Processing Whole Words

Doug Hennig

Visual FoxPro has many text processing functions, including ATC() and STRTRAN(). However, these functions suffer from one flaw: they are character-based rather than word-based, so they can find substrings you may not have wanted. In this article, Doug presents replacement functions that word on whole words.

We do a *lot* of text processing at Stonefield Software, especially with expressions. We use the built-in VFP text processing functions extensively, and usually they work well for us. However, sometimes they don't: when we want them to search for text as a word rather than any occurrence of the text.

ATC() replacement

One processing task we frequently do is look for the existence of a field in an expression. It's not as simple as it sounds. For example, suppose you have a SQL statement like this stored in a variable named lcSelect:

```
select Employees.FirstName,
       Employees.LastName,
       Employees.CountryOfBirth
    from ...
```

and you want to know if the Country field is involved in the query. Unfortunately, ATC('Country', lcSelect) returns a false positive because "Country" is contained within "CountryOfBirth." ATC looks anywhere in the string rather than looking at words.

One of our genius developers at Stonefield, Trevor Mansuy, wrote a replacement for ATC() called ATCWord that searches for words rather than substrings. So, using the example above, ATCWord('Country', lcSelect) returns 0.

Like ATC(), ATCWord expects three parameters: the string to search for, the string to search for a match in, and which occurrence to search for (optional: if it isn't passed, 1 is used). It returns the index of the first character of the match or 0 if there is no match. It uses the VBScript regular expression parser to do the hard work.

```
lparameters tcSearch, ;
  tcString, ;
  tnOccurrence
local lnOccurrence, ;
  lcLeftBoundary, ;
  lcRightBoundary, ;
  lcDelimiters, ;
  lcSearch, ;
  lcString, ;
  lcSearchString, ;
  loRegExpr, ;
  loMatches, ;
  lnReturn, ;
  loMatch, ;
  lcMatch

* Ensure the necessary parameters were passed.

assert vartype(tcSearch) = 'C' and ;
  not empty(tcSearch) ;
  message 'Invalid search string passed'
assert vartype(tcString) = 'C' ;
  message 'Invalid string passed'

* Bug out if the string is empty.

if empty(tcString)
  return 0
endif empty(tcString)

* Get the occurrence number.

lnOccurrence = ;
  iif(vartype(tnOccurrence) = 'N', ;
  tnOccurrence, 1)

* This is here for future proofing. Right now,
* the function expects whatever we search for
* to begin and end with non-word characters,
* but just in case, save the boundary
* characters in variables so we can change
* them easily. Right now, we're using word-
* boundaries (the space between a word and a
* non-word character). We originally wanted to
* use non-word characters (\W) for this, but
* the match consumes the character, so side-
* by-side matches won't be found. The drawback
* of \b is if we do an ATC() for a string with
* a non-word character at the beginning or
* end, there's no guarantee there's a word
* boundary on the other side of that
* character.

lcLeftBoundary  = '\b'
lcRightBoundary = '\b'

* Strip the usual name delimiters since they
* interfere with the search.

lcDelimiters = '"[]`'
lcSearch     = chrtran(tcSearch, ;
  lcDelimiters, '')
lcString     = chrtran(tcString, ;
  lcDelimiters, '')

* Certain RegExp special characters prevent
* matches, so escape them.

lcSearchString = AddRegExEscape(lcSearch)
```

```
lcString       = AddRegExEscape(lcString)

* In the pattern below, the three sets of
* parentheses represent three match groups.
* The first match group, (LB|^), means we
* first match either a non-word character or
* the beginning of a string. Similarly, (RB|$)
* means at the end we match either a non-word
* character or the end of the string. This
* means we make a match in every situation
* except where the string in lcSearch is a
* substring of one of the strings in tcString.

loRegExp = createobject('VBScript.RegExp')
loRegExp.IgnoreCase = .T.
loRegExp.Global     = .T.
loRegExp.Pattern    = '(' + lcLeftBoundary + ;
  '|^)(' + lcSearchString + ;
  ')(' + lcRightBoundary + '|$)'

* Test the string and check how many matches
* were made. If it's less than the occurrence
* passed, return 0.

loMatches = loRegExp.Execute(lcString)
if loMatches.Count < lnOccurrence
  lnReturn = 0

* Retrieve the specified occurrence from the
* Item collection. If the match is the one we
* want, return its start location in the
* string.

else
  loMatch  = loMatches.Item(lnOccurrence - 1)
  lcMatch  = loMatch.Value
  lnReturn = ;
    iif(upper(lcSearch) = upper(lcMatch), ;
    loMatch.FirstIndex + 1, 0)
endif loMatches.Count < lnOccurrence
return lnReturn
```

AddRegExEscape, called from ATCWord, escapes certain characters in regular expressions that prevent matches:

```
lparameters tcString
local lcString
lcString = strtran(tcString, '\', '\\')
lcString = strtran(lcString, '?', '\?')
lcString = strtran(lcString, '*', '\*')
lcString = strtran(lcString, '+', '\+')
lcString = strtran(lcString, '.', '\.')
lcString = strtran(lcString, '|', '\|')
lcString = strtran(lcString, '{', '\{')
lcString = strtran(lcString, '}', '\}')
lcString = strtran(lcString, '[', '\[')
lcString = strtran(lcString, ']', '\]')
lcString = strtran(lcString, '(', '\(')
lcString = strtran(lcString, ')', '\)')
lcString = strtran(lcString, '$', '\$')
return lcString
```

Here's an example that shows that ATC() gives a false positive while ATCWord correctly returns 0:

```
text to lcSelect noshow
select Employees.FirstName,
    Employees.LastName,
    Employees.CountryOfBirth
  from Employees
endtext
messagebox(atc('Country', lcSelect))
messagebox(ATCWord('Country', lcSelect))
```

STRTRAN() replacement

Another processing task we do frequently is replacing one field name with another. For example, in the following SQL statement, there are two fields named FirstName:

```
select Employees.FirstName,
       Customers.FirstName
    from ...
```

Rather than letting VFP create a unique name for these fields, we'd rather specify a name for the second one:

```
select Employees.FirstName,
       Customers.FirstName as FirstName_A
    from ...
```

You'd think STRTRAN() would make short work of this, but STRTRAN() has a similar problem to ATC(): it can find substring matches because it doesn't specifically handle words. To take care of this, Trevor wrote a function to replace STRTRAN() as well, called StrTranWord.

StrTranWord accepts three parameters: the string to search and replace in, the string to search for, and the string to replace the found string with. It returns the string with any replacements made. Unlike STRTRAN(), it doesn't accept the starting occurrence or the number of occurrences to replace parameters since we didn't need that functionality. It also doesn't accept the flags parameters, as it's always case-insensitive.

```
lparameters tcString, ;
  tcSearch, ;
  tcReplace
local loRegExp, ;
  lcSearch, ;
  lcReplace, ;
  loMatchCollection, ;
  lcLeftBoundary, ;
  lcRightBoundary, ;
  lcResult

* Ensure the necessary parameters were passed.

assert vartype(tcSearch) = 'C' and ;
  not empty(tcSearch) ;
  message 'Invalid search string passed'
assert vartype(tcString) = 'C' ;
  message 'Invalid string passed'
assert vartype(tcReplace) = 'C' ;
  message 'Invalid replacement string passed'

* Bug out if the string is empty.

if empty(tcString)
  return tcString
endif empty(tcString)

* If the search string begins or ends with a
* non-word character, this causes the string
* match to fail. In some cases, however, we
* want the match and replacement to be
* successful (i.e. TableName.FieldName,
* replacing "TableName." with "Replacement.").
* To handle this, we check the left and right
* sides of the search string for non-word
* characters and use them instead of \W if
```

```
* they exist. We also have to remove these
* characters from the search string.

loRegExp = createobject('VBScript.RegExp')
loRegExp.IgnoreCase = .T.
loRegExp.Global     = .T.
loRegExp.Pattern    = '\W'
lcSearch            = tcSearch
lcReplace           = tcReplace
loMatchCollection   =
loRegExp.Execute(left(tcSearch, 1))
if loMatchCollection.Count = 0
  lcLeftBoundary = '\W'
else
  lcLeftBoundary = ;
    AddRegExEscape(left(tcSearch, 1))
  lcSearch       = substr(lcSearch, 2)
endif loMatchCollection.Count = 0
loMatchCollection =
loRegExp.Execute(right(tcSearch, 1))
if loMatchCollection.Count = 0
  lcRightBoundary = '\W'
else
  lcRightBoundary = ;
    AddRegExEscape(right(tcSearch, 1))
  lcSearch        = left(lcSearch, ;
    len(lcSearch) - 1)
endif loMatchCollection.Count = 0

* Certain RegExp special characters prevent
* matches, so escape them.

lcSearch = AddRegExEscape(lcSearch)

* In the pattern below, the three sets of
* parentheses represent three match groups.
* The first match group, (\W|^), means we
* first match either a non-word character or
* the beginning of a string. Similarly, (\W|$)
* means at the end we match either a non-word
* character or the end of the string. This
* means we do the replacement in every
* situation except where the string in
* lcSearch is a substring of one of the
* strings in tcString.

loRegExp.Pattern = '(' + lcLeftBoundary + ;
  iif(lcLeftBoundary = '\W', '|^)(', ')(') + ;
  lcSearch + ')(' + lcRightBoundary + ;
  iif(lcRightBoundary = '\W', '|$)', ')')

* Do the replacement. The $1 and $3 refer to
* the first and third capturing groups in our
* RegEx. We add these in case the non-word
* match consumes a boundary character such as
* a space, an operator in an expression, etc.,
* and we want to put it back. If the first or
* third capturing groups captured a specific
* non-word character, ignore it during the
* replacement since a suitable character is
* assumed to be in the replacement string in
* this case.

lcResult = loRegExp.Replace(tcString, ;
  iif(lcLeftBoundary = '\W', '$1', '') + ;
  tcReplace + ;
  iif(lcRightBoundary = '\W', '$3', ''))
return lcResult
```

Here's an example that shows the difference between STRTRAN() and StrTranWord:

```
text to lcSelect noshow
select Employees.LastName,
    Employees.Tax,
    Employees.TaxStatus
  from Employees
endtext
messagebox(strtran(lcSelect, 'Tax', ;
  'Tax as Tax_A'))
messagebox(StrTranWord(lcSelect, 'Tax', ;
  'Tax as Tax_A'))
```

STRTRAN() replaces both instances of "Tax," giving:

```
select Employees.LastName,
    Employees.Tax as Tax_A,
    Employees.Tax as Tax_AStatus
  from Employees
```

which is not what we want, while StrTranWord gives the correct text:

```
select Employees.LastName,
    Employees.Tax as Tax_A,
    Employees.TaxStatus
  from Employees
```

Summary

VFP has some great text processing functions, but one of their shortcomings is that they aren't word-based. Fortunately, using regular expressions, we created replacement functions that provide the same functionality but handle whole words rather than substrings. Feel free to use this functions if you have a similar need.

Doug Hennig is a partner with 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 "VFPX: Open Source Treasure for the VFP Developer," "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." 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). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 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://vfpx.codeplex.com). He was a Microsoft Most Valuable Professional (MVP) from 1996 to 2011. Doug was awarded the 2006 FoxPro

Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).