April, 2002

Advisor Answers

Substituting variables into code

VFP 7/6/5/3 and FoxPro 2.x

Q: What's the rule for when to use macro substitution (&), when to use EVALUATE(), and when to use parentheses?

–David Birley (via CompuServe)

A: As in so many other areas, VFP provides several different ways to substitute values into code. All three approaches let you specify a variable or field name where FoxPro's syntax calls for something hard-coded.

Often, when there are such choices, you need to test in your production environment to figure out which technique to use. Happily, in this case, there are some pretty clear-cut rules that tell you which approach to use.

Using parentheses to substitute for a value is called "indirect reference" or a "name expression." It works only when FoxPro expects the name of something. So, for example, you can use this technique to specify the name or alias of a table in a USE command, the name of an index in SET ORDER or USE, a filename where information should be stored by a LIST command, a cursor name in any of FoxPro's SQL commands (such as SELECT or INSERT), and a number of other places.

For example, you might have code that can operate on any of a number of different tables, with some logic that determines which table to use at any given time. If you store the name of the desired table in cTable, you can open it with this code:

```
USE (cTable) IN 0
```

In practice, I always include an ALIAS clause when opening a table this way, so that I can refer to the right work area without having to do anything complex. So, my code is more likely to look something like this:

```
USE (cTable) AGAIN IN 0 ALIAS SomeAlias
```

Then, subsequent code that operates on the table can refer to it using SomeAlias, like this:

```
SELECT SomeAlias
```

Another situation in which name expressions are particularly useful is when users are allowed to choose an object to work with or enter a filename. For example, you might allow your users to choose a report to run, storing the name of the specified report in a variable called cReport. Then, to actually run the report, you can use code like:

```
REPORT FORM (cReport) TO PRINT
```

In general, if the item for which you're substituting is the name of something, try a name expression first. Sometimes, however, you need to substitute for something more complex.

The next level up is substituting for an expression. When you have an expression that needs to be evaluated before proceeding, use EVALUATE() (frequently abbreviated as EVAL()).

I don't use EVALUATE() as often as I use either name expressions or macros; the most common place I use it is to get an object reference when I have the name of a contained object.

For example, I have a builder that adds a new control to a container. If a reference to the container is stored in oContainer, and the new control's name is stored in cNewControl, I can get an object reference to the new control like this:

```
oNewControl = EVALUATE("oContainer." + cNewControl)
```

However, as Christof points out, you can do this much faster using GetPem():

```
oNewControl = GetPem( oContainer, cNewControl
```

EVALUATE() is useful in reports - you can use it both in field expressions and in group expressions. For example, suppose you have a report that you'd like to be able to group in a couple of different ways, but that's otherwise identical. You can specify the grouping expression as:

```
EVALUATE(cGroupExpr)
```

and then, make sure that your code that runs the report sets cGroupExpr to the appropriate value.

Finally, there are times when you want to substitute for a keyword or a clause or even a whole command. In that case, you need to use the macro operator (&).

Perhaps the most common use for & is to restore settings. For example, if you want to change the SAFETY setting that determines whether the user is prompted when files are overwritten, you might use code like this:

```
LOCAL cOldSafety
* Save the original setting
cOldSafety = SET("SAFETY")

SET SAFETY OFF
* Do whatever you need to do

* Restore the original setting
SET SAFETY &cOldSafety
```

In this example, the macro operator converts the string (either "ON" or "OFF") to a keyword. But sometimes, you need the macro operator even if what you're substituting isn't a keyword. For example, if you want to set the search path and have any expressions in the list of directories, you need to use a macro because name expressions aren't evaluated:

```
LOCAL cPath

cPath = HOME(2)+"TasTrade\Data"
cPath = cPath + ";" + HOME()

SET PATH TO &cPath
```

Macros also let you assemble complex commands in pieces. For example, when creating queries that vary at runtime, many people build them up one clause at a time, with code like this:

```
cFieldList = "FirstName, LastName"
cWhere = "Country = 'USA' AND Active"

SELECT &cFieldList ;
   FROM Customers ;
   WHERE &cWhere ;
   INTO CURSOR Results
```

This piecemeal approach is especially useful when the entire query is too long for a single string literal.

Name expressions work within some of the clauses of SELECT. For example, if you have a variable that holds the name of a field for the

field list, you can use a name expression, as you can for individual tables in the FROM list:

```
cFieldToSelect = "Company"
cTable = "Customers"

SELECT iID, (cFieldToSelect) AS cName ;
   FROM (cTable) ;
   INTO CURSOR Results
```

Macros have a couple of twists. First, only variables can be macro-expanded, not fields or properties. When the value you want to substitute is stored in a field or property, you have to copy it to a variable first. For example, if you've stored the previous path in a property called cPath of the current object, you can restore it like this:

```
LOCAL cOldPath
cOldPath = This.cPath
SET PATH TO &cOldPath
```

Be aware, also, that you can't use the "m." notation for variables that you're macro-expanding. That is, in the previous example, you can't say:

```
SET PATH TO &m.cOldPath
```

That's because the period is seen as a terminator for the item to be expanded. So, FoxPro reads the previous line as asking you to expand m, then tack that result in front of the string "cOldPath" before executing the command.

The period terminator means that, occasionally, you'll run into code that has two periods in a row. For example, you might see something like:

```
REPLACE SomeField WITH &cAlias..SomeOtherField ;
  IN SomeTable
```

However, I find I rarely need to use that notation anymore. I can almost always write the command in another, more readable, way. For instance, the example above could be written as:

```
uNewValue = EVALUATE(cAlias + ".SomeOtherField")
REPLACE SomeField WITH uNewValue IN SomeTable
```

Or:

```
REPLACE SomeField WITH &cAlias->SomeOtherField IN SomeTable
```

The second example uses the older "->" notation for referencing a field in an alias. It avoids the ambiguity of the period, which has several different meanings in VFP.

By now, you're probably wondering why there are three ways to substitute into a command. The answer is performance. Macros have been in the Xbase language since time immemorial. However, they can be slow, especially within loops and so forth. While a single macro expansion isn't likely to bog down your code, if you make a practice of writing code like:

```
USE &cTable ORDER &cOrder
```

you're definitely handicapping your application. A name expression will be faster than a macro or EVALUATE(), every single time. That's because FoxPro doesn't have to work as hard to evaluate a name expression – you're already providing a lot of information by using the parentheses instead of one of the other approaches.

Generally, EVALUATE() is faster than a macro. In reports, you have to use EVALUATE() because macros aren't available. So, when you have a choice, use EVALUATE(). Finally, use macros when nothing else will do.

–Tamar