November, 2007

Advisor Answers

NODEFAULT and DODEFAULT()

VFP 9/8/7

Q: I'm having trouble understanding the NODEFAULT and DODEFAULT() commands. Are they opposites? When would you use them?

A: NODEFAULT and DODEFAULT() are actually keywords rather than commands and can be used only inside class code. Both of them give you control over what code gets executed.

Let's start with DODEFAULT(). In VFP, code in a method of a subclass supersedes code in the same method of the parent class. For example, consider this class definition:

```
DEFINE CLASS TopLevel AS Custom

PROCEDURE DoSomething

WAIT WINDOW "This is parent class code."

ENDPROC

ENDDEFINE
```

If you issue the following commands, as you'd expect, you see a WAIT WINDOW that says "This is parent class code.":

```
o = CREATEOBJECT("TopLevel")
o.DoSomething()
```

Now suppose you create the following subclass of TopLevel:

```
DEFINE CLASS ChildOverride AS TopLevel

PROCEDURE DoSomething

WAIT WINDOW "This is child class code."

ENDPROC

ENDDEFINE
```

What happens when you instantiate ChildOverride and call the DoSomething method?

```
o = CREATEOBJECT("ChildOverride")
o.DoSomething()
```

You get a WAIT WINDOW that says "This is child class code." The code in ChildOverride.DoSomething overrides the code in TopLevel.DoSomething.

Usually, you don't want to simply override code this way. After all, if the parent class has code in a particular method, it should be because the code does something that every instance of the class and its subclasses needs to do. If it isn't something every instance needs to do, the code probably shouldn't be in the parent class.

But what if you need a subclass to do more than the parent class does in a particular method? You want to extend the parent class's code. That's where DODEFAULT() comes in. When you issue DODEFAULT in a method, you're telling VFP to call up the class hierarchy and execute the parent class's code for the current method.

Consider this subclass of the TopLevel class:

```
DEFINE CLASS ChildExtend AS TopLevel

PROCEDURE DoSomething

DODEFAULT()
WAIT WINDOW "This is child class code."

ENDPROC

ENDDEFINE
```

When you instantiate this class and call the DoSomething method, you get both WAIT WINDOWs, one after the other:

```
o = CREATEOBJECT("ChildExtend")
o.DoSomething()
```

You can also control when the parent class code is executed. Just put DODEFAULT() in the appropriate place in the child class method. For example, in this class, the child's WAIT WINDOW is displayed first:

```
DEFINE CLASS ChildExtendBefore AS TopLevel

PROCEDURE DoSomething
```

```
WAIT WINDOW "This is child class code."
DODEFAULT()

ENDPROC

ENDDEFINE
```

Although there are good techniques that help you avoid using it, DODEFAULT() is useful in many classes. For example, in a project I'm working on, the Init method of the base form class (frmBase) calls a number of other methods to do things like properly size and position the form. I also have a subclass of that base form class (frmReport) that's used for running reports. That class needs to do a number of other things in Init, such as prompting the user to save unsaved data before reporting. But it also still needs to be sized and positioned. So frmReport.Init issues DODEFAULT() to ensure that all the normal form behavior for this application occurs, in addition to the particular things relevant to reporting.

One final note on DODEFAULT(). Using it is like a function call. You need to pass along any parameters the parent class's method expects to receive.

Like DODEFAULT(), NODEFAULT gives you control over what happens in your application, but it applies specifically to events and to a few methods that behave like events. A number of the events associated with VFP's forms and controls have some built-in behavior. For example, the Keyboard event puts the key code for the character that was typed into the keyboard buffer. When the OpenTables method fires, it opens the tables and views in a form's Data Environment.

Fortunately, putting code into these methods doesn't prevent this native behavior from occurring. However, there are times when you want to suppress the native behavior. That's what NODEFAULT does.

For example, I've written a QuickFill combo class (published in the September '98 issue of FoxPro Advisor) that behaves like the comboboxes in Quicken. Each time the user presses a key, my code checks the keystroke and adjusts what's displayed in the combo. When my code changes the combo contents, I don't want the key the user typed added to the combo's value. So I use NODEFAULT to prevent the keystroke from going to the keyboard buffer.

Not every event responds to NODEFAULT. For example, putting NODEFAULT in Click doesn't prevent you from seeing a button go down and back up when you click it. In general, NODEFAULT is useful for

events that have something going on behind the scenes, not just something happening visually.

Here are some of the events and methods that respond to NODEFAULT:

- BeforeBuild–prevents the project from being built;
- BeforeRowColChange–prevents the row and/or column in the grid from changing;
- CloseTables—prevents tables in the DE from being closed;
- Deleted—prevents the record from being deleted;
- KeyPress–prevents the keystroke from being added to the buffer;
- LostFocus—prevents the control from losing focus;
- MouseDown/MouseUp—prevents clicks;
- OLEDragDrop—prevents normal drag-and-drop behavior;
- OpenTables–prevents the tables in the DE from being opened;
- QueryAddFile/QueryModifyFile/QueryNewFile/QueryRemoveFile/ QueryRunFile—prevents the specified file action from occurring;
- QueryUnload—prevents the form from being destroyed;
- WhatsThisMode—prevents "What's This Help" from firing.

Like any other keyword, NODEFAULT has no effect unless it's executed. This is actually really useful because it means you can test for certain conditions and issue NODEFAULT when they occur. For example, many people like to use code like this in QueryUnload:

```
IF MESSAGEBOX("Do you really want to quit?, 4+32) <> 6 && 6=yes
   NODEFAULT
ENDIF
```

This code prompts the user to confirm that the form is to be closed. (I have to add that I think most confirmation dialogs like this are simply a waste of the user's time and good will. I put this stuff into applications only when the client insists.)

There are times when you can profitably use both DODEFAULT() and NODEFAULT in the same method. The combination lets you change the point at which the native behavior of an event or method occurs. Normally, the native behavior of events and methods occurs at the very end, after any custom code has been executed. However, issuing DODEFAULT() tells VFP calls up the inheritance chain; when VFP reaches the end of the inherited method, the native behavior occurs. Adding NODEFAULT, then, prevents the native behavior from happening a second time.

When would do this? When you want the native behavior, but you also want to do something in code after the native behavior. For example, you might use this technique in OpenTables to create indexes after opening some views, like this:

```
* OpenTables method
* Get the tables to open right away
DODEFAULT()
* Prevent a second attempt at opening them
NODEFAULT

* Now do some indexing
IF USED("MyView")
   SELECT MyView
   INDEX ON MyField TAG MyField
ENDIF

RETURN
```

Having NODEFAULT and DODEFAULT() in your arsenal helps you write better code, with less cut-and-paste. Spend some time experimenting with them (especially to see where NODEFAULT does something and where it doesn't).

–Tamar

Upgrade to VFP 9 SP2?

VFP 9

Q: I know Microsoft released Service Pack 2 for VFP 9, but I hear there are some problems with it. Should I upgrade? If so, how do I do it?

A: This is a hard question. As you note, Service Pack 2 for VFP 9 was released back in October. It fixes a lot of bugs, some of which have been in the product for a long time. However, it introduces some problems, as well.

One problem was noted immediately. The splash screen was wrong. Microsoft fixed that, and re-released the service pack very quickly. If you choose to install it, make sure you get the October 16, 2007 version. (When you install it, the version number is 5815.)

Some of the new problems are easy to work around, others are not. The VFP community has been documenting the issues on the FoxPro Wiki at http://fox.wikis.com/wc.dll?Wiki~VFP9SP2BugList~Wiki. (It's worth noting that some of the items may not be SP2-specific, but

rather may have been discovered because lots of people installed SP2.) That article links to solutions or workarounds for some of them.

You can also find VFP bugs on Microsoft's Connect site. Go to connect.microsoft.com and choose "Visual Studio and .NET framework". To find FoxPro items, search for "foxpro or VFP" (without the quotes). This site is the place for you to report any bugs you find, as well.

In my view, the most serious issues are the reporting issues discussed by Cathy Pountney in her blog at http://cathypountney.blogspot.com/2007/11/gotcha-serious-report-bug-with-data.html, but you may find that other issues are more significant for you.

There is some good news in all this. Besides the bug fixes, SP2 includes some enhancements to the Report Designer. However, those are actually in the reporting APPs (ReportBuilder.APP, ReportOutput.APP and ReportPreview.APP). You can use the updated APP files with VFP 9 SP1; just copy the new files into the right folder.

In addition, Sedna has not been released yet. This collection of extensions to VFP 9 includes the My workspace, the improved Data Explorer, the new Upsizing Wizard and several other items. Again, these should work with VFP 9 SP1. Similarly, the controls DBI Technologies is offering free to registered VFP 9 developers will work without SP2.

It's my sense that many VFP developers are not installing SP2. I haven't done so on my primary machine yet, though I do have it installed on another machine. For now, I'm keeping my client projects in SP1, though I really want access to the Vista fixes that SP2 includes.

Given Microsoft's previously announced plan to end VFP development with SP2 and Sedna, I had hoped for a really solid release. Since that doesn't seem to be the case, I hope that Microsoft will rethink and decide to resolve the serious problems so that the final version of VFP is one we can all standardize on.

If you want to install SP2, you'll get the best results by following these steps:

1. Uninstall whatever version of VFP 9 you're now running, whether it's SP1 or one of the CTPs or Betas for SP2.

2. Install VFP 9.
3. Do NOT install SP1.
4. Install SP2.

–Tamar