January, 2006

Advisor Answers

Comparing logical values

VFP 9/8/7

Q: I've noticed that there are two different styles for checking the value of a logical field or variable. Some people compare it to .T. or .F., while others just check the value or use the NOT operator. Does it make a difference which way you do it?

A: You've hit on one of my pet peeves. Reading code that checks:

```
IF lFlag = .T.
```

really annoys me. From a logical perspective, that line is equivalent to:

```
IF lFlag
```

Similarly, these three lines all have the same result:

```
IF lFlag = .F.
IF NOT lFlag
IF !lFlag
```

But your question made me consider whether there's any internal difference. So I wrote a program to see whether one way is faster than the others. I checked the five cases shown above, testing each in a loop when lFlag was .T. and again when it was .F.

I found no significant differences between equivalent forms. However, interestingly, it appears that whichever case is false (that is, where the variable doesn't have the value we're looking for) takes slightly longer. That makes some sense, since in that case, VFP has to check for an ELSE clause. When the condition is met, that test is unnecessary.

I tested in VFP 9, VFP 8 SP1 and VFP 7 SP1. I was dismayed to find that VFP 9 took about half again as long for each test as the other two versions. (VFP 7 was a little faster than VFP 8, as well.) It's hard to imagine what could have changed between versions for such a simple test. My test code is included on this month's Professional Resource CD as TestLogicalFlags.PRG, so you can try it yourself.

Bottom line: it appears that which form you use for logical tests is a matter of personal preference (and perhaps, of your company's coding

standards). That said, I'll make an impassioned plea for you to avoid the ! for NOT. It's just too hard to read and too easy to overlook.

Testing this case made me wonder about assignments to logical variables. When you're setting a flag based on an expression, you can write the code in one of two ways:

```
IF Expr
   lFlag = .T.
ELSE
   lFlag = .F.
ENDIF
```

or simply:

```
lFlag = Expr
```

I tested using a simple numeric comparison for the expression.

In this case, the choice does have performance consequences, though they vary from version to version. In VFP 9, using IF-ELSE for the assignment took about 50% longer than the direct assignment. In VFP 7 and VFP 8, the difference was between 15% and 20%. As with testing a flag, VFP 7 was the fastest, but the difference between direct assignment in the three versions wasn't particularly large. My test code for this case is shown below and included on the PRD as TestLogicalAssignment.PRG.

```
#DEFINE PASSES 1000000

LOCAL lFlag, nPass, nStart, nEnd, nValPass, nTestVal

FOR nValPass = 1 TO 2

   IF nValPass = 1
      nTestVal = 437
   ELSE
      nTestVal = 0
   ENDIF

   nStart = SECONDS()
   FOR nPass = 1 TO PASSES
      IF nTestVal > 100
         lFlag = .T.
      ELSE
         lFlag = .F.
      ENDIF
   ENDFOR
   nEnd = SECONDS()

   ?"With IF-ELSE and expression = ", nTestVal > 100, ;
```

```
    PASSES, "passes = ", nEnd-nStart

   nStart = SECONDS()
   FOR nPass = 1 TO PASSES
      lFlag = nTestVal > 100
   ENDFOR
   nEnd = SECONDS()

   ?"With direct assignment and expression = ", ;
    nTestVal > 100, PASSES, "passes = ", nEnd-nStart

ENDFOR
```

Deciding which approach to use in this case is harder. Many people find the IF-ELSE version of assignment much easier to read. (In fact, my code for the comparison test includes such a block.) My sense is that the differences in real time are so small that you should go with the version you find easier to read and maintain unless you're in a situation where you need to squeeze every drop of performance out of your code.

As my code demonstrates, doing quick performance tests isn't very hard. Just wrap the code you want to test in a loop, checking SECONDS() before and after the loop. (In the code here, there are two loops. The outer loop varies the value of the expression, while the inner loop is a pass counter.) Then figure out how many passes you need to get measurable results. The more, slower, code you're executing in each pass, the fewer passes you need. (Christof offers an alternative approach—loop for a fixed number of seconds and count how many iterations you complete. Keep in mind that the resolution of the SECONDS() is 10 ms, so you need at least one second of testing for accuracy.)

Keep in mind that other things going on your computer can interfere with your results. To get truly accurate results, you need to turn off your virus scanner, tell your email client not to check for mail, and so forth before testing. Running your tests enough times that you see consistent results helps, too. Of course, this kind of testing is meant principally to guide you in programming choices; it's not real scientific research. For that, you'd need to use a separate test machine, on which you could reset the environment after each test.

–Tamar