Working with Work Areas

Tamar E. Granor

Although we've had techniques that let us ignore work area numbers and letters for many versions, some developers still write code that addresses work areas directly. This month, I'll look at how to write code without worrying about work area letters or numbers, and how to depend as little as possible on the currently selected work area. The result is better code that's easier to write and maintain.

In dBase II, you could open two tables simultaneously. Each occupied one work area. By the time I entered the Xbase world with FoxBase+, there were 10 work areas; most of the time, that was enough, but now and then, I found it limiting. FoxPro had 25 work areas in the standard version and 225 with the extended version; I'm not sure I ever found myself short of work areas, even in the standard version. When VFP shipped with 32,767 work areas per data session, I knew I'd never worry about available work areas again.

When there were only two work areas or even 10, keeping track of what table was open in which work area was no problem. But as soon as there were more work areas than I could keep track of in my head, I started making sure I didn't need to know where a given table was open.

The tools for letting us ignore work areas have gotten better and better over the years. There are three related issues. The first is to open a table without having to figure out what work area to use. The second is addressing an open table without knowing what work area it's in. The third is writing code without having to worry about what work area is current. Combining the three means you'll never worry about stepping on an open table or acting on the wrong table again.

Use aliases, not work area numbers or letters

Every time you open a table in VFP, the open table occupies a work area. As noted above, in VFP (all versions from 3 to 9), each data session offers 32,767 work areas. Work areas are numbered from 1 to 32,767 (there's also a hidden work area 0, used for system tables, but that's a story for another article). The first 10 work areas can also be referenced by the letters A through J.

When you USE a table, by default, it opens in the current work area. If there's already a table open in that work area, the open table is closed. Once a table is open in a given work area, you can refer to the work area by the alias of the open table. This is always a better approach than using the work area letter or number, since it lets your code express intent rather than structure. For example, if you know that the Customers table is open in work area 3, you can move to that work area with any of the following lines of code:

```
* Version 1
SELECT 3

or:

* Version 2
SELECT C

or:

* Version 3
SELECT Customers
```

The first two versions tell you very little; to understand them, you have to know what table is open in work area 3/C. The third version, though, tells you that you're moving to the work area for the Customer table (in fact, there's rarely a reason to select a particular work area anymore; I'll talk about that later in this article).

Open tables without thinking about work areas

While using the alias rather than the work area number in your code is handy, it wouldn't be all that powerful if you still had to remember which work areas are available every time you wanted to open a table. Fortunately, you can avoid that, as well.

VFP (and FoxPro before it) offers two ways to find an available work area. SELECT 0 moves to the lowest available work area, while the function call SELECT(1) returns the number of the highest available work area. You can be sure you're opening a table in an unused work area by issuing either of the following sequences:

```
* Version 1
SELECT 0
USE YourTable

or:

* Version 2
SELECT SELECT(1)
USE YourTable
```

In fact, you can consolidate either of those into one line of code, with one difference in behavior. The USE command, like many of VFP's Xbase commands, accepts the IN clause to specify the work area. When you specify IN 0, VFP opens the table in the lowest available work area:

```
* Version 3
USE YourTable IN 0
```

Similarly, if you specify IN SELECT(1), VFP opens the table in the highest available work area, like this:

```
* Version 4
USE YourTable IN SELECT(1)
```

The difference between the one-line versions 3 and 4 and the two-line sequences of versions 1 and 2 is that after the one-liners, you haven't changed work areas. You're still in the same work area you were in before opening the table. Often, that doesn't matter, especially if you're opening a series of tables. Consider these two blocks of code:

```
* Version 1
SELECT 1
USE Customers
SELECT 2
USE Orders
SELECT 3
USE OrderDetails

SELECT 1
* Do something with customer records

* Version 2
USE Customers IN 0
USE Orders IN 0
USE OrderDetails IN 0

SELECT Customers
* Do something with customer records
```

Not only is version 2 shorter and easier to read, but it expresses intent more clearly than version 1.

Use IN rather than SELECT

With the techniques above, you never have to think about work area numbers again, nor keep a chart to show you which table is open in which work area (I used to do that to ensure I didn't make mistakes). That's a major win, but you can do better.

Many of VFP's commands and functions operate in the current work area. If you forget to SELECT the right work area, or you do so but then something changes the work area before your code runs, you can get nasty, unexpected results. Fortunately, many of those commands now accept an IN clause to indicate which work area to operate on. Similarly, most of the functions accept a parameter to indicate the work area.

When you use the IN clause or pass the parameter, you ensure that the command or function operates where you want it to; no event, ON KEY LABEL or user action (such as clicking on a grid) can switch work areas on you. In addition, you don't need to save the current work area and switch to the desired work area, then restore the old work area after doing whatever you need.

Consider this code:

```
* Version 1
nOldSelect = SELECT()
SELECT Orders

CALCULATE SUM(Freight) TO nTotalFreight

SELECT (nOldSelect)
```

Four lines of code to do a simple calculation (full disclosure: I don't think I've ever used the CALCULATE command in an application; it's just convenient for this demonstration. One good place to use it though is instead of the COUNT, SUM and AVERAGE commands, as those don't support the IN clause). You can do it in one line, if you use the IN clause.

```
* Version 2
CALCULATE SUM(Freight) TO nTotalFreight ;
        IN Orders
```

Similarly, look at the difference when searching for a particular record with the SEEK() function:

```
* Version 1
nOldSelect = SELECT()
SELECT Orders

cOldOrder = ORDER()
SET ORDER TO OrderDate

IF SEEK({^ 1998-04-24})
   * Process records for this date
ENDIF

SET ORDER TO (cOldOrder)
SELECT (nOldSelect)
```

Using the optional alias and order parameters of the function, we can eliminate six lines of code and decrease the risk of error:

```
* Version 2
IF SEEK({^ 1998-04-24}, "Orders", OrderDate")
   * Process records for this date
ENDIF
```

Behind the scenes, of course, VFP does change work areas, but it handles all the details of making the change and restoring the original set-up. Keep this in mind, though, as you write code; commands that use IN and functions that pass the optional alias parameter must assume that they're executing in the specified work area.

The special case of REPLACE

While using the IN clause is convenient for most Xbase commands, for REPLACE, it's virtually required. First, REPLACE is destructive; changing data in the wrong area can have serious consequences. Second, REPLACE has a behavior that surprises many VFP developers; if you're at EOF in the current work area, nothing gets replaced.

When looking at other people's code, I often see lines like this:

```
* Don't write code like this
REPLACE Orders.Freight WITH m.nFreightTotal
```

If the current work area is Orders, that line behaves as you'd expect, substituting the value of nFreightTotal into the Freight field of the current work area. In this situation, the alias Orders is simply unnecessary.

What happens if the current work area is something other than Orders? That depends on what's open in the current work area. If there's no table open in the current area, the user is prompted to open a table; if he cancels that dialog, error 52, „No table is open in the current work area" fires. If the user picks a table to open, and that table has any records, the replacement succeeds. If the user picks a table to open, and the table is empty, the replace fails. Clearly, none of these results is desirable, since we certainly don't want the user prompted to open a table.

If there is a table open in the current work area, the results depend on the position of the record pointer. If it's pointing to a valid record, the replacement succeeds. If the record pointer is at the end-of-file marker, the replacement silently fails to take place. It's easy to see that, even though the REPLACE command shown above might succeed, it's pretty risky as written.

Why does VFP behave this way? Because REPLACE, like most other Xbase commands, is scoped to the current work area. If you don't specify otherwise, REPLACE is understood as being REPLACE NEXT 1 in the current work area. For comparison, consider the DELETE command, which also has a default scope of NEXT 1. You'd never issue DELETE with one work area selected, and expect it to delete a record in another work area, without including the IN clause. And, if the current work area is at EOF, you wouldn't expect DELETE to delete any records.

What makes REPLACE so confusing is its ability to replace fields in multiple tables at one time. For example, this is a valid (if dangerous) command:

```
REPLACE Person.cLast WITH m.cLast, ;
        Person.cFirst WITH m.cFirst, ;
        Address.cStreetAddr WITH m.cStreetAddrz ;
        Address.cCity WITH m.cCity, ;
        Address.cState WITH m.cState, ;
        Person.cEmail WITH m.cEmail
```

This command affects the current record in both Person and Address. But what if one of those tables is at EOF? VFP's rules determine that it's the current work area that matters most. If you're at EOF there, the REPLACE doesn't take place at all;

otherwise, it makes the attempt. However, if any of the other tables are at EOF, nothing happens in that table.

These risks mean that REPLACE should never be used to change records in more than one table at once and that it should always include the IN clause. Consider this variation of the command at the beginning of this section:

```
* Always use IN with REPLACE
REPLACE Freight WITH m.nFreightTotal IN Orders
```

There's no ambiguity here. The Freight field in the Orders table will be updated as long as Orders is not at EOF. No errors, no dialogs appearing to the user (unless Orders is not open, in which case this is a developer error), no failure based on the state of any other table.

For the more complex replace above, two commands would be better:

```
REPLACE cLast WITH m.cLast, ;
        cFirst WITH m.cFirst, ;
        cEmail WITH m.cEmail ;
   IN Person
REPLACE cStreetAddr WITH m.cStreetAddr, ;
        cCity WITH m.cCity, ;
        cState WITH m.cState ;
   IN Address
```

Where you can use IN

Not every Xbase command supports the IN clause. However, the ones that do include the ones you're most likely to use in applications. IN is supported for APPEND, BLANK, CALCULATE, DELETE, DISPLAY/LIST STRUCTURE, FLUSH, GO, PACK, RECALL, REPLACE, SEEK, SET FILTER, SET ORDER, SET RELATION, SKIP, UNLOCK, USE and ZAP.

In addition, virtually all of the functions that operate on the current work area by default include an optional alias parameter. Among them are AFIELDS(), ALIAS(), ATagInfo(), BOF(), EOF(), FILTER(), FLOCK(), FOR(), FOUND(), LOCK(), ORDER(), RECCOUNT(), RECNO(), RLOCK(), SEEK(), TAG() and USED(). For a complete list of functions that accept an alias parameter, search the VFP help using the string „nWorkArea | cTableAlias."

Unfortunately, some Xbase commands don't let you specify the IN clause. The two where this is the biggest issue for me are LOCATE and SCAN. When you use either of these, or other commands or functions that depend on the current work area, take precautions to ensure that you're where you think you are and that you leave things as you found them.

For all commands and functions that let you specify the alias, do so every single time. Your code will be more reliable and its purpose will be clearer.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. She currently focuses on working with other developers through consulting and subcontracting. Tamar is author or co-author of nine books including the award winning Hacker's Guide to Visual FoxPro and Microsoft Office Automation with Visual FoxPro. Her most recent books are Taming Visual FoxPro's SQL and What's New in Nine: Visual FoxPro's Latest Hits. Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar is a Microsoft Certified Professional and a Microsoft Support Most Valuable Professional. Tamar speaks frequently about Visual FoxPro at conferences and user groups in North America and Europe, including every FoxPro DevCon since 1993. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com