows. A dockable window is like a toolbar in that it can be docked at any edge of the screen. The Command, Document View, Data Session, Properties, and various Debugger windows are all dockable. To visually change a window's current dockable state, right-click in its title bar and choose Dockable. If Dockable is checked, the window can be docked. The new WDOCKABLE() function returns the current dockable state of the specified window and can optionally change it.

### Conclusion

We're done looking at new general-purpose commands and functions. Next month, we'll start looking at COM server improvements in VFP 7, both in terms of new capabilities and some new related functions.

*Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Saskatchewan, Canada. He is the author of 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.*